19th SC@RUG 2022 proceedings 2021-2022

Smedinga, Rein; Biehl, Michael

university of groningen

faculty of science and engineering

computing science

SC@RUG 2022 proceedings

19th SC@RUG 2021-2022

Rein Smedinga, Michael Biehl (editors)

rug.nl/research/bernoulli

SC@RUG 2022 proceedings

Rein Smedinga
Michael Biehl
editors

2022
Groningen

ISBN (e-pub): 978-94-034-2971-7
Publisher: University of Groningen
Title: 19th SC@RUG proceedings 2021-2022
Computing Science, University of Groningen
NUR-code: 980

About SC@RUG 2022

Introduction

SC@RUG (or student colloquium in full) is a course that master students in computing science follow in the first year of their master study at the University of Groningen.

SC@RUG was organized as a conference for the 19th time in the academic year 2021-2022. Students wrote a paper, participated in the review process and gave a presentation.

SC@RUG is organized by Rein Smedinga and Michael Biehl, both from the Bernoulli institute. Renée Lutke (School of Science and Engineering) helped with improving the presentaiton skills of the students.

Organizational matters

SC@RUG 2022 was organized as follows:
Students were expected to work in teams of two. The student teams could choose between different sets of papers, that were made available through the digital learning environment of the university, *Nestor*. Each set of papers consisted of about three papers about the same subject (within Computing Science). Some sets of papers contained conflicting opinions. Students were instructed to write a survey paper about the given subject including the different approaches discussed in the papers. They should compare the theory in each of the papers in the set and draw their own conclusions, potentially based on additional research of their own.

After submission of the papers, each student was assigned one paper to review using a standard review form. The staff member who had provided the set of papers was also asked to fill in such a form. Thus, each paper was reviewed three times (twice by peer reviewers and once by the expert reviewer). Each review form was made available to the authors through *Nestor*.

All papers could be rewritten and resubmitted, also taking into account the comments and suggestions from the reviews. After resubmission each reviewer was asked to re-review the same paper and to conclude whether the paper had improved. Re-reviewers could accept or reject a paper. All accepted papers[1] can be found in these proceedings.

In his lecture about communication in science, Rein Smedinga explained how researchers communicate their findings during conferences by delivering a compelling storyline supported with cleverly designed graphics. Lectures on how to write a paper, on scientific integrity and on the review process were given by Michael Biehl

Renée Lutke gave tutorials about presentation techniques and speech skills.

Students were asked to give a short presentation halfway through the period. The aim of this so-called two-minute madness was to advertise the full presentation and at the same time offer the speakers the opportunity to practice speaking in front of an audience. Renée Lutke and Rein Smedinga were present during these presentations.

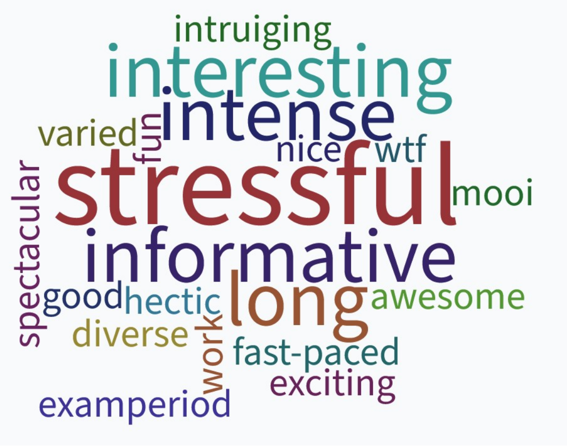
Renée Lutke gave tutorials in small groups to further practice presentation skills.

The final online conference was organized by the students themselves (from each author-pair, one was selected to be part of the organization and the other doing the chairing of one of the presentations). Students organized the conference by setting up the final program, find a sponsor for the breaks, etc. They also found a keynote speaker, **Jeroen Brandsma** and **Ana Roman** from **Belsimpel** who spoke about *Scaling under load*. The organizing students also created a website for this years conference, to be found on **https://www.studentcolloquium.nl/2022/**

The overall coordination and administration was taken care of by Rein Smedinga, who also served as the main manager of *Nestor*.

Students were graded on the writing process, the review process and the 2 minute madness presentation, the presentation during the conference and on their contribution in the organization of this conference.

For the grading of the 2 minute mandess presentations we used the assessments of the audience using the application *Poll Everywhere* and also used this application to find the best presentation of the day according to the audience. For the presentations during the conference we also used *Poll Everywhere* for the assessments of the audiencee (for 50%) and the assessments of Renée Lutke and Rein Smedinga (also for 50%). Poll Everywhere again was used to find the best presentation of the day and to ask the audience about their general finding of the symposium, resulting in the following outcome:



[1] this year, all papers were accepted

The gradings of the draft and final paper were weighted marks of the review of the corresponding staff member (50%) and the two students reviews (25% each).

The complete conference was also recorded and this recording was published on *Nestor* for self reflection.

The best 2 miniute madness presentation, the best conference proesentation and the best paper were awarded with a voucher and mentioned in the hall of fame.

Website

Since 2013, there is a website for the conference, see **www.studentcolloquium.n**l.

Thanks

We could not have achieved the ambitious goals of this course without the invaluable help of the following expert reviewers:

- Andrés Tello
- Asad Shahbahrami
- Bochra Boughzala
- Boris Koldehofe
- Fadi Mohsen
- George Azzopardi
- Heerko Groefsema
- Jiapan Guo
- Kerstin Bunte
- Majid Lotfian Delouee
- Michael Biehl
- Michael Wilkinson
- Michiel Medema
- Mirela Riveni
- Mostafa Hadadian
- Saad Saleh
- Vasilios Adrikopoulos

and all other staff members who provided topics and sets of papers.
. Also, the organizers would like to thank Renée Lutke for helping with the presentation skills and the *Graduate school of Science and Engineering* for making it possible to publish these proceedings and sponsoring the awards for best presentations and best paper for this conference and our symposium sponsor Belsimpel for providing our keynote presentation and the lunch during lunch break.

Rein Smedinga
Michael Biehl

Since the tenth SC@RUG in 2013 we added a new element: the awards for best presentation, best paper and best 2 minute madness.

Best 2 minute madness presentation awards

2022
David Visscher and Erwin de Haan
A review of networking the cloud datacentre
2021
Niels Bügel and Albert Dijkstra
Mining User Reviews to Determine App Security
2020
Andris Jakubovskis and Hindrik Stegenga
Comparing Reference Architectures for IoT
and
Filipe R. Capela and Antil P. Mathew
An Analysis on Code Smell Detection Tools and Technical Debt
2019
Kareem Al-Saudi and Frank te Nijenhuis
Deep learning for fracture detection in the cervical spine
2018
Marc Babtist and Sebastian Wehkamp
Face Recognition from Low Resolution Images: A Comparative Study
2017
Stephanie Arevalo Arboleda and Ankita Dewan
Unveiling storytelling and visualization of data
2016
Michel Medema and Thomas Hoeksema
Implementing Human-Centered Design in Resource Management Systems
2015
Diederik Greveling and Michael LeKander
Comparing adaptive gradient descent learning rate methods
2014
Arjen Zijlstra and Marc Holterman
Tracking communities in dynamic social networks
2013
Robert Witte and Christiaan Arnoldus
Heterogeneous CPU-GPU task scheduling

Best presentation awards

2022
Luc Pol and Jeroen Lammers
A High-Level Overview of Minimum Graph-Triangulation Approaches
2021
Niels Bügel and Albert Dijkstra
Mining User Reviews to Determine App Security
2020
none, because of corona virus measures no presentations were given
2019
Sjors Mallon and Niels Meima
Dynamic Updates in Distributed Data Pipelines
2018
Tinco Boekestijn and Roel Visser
A comparison of vision-based biometric analysis methods
2017
Siebert Looije and Jos van de Wolfshaar
Stochastic Gradient Optimization: Adam and Eve
2016
Sebastiaan van Loon and Jelle van Wezel
A Comparison of Two Methods for Accumulating Distance Metrics Used in Distance Based Classifiers
and
Michel Medema and Thomas Hoeksema
Providing Guidelines for Human-Centred Design in Resource Management Systems
2015
Diederik Greveling and Michael LeKander
Comparing adaptive gradient descent learning rate methods
and
Johannes Kruiger and Maarten Terpstra
Hooking up forces to produce aesthetically pleasing graph layouts
2014
Diederik Lemkes and Laurence de Jong
Pyschopathology network analysis
2013
Jelle Nauta and Sander Feringa
Image inpainting

Best paper awards

2022
Erblin Ibrahimi and Sven Hofman
State of the Art: Performance Overview of Black-Box Web Application Scanners
and
Willard Verschoore and Gerrit Sijberen Luimstra
Is it Not Yet Time to Swish? Comparing the ReLU and Swish Activation Functions

2021
Ethan Waterink and Stefan Evanghelides
A Review of Image Vectorisation Techniques

2020
Anil P. Mathew and Filipe A.R. Capela
An Analysis on Code Smell Detection Tools
and
Thijs Havinga and Rishabh Sawhney
An Analysis of Neural Network Pruning in Relation to the Lottery Ticket Hypothesis

2019
Wesley Seubring and Derrick Timmerman
A different approach to the selection of an optimal hyperparameter optimisation method

2018
Erik Bijl and Emilio Oldenziel
A comparison of ensemble methods: AdaBoost and random forests

2017
Michiel Straat and Jorrit Oosterhof
Segmentation of blood vessels in retinal fundus images

2016
Ynte Tijsma and Jeroen Brandsma
A Comparison of Context-Aware Power Management Systems

2015
Jasper de Boer and Mathieu Kalksma
Choosing between optical flow algorithms for UAV position change measurement

2014
Lukas de Boer and Jan Veldthuis
A review of seamless image cloning techniques

2013
Harm de Vries and Herbert Kruitbosch
Verification of SAX assumption: time series values are distributed normally

Contents

State of the Art: Performance Overview of Black-Box Web Application Scanners

Sven J. Hofman, Erblin Ibrahimi

Abstract— Finding vulnerabilities in web applications has always been a challenging task. Humans have evidently not been able to manually detect them all before deploying their code, and this has led to an increased interest in automated tools to aid in finding these vulnerabilities. Black-box scanners are one of the more popular types of automation tools that do this. They crawl through a web application and probe around in an attempt to find potential vulnerabilities. They do this without having any knowledge about the architecture of the web applications and therefore treat them as black-boxes.

Throughout the years, we have seen several papers that compared the performance of these black-box scanners. Our paper provides an overview of the results of some of these papers, which looked at the performance of several scanners in detecting the most occurring vulnerabilities. We have considered 11 papers that have been published since 2010, which, in total, analysed over 20 black-box scanners. Since the technology behind most of the tested scanners is proprietary, it is difficult to provide recommendations to improve their performances. We can, therefore, only look at their reported behaviours and summarise their findings.

Moreover, we take a more detailed look at the performance in detecting SQL injection vulnerabilities, since these kinds of vulnerabilities have historically been the most popular. We conclude that for first-order SQL injection vulnerabilities, the overall performance of the considered scanners has improved slightly. On the other hand, the detection rate of second-order SQL injection vulnerabilities has not improved at all and remains zero.

Index Terms—Black-box scanners; web application security; vulnerability detection; SQL injection.

1 Introduction

IBM estimated that the global average total cost of a data breach in 2020 was \$3.86m, with individual (mega) breaches costing up to \$392m [8]. In 2021, this increased to \$4.24m, with individual (mega) breaches costing up to \$401m [9]. Many consider the Equifax 2017 data breach to be the most expensive data breach ever. In this attack, records with private and sensitive information (such as social security numbers, addresses, and sometimes even driver's licenses) of 146.6 million Americans were stolen [30]. After it became public that these records had been stolen, Equifax lost \$4 billion dollars in stock market value according to Wall Street [14]. Even though the data breach happened in 2017, Equifax still feels the effects to this day; in 2022, the FTC reached a settlement regarding the data breach, which concluded that Equifax has to pay \$425 million to help people affected by the breach [7].

The most common causes behind these breaches in 2021 were compromised credentials (20%, \$4.37m) and phishing (17%, \$4.65m). The third and fourth most common causes, however, were cloud misconfiguration (15%, \$3.86m) and vulnerability in third-party software (14%, \$4.33m), respectively. Both of these are caused by vulnerabilities and exposures in code, and combined they are the cause for 29% of the data breaches [9]. In the case of the Equifax 2017 data breach, there was a vulnerability in a version of the third-party software Apache Struts, which allowed remote code injection attacks through crafted HTTP headers [16].

Detecting these vulnerabilities manually can be quite challenging, as many new vulnerabilities are discovered on a daily basis. In 2021 alone, 18,439 new common vulnerabilities and exposures (CVEs) were reported, which amounts to over 50 new CVEs logged every day [27]. OWASP (Open Web Application Security Project) categorises these CVEs and publishes the OWASP Top Ten every three to four years, which is a report that ranks the most occurring CVE categories [21]. Table 1 shows the top five occurring categories since its first rankings, published in 2004.

As we can see in this table, the same set of categories of CVEs has persisted throughout the years: XSS, (SQL) Injection, Broken Access Control, Broken Authentication & Session Management, and CSRF. Even though OWASP defines 60 categories [23], this paper will mostly focus on the aforementioned set of categories, as these seem to consistently be at the top. Later in this paper, we will focus more on injection (specifically first-order and second-order SQL injections), since this category regularly occupies a spot in the top three, and is often at number one.

Vulnerabilities are evidently becoming increasingly costly, and humans cannot detect them all by manually inspecting the system, which is evident from the fact that so many CVEs are reported on a daily basis. Researchers have, therefore, taken an interest in finding an automated approach to detecting vulnerabilities in systems. There are several kinds of approaches to automating the process of finding vulnerabilities, of which white-box and black-box testing are common ones [15].

In white-box testing, the tester has full access to the network and the architecture. This allows the tester to more accurately scan the web application for vulnerabilities, as they have more information at their disposal.

Black-box testers, on the other hand, have limited structural information about the web application. They treat the web applications as black-boxes, and scan them for security vulnerabilities, with limited knowledge about the internal workings of the application. This closely resembles the point of view of attackers, who (presumably) do not have access to structural information of the web applications either. Black-box scanners are then defined as tools that perform black-box testing for users.

Our contribution to the research on black-box scanners considers of providing this paper, which will serve as an overview of the findings of several papers that inspected and compared the performance of various black-box scanners. Moreover, we aim to answer the following research question: *Has the performance of black-box scanners improved for detecting first- and second-order SQL injection vulnerabilities throughout the years?*

- *Sven J. Hofman is with the Faculty of Engineering, University of Groningen, Email: s.j.hofman.1@student.rug.nl*
- *Erblin Ibrahimi is with the Faculty of Engineering, University of Groningen, Email: e.ibrahimi@student.rug.nl*

	1	2	3	4	5
2004	Unvalidated Input	Broken Access Control	Broken Authentication & Session Management	XSS	Buffer Overflow
2007	XSS	Injection Flaws	Malicious File Execution	Insecure Direct Object References	CSRF
2010	Injection	XSS	Broken Authentication & Session Management	Insecure Direct Object References	CSRF
2013	Injection	Broken Authentication & Session Management	XSS	Insecure Direct Object References	Security Misconfiguration
2017	Injection	Broken Authentication	Sensitive Data Exposure	XML External Entities	Broken Access Control
2021	Broken Access Control	Cryptographic Failures	Injection	Insecure Design	Security Misconfiguration

Table 1. Overview of the top five most occurring web application vulnerabilities from several OWASP Top Ten reports [20].

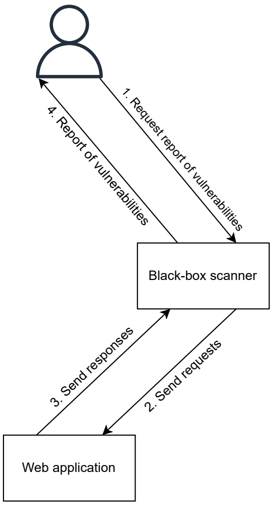


Fig. 1. Abstraction of the workflow of black-box scanners. The scanner sees the Web Application as a black-box and simply interacts with it by sending requests and investigating the responses.

2 Background

As we have previously seen in Table 1, injections are consistently at the top of the most occurring vulnerabilities. OWASP calls them "Injections", which is a collective name and not only includes SQL injections, but also NoSQL, ORM, and LDAP injections, among others. Note that the process of performing injections on the aforementioned languages and protocols is similar to performing SQL injection [18]. We will, therefore, only look at the details of SQL injections, since they are the focus of our paper.

2.1 Types of SQL Injections

SQL injections can generally be classified into three categories; first-order injection, second-order injection and lateral injection [17]. The papers that we consider in this overview only look at the first two categories, so these will be of relevance to us.

2.1.1 First-Order SQL Injection

First-order SQL injection is an attack that inserts a malicious query into places where this is not expected, which allows you to craft custom queries and run them on the database. This commonly occurs when an entry of a user-filled form contains such a malicious query. As an example, let us consider the following PHP query string, which uses PHP variables `$username` and `$password` that were retrieved from a filled-in form to check whether these credentials exist in the system:

```sql
SELECT *
FROM   users
WHERE  username = '$username' AND password = '$password';
```

In the case of a non-malicious form, where both `$username` and `$password` are `admin`, the query would look as follows (and work as expected):

```sql
SELECT *
FROM   users
WHERE  username = 'admin' AND password = 'admin';
```

Let us now consider the case where `$username` is `' OR 1=1; --`, and `$password` can be anything (we will continue with `admin` in this case). The query becomes

```sql
SELECT *
FROM   users
WHERE  username = '' OR 1=1; --' AND password = 'admin';
```

Running this query will give us all the entries in the `users` table, since `username = '' OR 1=1` always evaluates to `true`, independent of the table entries. Note that because of the `--` that was appended to the input, everything that comes after this will be commented out, and therefore disregarded. This means that `AND password = 'admin';` is, therefore, also ignored.

2.1.2 Second-Order SQL Injection

Second-order SQL injections, on the other hand, do not directly make use of the malicious queries, but rather store them in the database first. In contrast to first-order SQL injection attacks, the attacker simply inserts the malicious query as a database entry and hopes that an application later uses the database entries without filtering them, which results in inadvertently running the malicious query. In these types of attacks, the attacker usually does not know whether there are any applications that use the data in an unsafe manner, so second-order SQL injections attacks are run blindly, in the hope that this is the case.

2.1.3 Lateral Injection

Lateral injections work specifically on Oracle's PL/SQL. Simply put, it is performed when a data type is transformed into another data type. A common example is when a variable of type `NLS_Date_Format` or `NLS_Numeric_Characters` are transformed into a string, using `To_Char()`.

2.2 Black-Box Scanners

In Figure 1, a general schema is shown of how a black-box scanner can be used to detect vulnerabilities in a web application. However, it does not display the internal workings of a black-box scanner. Each black-box scanner works differently, but they all share a similar approach. This procedure can be described by the following stages:

0. **Set-up:** as an initialisation step, the user sets the configuration of the particular scanner. In this step, the user can, for example, select which particular vulnerabilities to scan for. This can be useful, as it can save a significant amount of time since black-box scanners take quite a lot of time to run. The exact settings depend on the scanner.

1. **Crawling:** once the scanner has been set up properly, the user starts the scanner. Black-box scanners can either be run locally or through a server (or both). The scanner starts crawling through the website and obtains the HTML source code of each accessible link. Note that this is not always a trivial task. As an example, consider a page that can only be accessed after having filled in a login form or captcha. This requires the scanner to restart the crawling phase after having filled in this form.

2. **Identifying forms and entry points:** after the HTML code has been parsed, the scanner tries to identify entry points into the web application. There are several kinds of entry points; forms are the most common ones. For example, if the scanner finds a form, it could try to detect whether it is a login form. A typical login form has two HTML elements of type `text`; one with a description of `username` or `login`, and one with a description of `password`. Lastly, there should be a button of some sort that sends a `POST` request. These kinds of vulnerable entry points that are used to exploit a web application are called attack vectors.

3. **Constructing and submitting attack code:** when an entry point has been identified (such as the aforementioned login forms), the scanner will produce data that will be used to fill in the form. There are several ways of getting this data:

   - Getting data that has been pre-defined in a database or dictionary. A common example is trying to use SQL injection queries. Note that some scanners can also be set up to always use a user-defined username whenever the scanner detects an HTML element that requires a username. A common test of scanners is to try the login credentials `admin` / `admin`, but in those scanners, these credentials can be customised.
   - Generating data randomly from scratch. Fuzzers can be very useful tools for this.

   After the form has been filled, the scanner sends the request to the server and awaits a response.

4. **Analysing replies:** once the scanner receives a response from the server, it starts parsing it and determines whether it is a 'valid' response. This is also non-trivial. First of all, responses to entry point requests are commonly given in a human-readable format. This means that the scanners should be able to intelligently extract the relevant information from such a response. Next, if an error is returned, the scanner should be able to determine whether this is expected, which also depends on the type of error. Continuing with the aforementioned example of the user creation form, one should expect to get an error when providing a username or password that is too short. If the server returns an SQL error, on the other hand, this should be reported to the user as this implies that an SQL injection vulnerability exists.

Paper	Date	Web Application Vulnerability Scanners
[5]	2010	Acunetix WVS 6.1, AppScan 7.8.0.0, Burp 1.2, **Grendel-Scan** 1.0, Hailstorm 5.7, MileScan 1.4, N-Stalker 2009, NTOSpider 3.2.067, Paros 3.2.13, **w3af** 1.0-rc2, Webinspect 7.7.769.0
[1]	2010	Acunetix WVS 6.5, HailStorm Pro 6.0, WebInspect 8.0, Rational AppScan 7.9, McAfee SECURE Web, N-Stalker 7.0.0, QualysGuard PCI Web, NeXpose 4.8.0
[11]	2011	Acunetics WVS 7.0, Rational AppScan 8.0, QualysGuard Express Suite 6.16.60-1
[2]	2012	Acunetics WVS 6.5, AppScan 7.9, WebInspect 8.0, QualysGuard PCI 2009
[29]	2013	**IronWASP** 0.9.5.0, NetSparker 2.5, N-Stalker 7.1.1.126, **OWASP ZAP** 2.0.0, **w3af** 1.2-r6654, **Vega** 1.0 (beta)
[12]	2015	**OWASP ZAP** 2.3.1, **skipfish** 2.10b
[25]	2015	Acunetix WVS 8.0, Rational AppScan 9.0, **OWASP ZAP** 2.3.1
[10]	2017	Acunetix WVS 10, AppSpider 6.0, Arachni 2.2.1, BurpSuite 1.6.12, **IronWASP** 0.9.7.1, NetSparker 2.3, **skipfish** 2.10, **Vega** 1.0 (beta), **Wapiti** 2.3.0, **w3af** 1.2, **OWASP ZAP** 2.3.1
[26]	2018	Acunetix WVS 11.0, BurpSuite 1.7.30, NetSparker 4.7.1, Nessus (cloud based), **OWASP ZAP** 2.7.0
[13]	2018	Arachni 1.5.1, **OWASP ZAP** 2.7.0
[28]	2021	Arachni 1.5.1-0.5.12, **IronWASP** 0.9.8.6, **skipfish** 2.10 (beta), **Vega** 1.0, **OWASP ZAP** 2.7.0

Table 2. Web application vulnerability scanners and their versions used in different papers. Open-source scanners are displayed in bold.

3 Overview

This section will provide an in-depth overview of the research conducted on the evaluation of black-box scanners for web applications. These black-box scanners are called web application vulnerability scanners (WAVSs). We will consider most of the papers published on this topic in the years 2010 up until now. There are three components in the process of evaluating WAVSs that are most important: (i) the WAVSs on which tests are performed, (ii) the vulnerable web application(s) to run the WAVSs on, (iii) the measures used to quantify the performance of the WAVSs. In total, 11 papers will be included in this study. The three components mentioned above will be discussed in the coming sections.

3.1 Web Application Vulnerability Scanners

The general concept behind black-box scanners has been explained in Section 2.2. For both their mandatory (e.g. identify SQL injection) and optional (e.g. typecasting) features, web application scanners have to meet certain requirements specified in [3]. Even though the minority of the WAVSs is open source [24], a significant amount of the research papers on testing WAVSs consider open-source tools, as can be seen in Table 2. This table gives an overview of the tables used by the research papers. Open-source tools are considered due to the fact that they are easily accessible for the public to be used and for the researchers to be examined. The platforms on which the tested scanners can be used differ and some of the scanners are only provided as a cloud-based service.

3.2 Vulnerable Web Applications

In addition to selecting the black-box scanners, users have to pick one or more vulnerable web applications to evaluate the performance of the scanners. The metrics used to classify the scanners' qualities are discussed in Section 3.3. In this section, we will provide the vulnerable web applications used by the selected research papers. These web applications are also known as test-beds.

One of the most widely used test-beds is called WackoPicko and was designed by Doupé et *al.* [5]. It contains several of the vulnerabilities mentioned in Table 1, such as stored cross-site scripting and second-order SQL injection. Exploiting the latter requires the scanner to first authenticate itself by providing login credentials, whereas this is not required for the former. We will see later in Section 4.1 that this affects the performance of the WAVS.

Paper	Test-beds	Language
[5]	WackoPicko (16/2)	PHP
[1]	Drupal 4.7.0 (23/2), phpBB2 2.0.19 (13/1), Wordpress 1.5strayhorn (42/8), PCI (·/3)	PHP
[11]	PCI (·/3), WackoPicko (16/2), MatchIt (1/1)	PHP, MySQL
[2]	PCI (·/3), 27 web applications (·/·)	PHP, ASP, Java, Python, Ruby
[29]	WackoPicko (16/2)	PHP
[12]	DVWA (8/2), WAVSEP (611/105)	PHP, JS, Java
[25]	WackoPicko (16/2), Scan-bed (1/1)	PHP
[10]	WAVSEP (611/105)	Java
[26]	7 web applications (·/·)	C#, PHP, ASP, HTML5
[13]	OWASP Benchmark (2740/504)	Java, HTML
[28]	OWASP WebGoat (·/·), DVWA (8/2)	PHP, JS, Java, HTML

Table 3. Test-beds used by the different papers. The numbers within the parenthesis are the total number of vulnerabilities and the number of SQL injection vulnerabilities, respectively. These are obtained from the papers in which they are used, except for OWASP Benchmark, which is obtained from [19]. A dot indicates that the number is not known.

A paper from Bau et *al.* [1] from the same year used multiple web applications to test the scanners. This included applications with Drupa, phpBB and WordPress that were all known to contain vulnerabilities. In addition to this, they too constructed their own custom test-bed referred to as PCI. In the paper, it is mentioned that the number of vulnerabilities in their test-bed is "fairly proportional with the vulnerability population in the wild". The web application contains the top vulnerabilities occurring in 2010, stated in Table 1. Later, in 2012, Bau et *al.* [2] published another paper where several scanners were tested against 27 early-stage web applications built by Silicon Valley startups and freelancers, using five different programming languages.

Similarly, Khoury et *al.* [11] created their own custom test-bed called "MatchIt". It was designed to test a scanner's ability to detect a single second-order SQL injection vulnerability by eliminating factors such as user registration or complicated crawling that could impede the scanner's exploitation of this vulnerability. This paper also used PCI and WackoPicko in their evaluation of black-box scanners.

In addition to WackoPicko, Parvez et *al.* [25] used their own custom test-bed called "Scan-bed". This application was constructed for the same reason as MatchIt.

Unlike the other papers, Qasaimeh et *al.* [26] used seven web applications created by several commercial companies such as IBM and Hewlett-Packard. These web applications were written in different languages like PHP and HTML5.

Paper	Metrics
[5]	TP, FP, FN, Running time, Reachability
[1]	TP, FP, Scanner footprint, Running time
[11]	Network traffic, Attack vectors
[2]	TP, FP, FN
[29]	TP, FN, Running time
[12]	TP, FP, Precision
[25]	Network traffic, Attack vectors
[10]	TP, FP, FN, Precision, Recall, F-score
[26]	TP, FP, Accuracy
[13]	TP, FP, TN, FN
[28]	TP, FP, TN, FN, Precision, Recall, Youden Index

Table 4. Evaluation metrics used by the different papers.

Lastly, The Web Application Vulnerability Scanner Evaluation Project (WAVSEP) [4], Damn Vulnerable Web Application (DVWA) [6], OWASP Benchmark [19] and OWASP WebGoat [22] are all open-source applications that have also been used to test the black-box scanners. An overview of the test-beds and their properties can be found in Table 3.

3.3 Evaluation Metrics

The objective of evaluating WAVSs is to measure to what extent these tools can discover and accurately diagnose web application problems. Several measures have been used to accomplish this and test the performance of WAVSs. These include well-defined indicators such as the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). In the context of WAVSs, these metrics respectively mean that the scanner identifies a real vulnerability, fails to identify a real vulnerability, correctly disregards a 'fake' vulnerability, and fails to disregard a 'fake' vulnerability. We will refer to the detection rate as the true positive rate, that is

$$\text{detection rate} = \frac{TP}{TP+FN}.$$

Using these measures, additional commonly used metrics can be defined like accuracy, precision, recall, F-score and Youden index. Other performance measures that are considered are the running time and scanner footprint (the number of bytes sent and received). Doupé et *al.* [5] designed a "Reachability" score; a subjective measure of how difficult it is for the WAVS to crawl to the page containing the vulnerability.

Research by Khoury et *al.* [11] and Parvez et *al.* [25] did not make use of any of the metrics above. Instead, they monitored the network traffic to store and execute attacks that exploit the vulnerabilities. Furthermore, they counted the number of distinct and relevant attack vectors implemented using the fields susceptible to second-order SQL injection. An overview of the evaluation metrics used by different papers is given in Table 4.

4 SQL Injection

As one can see from Table 1, (SQL) injection has been scoring very high in the past decade as being one of the most common vulnerabilities among web applications. It is therefore no surprise that the majority of the papers published in the field of evaluating black-box web application security scanners are primarily focused on this type of attack. In this section, we will provide an analysis of the results discovered in the research papers regarding both first- and second-order SQL injection, the theory of which has been explained in Section 2.1. Moreover, we will answer our research question regarding the improvements in performance of WAVSs throughout the years. We should note that in most of the papers of this research field, the results of different scanners have been anonymised. Due to this, it is difficult to provide a direct comparison between the performance of specific scanners in different papers. However, a general conclusion can still be derived from the findings of the papers that are considered. We will provide the results below in chronological order.[1]

4.1 Results

In 2010, Doupé et *al.* [5] showed, using their WackoPicko test-bed, that all the tested scanners were able to detect the first-order SQL injection vulnerability. However, none of the scanners were able to detect the second-order SQL injection vulnerability. Both they and Khoury et *al.* [11] in 2011 mentioned that this is due to the valid registration that is required in WackoPicko to exploit the vulnerability; scanners were not able to detect the fact that the fields “Password” and “Password Again” required the same values.

Bau et *al.* [1] found in 2010 that considering the average of all scanners, 21.4% of the first-order SQL injection vulnerabilities had been detected, using the scanners provided in Table 2. Note that the results mentioned in this paper were anonymous. However, using the more recent paper of Bau et *al.* [2] from 2012, we can deduce that both WebInspect and AppScan had the lowest detection rate, namely 14.2%. QualysGuard attained a detection rate of 29%, whereas the highest rate was achieved by Acunetix WVS: 43%. Moreover, Bau et *al.* [2] did not observe any changes in 2012 for the above four scanners for both first- and second-order SQL injection vulnerabilities. They speculated that this was due to the fact that this type of detection was outside the scope for the scanners at that time. On the other hand, very few false positives were found for SQL injections. This would make sense if the scanners did not consider this type of vulnerability; one does not make false detections if one does not make any detections at all.

Of the scanners tested in Suteva et *al.* [29] in 2013, none were able to detect second-order SQL injection vulnerabilities. Four out of six scanners were able to exploit first-order SQL injection vulnerabilities, but none verified to the user that the vulnerabilities were found. N-Stalker and w3af were not able to detect the possibility for first-order SQL injection in any way.

In 2015, Makino and Klyuev [12] evaluated two scanners: OWASP ZAP and skipfish. They used WAVSEP and DVWA as test-beds and reported that these contained 105 and 2 SQL injection vulnerabilities, respectively. According to their study, the precision of OWASP ZAP on WAVSEP was 100%, meaning that OWASP ZAP did not register any false positives. On the other hand, skipfish had a precision rate of 9.5%. For all the false positives test cases that WAVSEP contains, skipfish recorded all of them as real vulnerabilities, which therefore resulted in a false positive rate of 100%. For the DVWA benchmark, the paper provided the number of true positive vulnerabilities. Here, OWASP ZAP and skipfish recorded 2 and 1 vulnerabilities, resulting in a 100% and 50% detection rate for OWASP ZAP and skipfish, respectively. So overall, OWASP ZAP performed better than skipfish. Unfortunately, the paper did not provide separate results for first- and second-order SQL injection vulnerabilities. Consequently, we do not know if these scanners have made improvements with respect to previous scanners in terms of detecting second-order SQL injection vulnerabilities.

We mentioned in Section 3.3 that Khoury et *al.* [11] and Parvez et *al.* [25] made use of network traffic to test the capabilities of black-box scanners. As such, it is hard to compare these results to the findings of the other papers. Not only this, both papers mostly used different scanners and test-beds, and anonymised their findings. For example, WackoPicko is used in both papers, but the network traffic generated by these scanners is not given by Parvez et *al.* [25]. Similarly, Acunetix WVS is tested in both papers, but we cannot juxtapose the results of this scanner due to the omission of scanner names with the results. The only thing that can be concluded after collating the findings of these two papers is that none of the scanners were able to implement attack codes for the second-order SQL injection vulnerabilities. Hence, in the context of second-order SQL injection, the detection rate has not improved from 2011 to 2015 based on these papers.

Idrissi et *al.* [10] performed an extensive evaluation of several black-box scanners in 2017. As can be seen from Table 2, they used many scanners (both commercial and open source) and tested these on the WAVSEP web application. At this time, WAVSEP contained 136 real SQL injection vulnerabilities and 10 test-cases representing ‘fake’ vulnerabilities to measure how lenient the scanners were in recording vulnerabilities. One thing that stands out is that skipfish obtained a false positive rate of 0%, instead of 50% in 2015 [12]. Acunetix WVS achieved a detection rate of 100%, although it should be noted that by looking at the source code of WAVSEP [4], it appears that it does not contain any second-order SQL injection vulnerabilities, whereas the test-bed used by Bau et *al.* [2] (which also tested Acunetix WVS) does. The average detection rate is 93.8%, with the lowest being 59.6% and the highest being 100%. This is much better than the result published by Bau et *al.* [2] in 2012, albeit on a different test-bed.

The most recent paper that specifically mentions results SQL injection vulnerabilities is from 2018, by Mburano and Si [13]. OWASP ZAP and Arachni were compared on the OWASP Benchmark. A detection rate of 58.1% and 20% was reported for OWASP ZAP and Arachni, respectively.

4.2 Discussion

We have seen that since 2010, many research papers have made an attempt to evaluate black-box scanners. The most important observations that can be made from these results are the following: (i) an increase in performance of detecting first-order SQL injection vulnerabilities can be observed, (ii) the detection rates for first and second-order SQL injection vulnerabilities still differ significantly, (iii) black-box scanners remain unable to detect second-order SQL injection vulnerabilities. The first observation is discussed in Section 4.2.1 and the second and third observations are discussed in Section 4.2.2.

4.2.1 First-Order SQL Injection Vulnerabilities

Using a large number of scanners, Bau et *al.* [1] concluded in 2010 that the average detection rate of detecting first-order SQL injection vulnerabilities was 21.4%, with the range of detection rates being [14.2,43]. In 2012, they showed that this had not changed. The next extensive (i.e., including a large number of scanners) study on evaluating black-box scanners was done in 2018 by Idrissi et *al.* [10]. Here, the average detection rate equalled 93.8%, with a range of [59.6, 100]. Based on these results, the performance of black-box scanners has increased significantly in the context of first-order SQL injections. Ideally, one should have more sources when drawing such a conclusion. However, even if the results of Idrissi et *al.* [10] are exaggerated, it is still very likely that the results are better than those of black-box scanners in 2010.

4.2.2 Second-Order SQL Injection Vulnerabilities

In all papers where both first- and second-order SQL injection vulnerabilities were discussed, the latter had a worse detection rate than the former. This is not surprising; finding second-order SQL injection vulnerabilities using black-box scanners is significantly more difficult, since black-box scanners are based on the idea of knowing little about the internal workings of the application. For first-order injection, this is less relevant since the scanner can directly verify if the attack has worked. For second-order SQL injection, the scanner not only has to implement the attack, but it also has to find a way to force the application to trigger the attack without knowing the source code of the application.

All the papers that investigated second-order SQL injection vulnerabilities found that the detection rate of the black-box scanners for this vulnerability is zero. A possible explanation for this result was given by Bau et *al.* [1] in 2010. There, it was reported that “multiple vendors confirmed their difficulty in designing tests which detect

[1] Note that not all 11 papers are included in this section, since not all papers provided specific information about the performance of black-box scanners on SQL injection vulnerabilities.

second-order vulnerabilities". Moreover, in 2012, Bau et *al.* [2] suspected that the industry does not put much effort in the development of detecting second-order SQL injection vulnerabilities, due to the existence of a known solution, namely prepared statements. We believe that this is the most likely reason for the non-existing improvement of the performance on second-order SQL injection vulnerabilities. This also provides a justification as to why no papers have been published since 2015 that study the performance of black-box scanners on this specific vulnerability.

5 Conclusion

Black-box scanners can be a useful aid in the search for vulnerabilities in a web application. However, they are not able to detect all the existing vulnerabilities. Several papers have shown this to be the case when they ran popular black-box scanners on various open-source test-beds, testing for the most popular vulnerabilities at the time. This paper presented an overview of the findings of these papers. More specifically, 11 papers were considered that, in total, tested more than 20 black-box scanners on over 40 test-beds.

In the second part of our paper, we looked at the performance of black-box scanners in the context of SQL injection. This resulted in the following answer to our research question: the detection rate of first-order SQL injection vulnerabilities has improved, but the detection rate for second-order SQL injection vulnerabilities did not and remained zero. This is worrying, since injection remains at the top of the most occurring vulnerabilities, according to the OWASP Top Ten. It became clear that the detection rate for SQL injection has partially improved over time; first-order SQL injection vulnerabilities are detected more often, whereas the detection of second-order SQL injection vulnerabilities has not improved in any manner.

Further research can be done on how the detection rate of SQL injection vulnerabilities can be improved. Since the majority of the scanners are proprietary and we, therefore, do not have access to the source code or methodology, it is hard to make a general recommendation on how they can be improved. This is especially the case since there is no standardised set of evaluation metrics for the performance of black-box scanners, which also makes it difficult to properly compare the papers and their results. Finding a standardised set of evaluation metrics could be beneficial in future research since it would allow researchers to get a clearer overview of what aspects of the black-box scanners have advanced, and which ones still need improvements.

Acknowledgements

The authors wish to thank dr. Fadi Mohsen for his valuable feedback and willingness to help out whenever we had questions.

References

[1] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell. State of the art: Automated black-box web application vulnerability testing. In *2010 IEEE symposium on security and privacy*, pages 332–345. IEEE, 2010.

[2] J. Bau, F. Wang, E. Bursztein, P. Mutchler, and J. C. Mitchell. Vulnerability factors in new web applications: Audit tools, developer selection & languages. *Stanford, Tech. Rep*, 2012.

[3] P. E. Black, E. Fong, V. Okun, R. Gaucher, et al. Software assurance tools: Web application security scanner Functional Specification Version 1.0. *Special Publication*, 2008.

[4] S. Chen. The Web Application Vulnerability Scanner Evaluation Project. https://github.com/sectooladdict/wavsep. Accessed: 2022-02-26.

[5] A. Doupé, M. Cova, and G. Vigna. Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131. Springer, 2010.

[6] DVWA. Damn Vulnerable Web Application (DVWA). https://dvwa.co.uk/. Accessed: 2022-03-01.

[7] FTC. Equifax Data Breach Settlement. https://www.ftc.gov/enforcement/cases-proceedings/refunds/equifax-data-breach-settlement. Accessed: 2022-02-24.

[8] IBM. Cost of a Data Breach Report 2020. https://www.ibm.com/downloads/cas/QMXVZX6R. Accessed: 2022-02-17.

[9] IBM. Cost of a Data Breach Report 2021. https://www.ibm.com/downloads/cas/OJDVQGRY. Accessed: 2022-02-17.

[10] S. Idrissi, N. Berbiche, F. Guerouate, and M. Shibi. Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076, 2017.

[11] N. Khoury, P. Zavarsky, D. Lindskog, and R. Ruhl. An analysis of black-box web application security scanners against stored sql injection. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 1095–1101. IEEE, 2011.

[12] Y. Makino and V. Klyuev. Evaluation of web vulnerability scanners. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 399–402. IEEE, 2015.

[13] B. Mburano and W. Si. Evaluation of web vulnerability scanners based on owasp benchmark. In *2018 26th International Conference on Systems Engineering (ICSEng)*, pages 1–6. IEEE, 2018.

[14] Money. Equifax Data Breach Settlement. https://money.com/equifaxs-massive-data-breach-has-cost-the-company-4-billion-so-far/. Accessed: 2022-02-24.

[15] S. Nidhra and J. Dondeti. Black box and white box testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2):29–50, 2012.

[16] NVD. CVE-2017-5638 Detail. https://nvd.nist.gov/vuln/detail/cve-2017-5638. Accessed: 2022-02-26.

[17] Oracle. Types of SQL Injection Attacks. https://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_attacks.htm. Accessed: 2022-03-02.

[18] OWASP. A03 Injection. https://owasp.org/Top10/A03_2021-Injection/Y. Accessed: 2022-02-24.

[19] OWASP. Owasp Benchmark. https://owasp.org/www-project-benchmark/. Accessed: 2022-02-19.

[20] OWASP. Owasp Top 10 Document Repository. https://github.com/OWASP/Top10. Accessed: 2022-02-24.

[21] OWASP. Owasp Top Ten. https://owasp.org/www-project-top-ten/. Accessed: 2022-02-24.

[22] OWASP. Owasp WebGoat. https://owasp.org/www-project-webgoat/. Accessed: 2022-03-01.

[23] OWASP. Vulnerabilities. https://owasp.org/www-community/vulnerabilities/. Accessed: 2022-02-17.

[24] OWASP. Vulnerability Scanning Tools. https://owasp.org/www-community/Vulnerability_Scanning_Tools. Accessed: 2022-02-19.

[25] M. Parvez, P. Zavarsky, and N. Khoury. Analysis of effectiveness of black-box web application scanners in detection of stored sql injection and stored xss vulnerabilities. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 186–191. IEEE, 2015.

[26] M. Qasaimeh, A. Shamlawi, and T. Khairallah. Black box evaluation of web application scanners: Standards mapping approach. *Journal of Theoretical and Applied Information Technology*, 96(14):4584–4596, 2018.

[27] Redscan. Redscan analysis of NIST NVD reveals record number of vulnerabilities in 2021. https://www.redscan.com/news/nist-nvd-analysis-2021-record-vulnerabilities/. Accessed: 2022-02-17.

[28] P. Sarpong, L. Larbi, D. Paa, I. Abdulai, R. Amankwah, and A. Amponsah. Performance evaluation of open source web application vulnerability scanners based on OWASP benchmark. *International Journal of Computer Applications*, 174:15–22, 02 2021.

[29] N. Suteva, D. Zlatkovski, and A. Mileva. Evaluation and testing of several free/open source web vulnerability scanners. In *The 10th Conference for Informatics and Information Technology (CIIT 2013)*, pages 221–224. FCSE, 2013.

[30] U.S. Securities and Exchange Commission. Equifax's statement for the record regarding the extent of the cybersecurity incident announced on september 7, 2017. https://www.sec.gov/Archives/edgar/data/33185/000119312518154706/d583804dex991.htm. Accessed: 2022-02-26.

A Review of Feature Selection and Ranking Methods

Tom Maguire and Lennard Manuel

Abstract—Datasets are constantly increasing in size due to the ever-growing amount of data that is collected. However, the quality of this data is not always of the highest standard, with datasets containing many irrelevant features. Feature selection and ranking help to address this problem.

Feature selection is the process of identifying the most important or relevant features in a dataset. This process is essential because by only looking at the relevant features, the speed of computation and the accuracy of classification can be improved. Feature selection allows for irrelevant features to be discarded and ignored.

In this paper, we take a look at four different methods that are used for feature selection, namely XGBoost, FeatBoost, ReliefF and Boruta. The goal of this paper is to provide an overview of what types of feature selection algorithms exist by conducting a literature review, assessing what the algorithms' strengths and weaknesses are, and how they compare to each other. This comparison is done on their computation times and redundancy rates. The results showed that XGBoost had the fastest average computation time and that FeatBoost was the best in reducing redundancy among the selected features.

Index Terms—Feature selection, feature ranking, boosting, XGBoost, Boruta, FeatBoost, ReliefF.

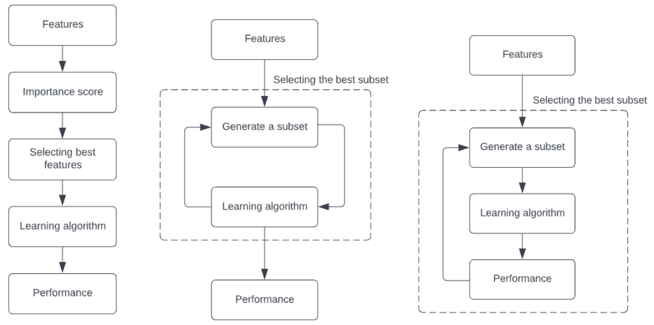
1 Introduction

The amount of data created worldwide has been increasing exponentially over the last decade. In 2010, two zettabytes of data were created and only ten years later this number has grown to 64.2 zettabytes [11]. This explosion of data has led to very large datasets, containing many irrelevant features, which increases the challenges of creating useful models to make accurate predictions on these databases, or of deriving useful information from them.

The curse of dimensionality [2] explains the idea that as the number of features in a dataset increases, the number of samples required to create a predictive model with an arbitrary level of accuracy also grows exponentially.

Feature selection is the process of identifying the most important or relevant features in a dataset. As a result of the ever-increasing amount of data, feature selection is now more important than ever before [10]. Feature selection is important for a few reasons. Firstly, it allows for the accuracy of models to be improved as removing irrelevant features helps to reduce the amount of overfitting[8]. Secondly, using feature selection to identify and remove irrelevant features from the dataset reduces the effect of the curse of dimensionality. Thirdly, even if feature selection is not directly helpful in increasing a model's ability to make predictions on a dataset, it is still beneficial as it reduces storage cost for that dataset as well as reducing the cost of performing computations. Lastly, there are disciplines where the primary goal of analysing a dataset is to understand the relationship between features and a certain outcome. This is common in biomedical applications where the goal is to identify key indicators of specific diseases [1]. Feature selection is key here, as it shows the hidden relationships between combinations of features and the presence of diseases.

In this paper, the aim is to analyse and compare recent research in the field of feature selection. The objective is to explore the potential benefits and drawbacks of the different proposed ideas and to identify areas of agreement or conflict in the recent research. First, a background of feature selection will be given in Section 2. Then, the three main papers that make up the bulk of the discussion will be introduced in Section 3. In Section 4, the feature selection algorithms that are used for the experiments will be discussed, after which the experimental settings will be described in Section 5. After that, we will dive into the results and discuss them in detail in Section 6 before finally concluding the research in Section 7 and offering some potential future work in Section 8.



(a) Filter method. (b) Wrapper method. (c) Embedded method.

Fig. 1: Three types of feature selection.

2 Background

Feature selection is a large field of research due to its importance. Previous literature has proposed many different methods for feature selection. These methods normally fall into one of three distinct categories: filter, wrapper and embedded [4].

The first category of methods, filter methods, rank features independently of the modelling algorithm used. The features are ranked using a performance measure, such as information loss, similarity and other statistical measures. Examples of filter methods include Chi-square and gain ratio.

Wrapper methods differ from filter methods as they use the performance of the modelling algorithm to evaluate the quality of a feature or subset of features. Wrapper methods make use of various search strategies to perform an exhaustive search of the space of possible feature sets. The aim of these search strategies is to identify candidate subsets which are then evaluated using the wrapped modelling algorithm.

Embedded methods are methods in which feature selection is performed as an integral part of the modelling algorithm's execution, hence the name embedded. As the modelling algorithm is run, irrelevant features are identified and discarded. Examples of embedded methods include random forests and linear classifiers, such as support vector machines (SVM), used in combination with regularised cost functions such as Lasso. These different categories of feature selection methods are shown in Figure 1.

Lasso regularization works by adding a penalty term which is the

- *Thomas Maguire of the Faculty of Science and Engineering, University of Groningen, E-mail: t.d.maguire@student.rug.nl*
- *Lennard Manuel of the Faculty of Science and Engineering, University of Groningen, E-mail: l.r.manuel@student.rug.nl*

L1 regularization of the weight vector to the cost function [14]:

$$Cost = \sum_{i=0}^{N}(y_i - \hat{y}_i) + \sum_{j=0}^{M}|W_j|_1 \tag{1}$$

For this paper, the notation $\hat{y}$ is used to indicate the predicted value of feature y by an arbitrary modelling algorithm. L1 regularization penalizes non-zero weight values and this leads to irrelevant features being assigned a weight value of 0. Thus by using L1 regularization as a penalty term in a cost function and using a classifier such as SVM, feature selection is performed as only relevant features have non-zero weight values. This also ranks the features as the remaining features' importance is ranked by the magnitude of the associated weight value for that feature.

The choice of which class of methods to use is dependent on the objectives of feature selection. Filter methods are quick and can be run independently of any modelling algorithm, meaning they can be used as a pre-processing step without having to have identified a modelling algorithm. However, empirical evidence has shown that wrapper methods perform better on average than filter methods across an array of datasets [12]. A common rebuff of wrapper methods is that they are less general than filter algorithms, meaning that they only have good performance when used with the modelling algorithm used to evaluate features. However, there is evidence that a wrapper's generalisation to other modelling algorithms is dependent on the relative complexity of the modelling algorithm used to evaluate features. Simple classification algorithms such as KNN used as the evaluation function for wrapper methods lead to good generalisation to other classification algorithms [12].

3 Literature Review

In this section, the three main papers looked at in this paper are introduced.

3.1 A Practical Approach to Feature Selection

This paper is much older than the other two papers looked at in this section, having been published in 1992. It introduces the nowadays well-known relief algorithm as an approach to feature selection [6]. What set the relief algorithm apart from previous filter methods was the changing of the key objective. Instead of the goal being to find the smallest sufficient subset of features, which was the goal of the majority of algorithms that preceded relief, the algorithm took a different approach by using statistical methods to identify relevant features to find a subset of relevant features. The approach used by the relief algorithm meant that it was not guaranteed to find the optimal subset of features but it would normally find a good subset of features in linear time $O(pn)$, where p is the number of features and n is the number of training instances.

The relief algorithm is a filter method, and at the time it was first introduced it was impactful due to its ability to detect and understand feature interaction. The performance of the relief algorithm was compared to an exhaustive search and a heuristic search and the results showed that the relief algorithm was a big improvement in terms of its tolerance to noise and its ability to understand feature interaction. However, relief does not help with redundant features. If most features are relevant to the concept, it selects all the relevant features even if a smaller subset of these features is sufficient for describing training instances. The paper only introduced the relief algorithm for two-class classification problems but it does state in its future work section that the algorithm can easily be extended to solve multi-class classification problems. To do this the multi-class classification problem has to be reframed as a set of two-class classification problems.

3.2 Feature Selection with the Boruta Package

The paper begins by introducing the minimal-optimal problem and the all-relevant problem [7]. The minimal-optimal problem is the problem of finding the smallest possible subset of relevant features. The relief algorithm from the previous section gives an approximate solution to the minimal-optimal problem. The all-relevant problem is the problem of identifying all features which are relevant to classification, no matter how little the importance of the feature. To solve this, all attributes with a correlation with the class label higher than that of a randomly generated feature need to be identified.

The choice of solving the minimal-optimal problem or all-relevant problem depends on the overall goal. A minimal-optimal set of features is sufficient to make black box predictions of a class label. To understand the underlying mechanics the set of all relevant features must be known.

Solving the all-relevant problem is harder than solving the minimal-optimal problem because filter methods cannot be used as a feature may be unimportant on their own, but important in combination with other features.

The paper introduces a novel feature selection algorithm called Boruta to solve the all-relevant problem. The algorithm is a wrapper method that uses a random forest classification algorithm to evaluate features. The key idea this paper uses to solve the all-relevant problem is to determine the relevance of features by creating random features and using the relevance of these random features as a baseline. The actual features are compared to this baseline and any feature which has a higher relevance than the baseline in included in the set of relevant features. The relevance of the random features is only non-zero due to random fluctuations.

The performance of the new algorithm is only demonstrated on one dataset and the performance is compared to random forest classifiers using different sets of attributes. This paper differs from the other papers looked at in this section because it is the only paper that solves the all-relevant problem instead of solving the minimal-optimal problem. This paper is also the only one that comes with a ready to go implementation of the algorithm in the form of the Boruta package in R.

3.3 A framework for feature selection though boosting

The paper begins by explaining the motivation for the work in the paper [1]. This motivation is the increasing size of datasets as explained in the introduction section of this paper. The paper gives the relevant background information by explaining the use of boosting to modify the feature space in wrapper feature selection methods. Then, it builds on this idea and identifies three specific novelties it contributes to the research field.

The first contribution it makes that builds on previous studies is the use of a weighting strategy that assigns each sample a weight that is inversely proportional to its prediction probability. This is an improvement on previous work, where every misclassified sample had its weight adjusted by the same amount, as it means that the more incorrectly a sample is classified, the greater weight it is given in the next iteration.

To obtain the feature scores, a gradient boosting tree model, XGBoost, is used [3]. It has been shown to be more predictive in larger datasets and gives feature scores that more accurately account for complex interactions between features than individual trees.

The second contribution the paper claims to make is to decouple feature selection from ranking. This is achieved by using a two-step process to determine the best feature for each iteration. The first step is consistent with previous research as a tree model is trained on all features and the top-ranked features are then obtained from the embedded scores. The second step, inspired by Iterative Input Selection (IIS), is to use a classifier to evaluate the performance of all the high ranking features obtained in the previous step. The benefits of this two-step process are that it helps overcome inconsistencies in the original feature rankings and improves robustness. Another advantage is that it helps to nullify the negative effects of feature redundancy in large datasets.

The final contribution the paper claims to make is a method to prevent the premature stopping of the algorithm. Another idea taken from IIS used in FeatBoost is that a feature is not removed from the list of candidate features once it has been selected. This works as a stopping condition because the algorithm terminates if the same feature is selected twice.

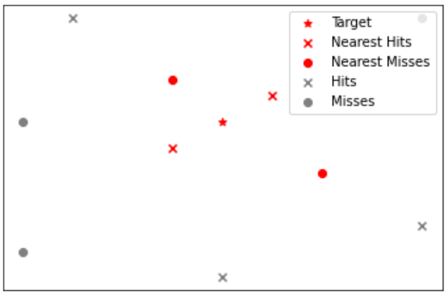


Fig. 2: Example of ReliefF algorithm selecting $k = 2$ nearest hits and misses.

4 METHODOLOGY

In order to comprehend the results, there needs to be an understanding of the feature selection methods that are used. So, in this section, the used feature selection algorithms are described.

4.1 ReliefF

The ReliefF algorithm is a filter method, which computes a score for each feature, allowing for the features to be ranked [6]. ReliefF is an extension of the Relief feature selection algorithm, which computes a feature score for all features and ranks them accordingly.

This original Relief algorithm does this by first initializing the weights of all attributes to zero, after which a random target instance R_i is selected. For this target instance R_i, both the nearest 'hit' and nearest 'miss' are found. The nearest hit indicates the sample that is closest to the target instance while having the same class. The nearest miss indicates the sample closest to the target instance while having a different class.

Once the nearest hit and miss are found, the weight vector is updated. If the miss is closer to the target instance than the hit, the weight vector is decreased. The opposite happens when the hit is closer to the target instance than the miss. Relief does this by using the Euclidean (L2) norm distance. Once this algorithm has iterated m times, the weight vector is divided by m, turning it into a relevance vector. If the value of a feature of the relevance vector is larger than a threshold τ, the feature is selected.

As opposed to only finding the nearest hit and miss in the Relief algorithm, ReliefF finds the k-nearest hits and misses. An example of this is shown in Figure 2. ReliefF then updates the weights of all the attributes as it averages the contributions of the k-nearest hits and misses. This way, the algorithm is more robust to noise. Furthermore, ReliefF uses the Manhattan (L1) norm distance instead of the Euclidean (L2) norm used by the Relief algorithm as experiments have shown there is no practical difference based on the choice of norm [9]. ReliefF is also compatible with multi-class problems and can handle incomplete datasets quite well, as it also uses the prior probability P(C) of class C to calculate the weight vector.

4.2 Boruta

The Boruta algorithm is a wrapper around the random forest algorithm, which is an ensemble method that builds multiple decision trees on different samples [7]. The output of the random forest is thus the class that occurs most often in those decision trees. Boruta uses the Z-score of the random forest algorithm.

Boruta uses shadow attributes, which shuffle the values of the original features around randomly, to measure whether certain features are relevant. In essence, Boruta adds extra randomness to the dataset and then uses a majority vote to reduce the possible impact random fluctuations can have on computing relevant features.

Once the shadow attributes are added to the dataset, a random forest is fitted on it. When this is done, the Z-scores are gathered, which are calculated by dividing the average loss by the standard deviation. The maximum Z-score among shadow attributes (MZSA) is found and used as a threshold. If the Z-score of a non-shadow feature is higher than this threshold, it is deemed important. If it is not, it is deemed unimportant, as it means that the feature is not able to perform better than the best-randomized feature. This labelling is done for all features, and when it is done, the shadow attributes are removed. A note should be made that one run of Boruta is not enough to ensure reliable results, as noise may influence the results, so more iterations are done. Finally, we end up with only the features that are deemed important by Boruta. An example is shown in Figure 3, where LSTAT, the percentage of lower status of the population, and RM, the average number of rooms per dwelling, are shown to have the highest Z-score. The green features are the important features, whereas the red ones are removed, as they perform worse than the blue shadow features.

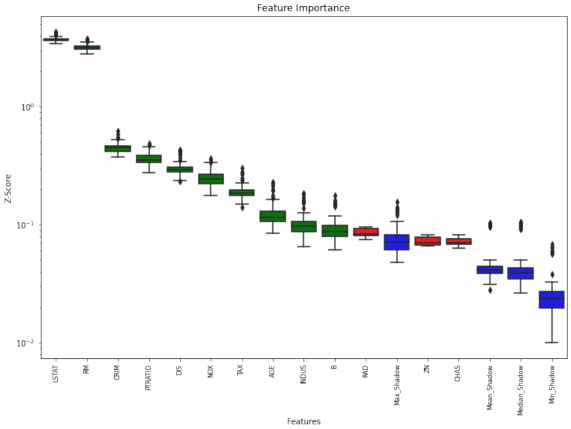


Fig. 3: Example of Z-scores of features calculated by Boruta on Boston housing dataset.

4.3 XGBoost

XGBoost belongs to the class of gradient boosting tree algorithms [3]. Since its inception, it quickly became the go-to method for solving machine learning problems due to its efficiency and predictive power.

In gradient boosting tree algorithms, ensembles are constructed in an iterative training process. Each iteration, one tree is added to the ensemble. Each new tree is fit to correct the misclassifications made by the prior models. The loss function is calculated using gradient descent, and the tree that minimizes the loss function is then added to the ensemble.

For XGBoost, each tree in the ensemble is a regression tree with continuous scores on each leaf. For each example, the decision rules in the trees are used to classify, which results in the leaves having scores. To calculate the final prediction, the scores in the corresponding leaves are summed. XGBoost uses the following cost function:

$$Cost = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \tag{2}$$

where $\Omega(f)$ is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2, \tag{3}$$

where T is the number of leaves on each tree, and γ is the regularization parameter. Furthermore, each f_k corresponds to an independent tree structure q and leaf weights w.

The first term is a loss function that measures the distance between the predicted and correct value. The second term is a regularization term that punishes the complexity of the model. This term helps to avoid problems with overfitting.

The model is trained in an additive manner. For each iteration, the f_t that most improves the model according to the loss function previously defined is added to minimize the following objective:

$$Cost = \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_k \Omega(f_t), \tag{4}$$

where $\hat{y}_i^{(t-1)}$ is the prediction of the i-th instance at the t-th iteration.

The tree which best minimizes the objective is found using gradient descent.

The best split needs to be found at iteration. XGBoost can use either an exact greedy algorithm that looks at every possible split on every possible feature or an approximate algorithm that is much quicker but does not explore the whole search space.

4.4 FeatBoost

FeatBoost expands on boosting algorithms [1]. It uses tree ensemble models and ranks candidate features based on the embedded feature relevance scores of these tree models.

FeatBoost can use any tree-based classifier, but in the paper that introduced the algorithm, the tree-based classifier is XGBoost. The initial weights are set to $\frac{1}{n}$. This tree-based classifier, H_1 is then trained on all the examples in the training set, which produces an initial ranking of all the features.

The top m features are evaluated and compared as single-input tree-based classifiers, H_2. The best performing feature is then added to the subset of selected features. For any iteration after this initial one, H_2 is used to evaluate each of the m features.

Lastly, H_1 is trained on all the selected features. Then the sample weights for misclassified samples are updated inversely proportional to their predicted probability.

The training iterations continue until either the increase in accuracy between two iterations is below the user-defined threshold or until the same feature is picked twice. In this case, the weights are set to their initial values, and the iteration is repeated. If this leads to an improvement, the iterations continue, and if it does not, the algorithm stops.

5 Experimental Settings and Evaluation

In this section, the datasets used are described, after which the validation process is detailed. Finally, the settings used for the experiments conducted by Alsahaf et al. are described [1].

5.1 Datasets

For the experiments, 8 real datasets are used, as well as a synthetic one called Madelon [5]. The datasets' details and dimensions are described in Table 1.

Table 1: Datasets used in comparison. Obtained from ASU feature selection repository [1].

Name	#Features	#Samples	#Classes	Domain
Madelon	500	2600	2	Synthetic
Isolet	617	2560	2	Speech Recognition
PCMAC	3289	1943	2	Text
Relathe	4322	1427	2	Text
Basehock	4862	1993	2	Text
Coil20	1024	1440	20	Face Image
ORL	1024	400	40	Face Image
WarpPIE10P	2420	210	10	Face Image
Pixar10P	10000	100	10	Face Image

5.2 Validation Process

To compare the performances of the feature selection methods, we take a look at their average computation times and the redundancy rate (RED) of the subsets of features. This is measured through the following formula, which was designed and used by both Yamada et al. and Zhao et al. [13] [15]:

$$RED(\mathscr{X}) = \frac{1}{p(p-1)} \sum_{f_i f_j \in \mathscr{X}, i>j} |p(f_i, f_j)|, \tag{5}$$

where $p(f_i, f_j)$ is the Pearson correlation coefficient between features f_i and f_j. A smaller value of $RED(\mathscr{X})$ is best, as it means that the subset of $\mathscr{X}$ has low redundancy, suggesting that the subset of features are correctly selected.

To evaluate the different feature selection algorithms, a table is made for each method of the average computation time, as well as the average redundancy rates for up to the top 10 and top 100 features for each method. The analysis is done on the resulting tables.

5.3 Settings

The settings for the algorithms are as follows:

1. **XGBoost**: Most of XGBoosts parameters are set to default, except for a value of 100 trees, combined with a maximum tree depth of 20.

2. **FeatBoost**: FeatBoost is used twice for the comparison: one with an XGBoost classifier, with the same settings as the algorithm itself, as well as one with a Nearest Neighbour classifier. The parameters of this NN classifier are set to: $k = 3$, $m = 50$, $p' = 100$, and $e = 10^{-18}$

3. **Boruta**: Boruta uses XGBoost as the base ranking algorithm instead of random forest. The parameters of the XGBoost algorithm within Boruta are the same as the XGBoost algorithm itself, with default values in the remaining parameters.

4. **ReliefF**: ReliefF has the number of neighbours (hits/misses) set to 10.

The specifics of the machine on which the experiments are run, are not mentioned in the paper by Alsahaf et al [1].

6 Results and discussion

Table 2 shows the average computation times in minutes for each feature selection algorithm, tested on all datasets. XGBoost is shown to perform the quickest for all datasets, which makes sense as it only needs one fitting. ReliefF is the second-fastest as it can choose features in linear time, no matter the type of data. It is, followed by FeatBoost with a NN classifier, which is faster than FeatBoost with an XGBoost classifier. This result is also expected, as a NN classifier is a less complex, quicker algorithm than XGBoost. Finally, we see that FeatBoost with an XGBoost classifier is only slightly faster than Boruta on some datasets. Boruta is relatively slow, as computing the importance of all features can take a long time. This depends on the base ranking algorithm, in this case XGBoost, which is a more complex algorithm than NN for example. Choosing random forest as the base ranking algorithm could speed up the process as it builds the trees in parallel.

Table 3 shows the mean and standard deviation of the redundancy rate of up to the top 10 selected features. The FeatBoost algorithm performs the best overall, either with XGBoost or NN as the classifier. Boruta manages to achieve the best performance on one dataset only: WarpPIE10P.

Table 4 shows the redundancy rate of up to the top 100 selected features. FeatBoost with NN as wrapped classifier performs the best but is closely followed by XGBoost. Boruta and FeatBoost with XGBoost as classifier only achieve the best performance in one dataset each.

It is interesting to note that FeatBoost performs equally as well with NN as classifier as it does with XGBoost as classifier. Because FeatBoost with NN is significantly faster than FeatBoost with XGBoost, it might be advisable to use FeatBoost with NN if you were to choose between the two. Choosing a somewhat simpler classifier does not impact the performance negatively.

Another thing to note is that while XGBoost does not perform the best when it comes to the mean redundancy rate of up to the top 10 selected features, it does perform quite well when more features are

Table 2: Mean and standard deviation of computation time (in minutes) of each feature selection algorithm [1].

	FeatBoost (H_2 = XGB)	FeatBoost (H_2 = NN)	XGBoost	ReliefF	Boruta
Madelon ($p = 500$, $n = 2600$)	58.71 (11.89)	3.56 (0.36)	**0.25 (0.01)**	1.02 (0.02)	20.81 (3.01)
Isolet ($p = 617$, $n = 1560$)	168.45 (31.62)	52.53 (21.78)	**1.04 (0.02)**	0.72 (0.01)	154.91 (3.23)
PCMAC ($p = 3289$, $n = 1943$)	186.07 (57.88)	45.07 (12.04)	**1.11 (0.03)**	2.81 (0.08)	105.67 (4.79)
Relathe ($p = 3289$, $n = 1943$)	163.38 (32.6)	60.92 (21.09)	**1.08 (0.01)**	2.77 (0.03)	91.73 (3.06)
Basehock ($p = 4862$, $n = 1993$)	217.52 (40.95)	73.33 (24.3)	**1.54 (0.07)**	4.34 (0.1)	131.14 (4.76)
Coil20 ($p = 1024$, $n = 1440$)	333.85 (60.68)	36.67 (9.27)	**1.18 (0.04)**	2.19 (0.06)	241.23 (17.47)
ORL ($p = 1024$, $n = 400$)	168.38 (48.91)	18.18 (4.65)	**0.56 (0.03)**	0.57 (0.03)	102.35 (12.82)
WarpPIE10P ($p = 2420$, $n = 210$)	22.91 (4.88)	5.27 (1.45)	**0.25 (0)**	3.28 (0.05)	24.12 (1.06)
Pixraw10P ($p = 10000$, $n = 100$)	6.91 (1.51)	3.4 (0.49)	**0.37 (0.01)**	5.87 (0.08)	41.15 (0.7)

involved. It does, however, perform worse when the number of features is significantly greater than the number of samples, for example in the ORL, WarpPIE10P and Pixraw10P datasets. This is because there is simply less to learn from for every feature for XGBoost.

All in all, FeatBoost and XGBoost are capable of reducing redundancy quite effectively. Furthermore, XGBoost is also quite efficient when it comes to average computation time.

7 Conclusion

The importance of feature selection and ranking cannot be understated with the increasing sizes of datasets nowadays. That is why, in this paper, we examined different methods of feature selection and ranking algorithms, selected four different feature selection algorithms in FeatBoost, XGBoost, ReliefF and Boruta, and compared their performances based on their computation times and redundancy rates.

By comparing these algorithms, we found that XGBoost had the fastest average computation times over all the datasets. Moreover, FeatBoost performed the best when it came to reducing redundancy in the features, either with NN as classifier of with XGBoost as classifier. However, FeatBoost with NN as classifier performed just as good as FeatBoost with XGBoost as classifier, while also being quicker. ReliefF showed to have one of the fastest average computation times, but was not the best in reducing the redundancy in any of the datasets. Boruta was one of the slowest feature selection algorithms, and only managed to impress in reducing the redundancy in one dataset.

8 Future Works

In future research, a few things could be done differently when making a review of feature selection and ranking methods. First of all, in this review, a few different feature selection methods were picked and compared. Although the methods differ significantly from one another, more methods exist and might be better than the methods compared in this paper. Secondly, FeatBoost used XGBoost and Nearest Neighbours as a classifier but can be tried out and compared once again with other algorithms as the classifier. This might give different results. Thirdly, the parameters used for the feature selection methods used in this paper can also be changed and cause differences in the results. Finally, more datasets with more variety, such as more features, more classes, or even more samples, can be used and tried out.

Acknowledgements

The authors wish to thank our expert reviewer Dr. G. Azzopardi for providing us with the source material, as well as giving us valuable feedback. The authors also wish to thank Arjan Dekker and Anthonin Thioux for reviewing our paper and providing us with valuable feedback.

References

[1] A. Alsahaf, N. Petkov, V. Shenoy, and G. Azzopardi. A framework for feature selection through boosting. *Expert Systems with Applications*, 187, Jan. 2022.

[2] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[3] T. Chen and C. Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016.

[4] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182, mar 2003.

[5] I. Guyon, J. Li, T. Mader, P. Pletscher, G. Schneider, and M. Uhr. Feature selection with the clop package. 01 2006.

[6] K. Kira and L. A. Rendell. A practical approach to feature selection. In D. Sleeman and P. Edwards, editors, *Machine Learning Proceedings 1992*, pages 249–256. Morgan Kaufmann, San Francisco (CA), 1992.

[7] M. B. Kursa and W. R. Rudnicki. Feature selection with the boruta package. *Journal of Statistical Software*, 36:1–13, 2010.

[8] A. Y. Ng. Feature selection, l_1 vs. l_2 regularization and rotational invariance. 2004.

[9] R. J. Urbanowicz, M. Meeker, W. L. Cava, R. S. Olson, and J. H. Moore. Relief-based feature selection: Introduction and review. *Journal of Biomedical Informatics*, 85(1):189–203, 2018.

[10] M. M. Usman, O. Owolabi, and A. Ajibola. Feature selection: It importance in performance prediction. 2020.

[11] A. von See. Amount of data created, consumed, stored 2010-2025, Jun 2021.

[12] B. Xue, M. Zhang, and W. Browne. A comprehensive comparison on evolutionary feature selection approaches to classification. *International Journal of Computational Intelligence and Applications*, 14(02), 2015.

[13] M. Yamada, W. Jitkrittum, L. Sigal, E. P. Xing, and M. Sugiyama. High-dimensional feature selection by feature-wise kernelized lasso. *Neural Computation*, 26(1):185–207, Jan 2014.

[14] H. Zare, G. Haffari, A. Gupta, and R. Brinkman. Scoring relevancy of features based on combinatorial analysis of lasso with application to lymphoma diagnosis. *BMC Genomics*, 14, 01 2013.

[15] Z. Zhao, L. Wang, and H. Liu. Efficient spectral feature selection with minimum redundancy. In *AAAI*, 2010.

Table 3: Mean and standard deviation of redundancy rate of up to the top 10 selected features of each algorithm [1].

	FeatBoost (H_2 = XGB)	FeatBoost (H_2 = NN)	XGBoost	ReliefF	Boruta
Madelon ($p = 500$, $n = 2600$)	**0.125 (0.023)**	0.144 (0.029)	0.139 (0.0022)	0.172 (0.005)	0.145 (0.028)
Isolet ($p = 617$, $n = 1560$)	0.074 (0.009)	**0.058 (0.012)**	0.067 (0.007)	0.59 (0.005)	0.066 (0.003)
PCMAC ($p = 3289$, $n = 1943$)	0.016 (0.008)	**0.007 (0.004)**	0.021 (0.006)	0.17 (0.003)	0.021 (0.006)
Relathe ($p = 3289$, $n = 1943$)	**0.017 (0.006)**	0.018 (0.005)	0.019 (0.004)	0.024 (0.002)	0.017 (0.005)
Basehock ($p = 4862$, $n = 1993$)	0.008 (0.002)	**0.006 (0.003)**	0.007 (0.002)	0.008 (0.002)	0.007 (0.001)
Coil20 ($p = 1024$, $n = 1440$)	**0.118 (0.021)**	0.139 (0.031)	0.122 (0.014)	0.176 (0.018)	0.127 (0.021)
ORL ($p = 1024$, $n = 400$)	0.162 (0.022)	**0.14 (0.022)**	0.148 (0.009)	0.166 (0.008)	0.142 (0.006)
WarpPIE10P ($p = 2420$, $n = 210$)	0.32 (0.029)	0.287 (0.019)	0.281 (0.007)	0.286 (0.006)	**0.274 (0.016)**
Pixraw10P ($p = 10000$, $n = 100$)	**0.431 (0.033)**	0.444 (0.03)	0.466 (0.016)	0.474 (0.004)	0.47 (0.001)

Table 4: Mean and standard deviation of redundancy rate of up to the top 100 selected features of each algorithm [1].

	FeatBoost (H_2 = XGB)	FeatBoost (H_2 = NN)	XGBoost	ReliefF	Boruta
Madelon ($p = 500$, $n = 2600$)	0.12 (0.027)	0.144 (0.029)	**0.013 (0)**	0.014 (0)	0.137 (0.024)
Isolet ($p = 617$, $n = 1560$)	0.076 (0.005)	**0.058 (0.012)**	0.104 (0.003)	0.093 (0)	0.066 (0.005)
PCMAC ($p = 3289$, $n = 1943$)	0.015 (0.008)	**0.007 (0.004)**	0.012 (0.002)	0.011 (0.001)	0.02 (0.005)
Relathe ($p = 3289$, $n = 1943$)	0.016 (0.005)	0.018 (0.005)	**0.013 (0.001)**	0.014 (0.001)	0.017 (0.005)
Basehock ($p = 4862$, $n = 1993$)	0.008 (0.002)	**0.006 (0.003)**	0.011 (0.001)	0.012 (0.001)	0.008 (0.002)
Coil20 ($p = 1024$, $n = 1440$)	0.118 (0.021)	0.139 (0.031)	**0.113 (0.004)**	0.122 (0.004)	0.124 (0.017)
ORL ($p = 1024$, $n = 400$)	0.16 (0.022)	**0.14 (0.022)**	0.154 (0.006)	0.158 (0.005)	0.142 (0.009)
WarpPIE10P ($p = 2420$, $n = 210$)	0.32 (0.026)	0.287 (0.019)	0.31 (0.008)	0.319 (0.004)	**0.275 (0.018)**
Pixraw10P ($p = 10000$, $n = 100$)	**0.433 (0.03)**	0.444 (0.03)	0.44 (0.011)	0.44 (0.006)	0.471 (0.001)

Comparing Parallel Algorithms for Topological Watershed

Abel Nissen (s3724786), Christopher Worthington (s3715086)

Abstract— The watershed transform is a common technique used to segment images based on grey values. The topological watershed is a variant which utilizes the grey level contrast of the original image. In this paper, we look at two implementations of parallel algorithms available today for the topological watershed. First, we explain the topological watershed and discuss related work in the field of the regular watershed algorithm. Second, we consider general steps taken towards parallelisation. Finally, we quantitatively and qualitatively compare the two implementations. This comparison lead us to conclude that the two implementations at their respective ideal speed-up perform remarkably similar, with the older implementation coming out on top because it allows for scaling when adding more threads.

Index Terms— Watershed transform, topological watershed, parallelisation, image segmentation.

1 Introduction

1.1 Watershed Algorithms

The watershed transform, proposed originally by Digabel and Lantuejoul [1], is a popular tool used to segment grey-scale images into regions with similar grey values. An extension of this algorithm is the topological watershed, which additionally preserves more information about the contrast of the original image [2], for example, some of the grey-scale information from the original image is preserved, which may be useful in reconnecting corrupted contours.

Watershed algorithms use the concept of watershed basins where land is split based on areas in which water will all flow to the same point, then the points where these catchment basins meet make the watershed lines. See Figure 1 for a visualisation.

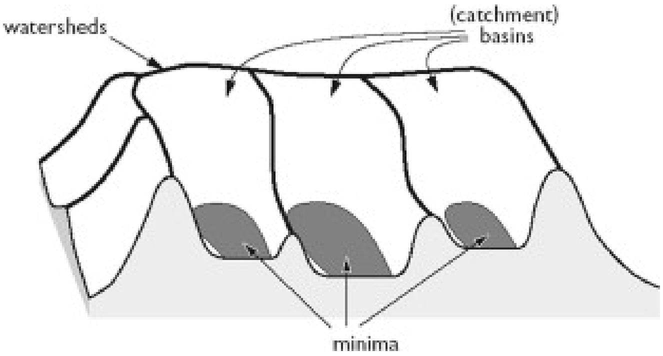


Fig. 1. Visualisation of the watershed principle, showing minima, basins and watershed lines [3]

This is represented in the watershed transform by a binary image with zeros being the basins and ones being the watershed lines. This can be sequentially calculated using a queue and iterating over grey levels to continually label the basins, in a way that is like filling the basins with water. Then at any point that two basins meet, a watershed line can be placed.

A simple visual example of the application of the watershed algorithm can be seen in Figure 2. Here, all of the different coins in the digital image are uniquely identified and separated in the output. While only being a simple example, this conveys the main idea of the watershed algorithm, in partitioning a digital image into multiple segments.

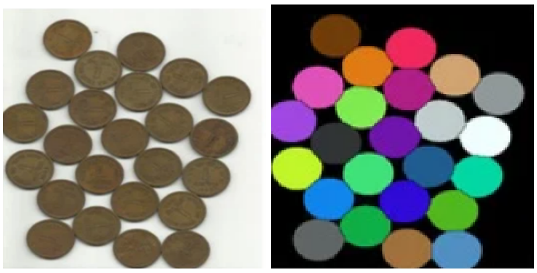
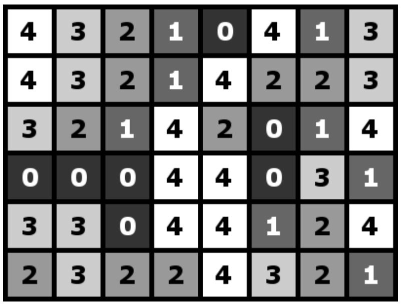


Fig. 2. Visual application of the watershed algorithm on a digital image of coins [4]

In the topological watershed, the original grey levels of each pixel in the digital image are used for the resulting output. Basins are labelled by the grey level of their lowest point (or, pixel), rather than simply zero. Furthermore, each point on the watershed lines have the value of the lowest path between the two minima it separates, preserving a contrast between the basins.

1.2 Example Transformation

In order to properly illustrate the functions of both the regular watershed transform, and the topological watershed transform, we will give an example on a simple grey-scale image. See Figure 3 for the grey image as it is before any transformations have been applied.



4	3	2	1	0	4	1	3
4	3	2	1	4	2	2	3
3	2	1	4	2	0	1	4
0	0	0	4	4	0	3	1
3	3	0	4	4	1	2	4
2	3	2	2	4	3	2	1

Fig. 3. Original grey-scale pixel values of a digital image [5]

The watershed transform, as described before, is a binary image that we can see in Figure 4. Here the image has been segmented into

- *Abel Nissen is an MSc Computing Science student of the University of Groningen, E-mail: a.j.nissen@student.rug.nl*
- *Christopher Worthington is an MSc Computing Science student of the University of Groningen, E-mail: c.j.worthington@student.rug.nl*

7 basins, varying in size. It is also clear that the watershed lines often appear on the higher grey levels in the image, but can appear on lower grey levels when there are minima near to each other.

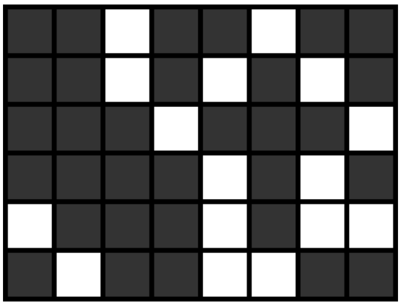
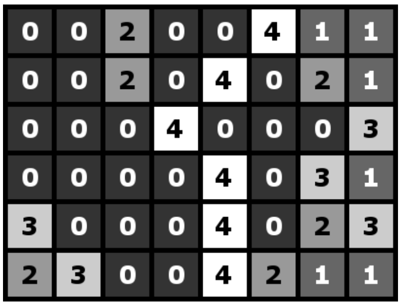


Fig. 4. Watershed transform of Figure 3 [5]

Finally, the topological watershed of the image is not just binary and instead keeps more grey level information, as can be seen in Figure 5. It can be observed that the basins are not only zero, with three of the basins having the value one and one having the value two. Then the values on the watershed lines give some information on the contrast between the basins.



0	0	2	0	0	4	1	1
0	0	2	0	4	0	2	1
0	0	0	4	0	0	0	3
0	0	0	0	4	0	3	1
3	0	0	0	4	0	2	3
2	3	0	0	4	2	1	1

Fig. 5. Topological watershed of Figure 3 [5]

1.3 Parallelisation

Due to the constantly increasing image sizes, multiple parallel algorithms have been designed for different watershed transform implementations [5]. However, in stark contrast, very few parallel algorithms have been designed for the topological watershed. Of the few implementations that have been made, we will be comparing the designs introduced in two papers. One paper from 2011 [5]. And the other from 2017 [6]. These papers describe parallel algorithms for the topological watershed by parallelising certain stages of the sequential algorithm. We are interested to see how the two algorithms compare.

Thus, the aim of this paper is to provide a review analysis of these techniques, to find whether both implementations are actually different, and if one is more effective than the other overall. We will discuss and judge the two algorithms based on their complexity, cost of computing resources, and efficiency. To do this we will break down each algorithm into its individual components, to be able to compare them side by side in separate parts. This will provide a guideline between the two implementations, which in the field of image segmentation could assist in choosing the most suitable approach for the needs of a professional, as they can be context dependent.

This paper is structured as follows: Section 2 addresses work that is related to parallelisation of watershed and other image processing algorithms. Section 3 describes the preliminary information required for parallelisation of sequential algorithms. Then, Section 4 provides an overview of two state-of-the-art techniques. Section 5 presents a detailed comparison and discussion of the two approaches, where we will be looking at the comparative speed-up. Finally, Section 6 offers the conclusions of our investigation, and Section 7 suggests possible extensions for the research conducted in this paper.

2 Related Work

This paper focuses on the parallelisation of the topological watershed, however, it is useful to discuss some related parallelisation strategies. The nearest related algorithm here is simply the standard watershed transform. Several parallelisation strategies for this algorithm are outlined in [7]. The paper discusses both distributed and shared memory parallel implementations of the watershed transform. It was found that the immersion strategy, as briefly described in Section , for the watershed transform, an implementation of which can be found in [8], is not easy to parallelise due to its use of a FIFO queue. Alternatively, using structures containing topological distances and modified UNION-FIND algorithms [9] allowed for multiple parallel implementations. The paper finds that, in the end, due to a global operator always being required in any watershed implementation, a moderate speed-up is always to be expected. This is especially apparent with spiral structures such as in Figure 6, where many basins must be joined together when compiling the separate watersheds that have been produced across a parallel system. This is something we can expect to persist for the topological watersheds due to it also needing a global operator and having the same joining of basins .

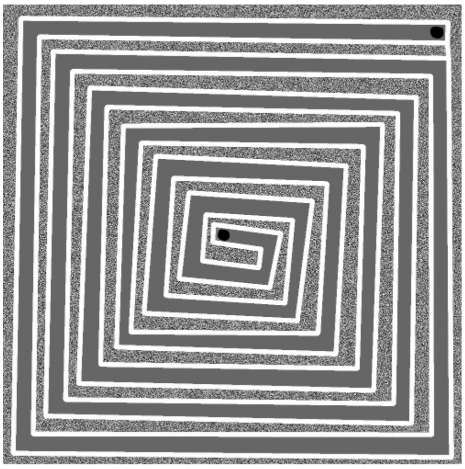


Fig. 6. Grey-scale image with a spiralling plateau [5]

In addition, [10] gives a good general solution for this issue of non-separable and global morphological attribute filters. The paper uses Min and Max-Tree algorithms to achieve efficient parallelism for many different attribute filters. This is pivotal to the parallelisation of the topological watershed implementation given in [5], which makes use of a Min-Tree algorithm.

3 Preliminaries

Going back to the early years of personal computers, processors had only a singular core. At the time, development was quick, with Moore's law holding up, or over achieving it for many years [11]. See Figure 7 from Intel that illustrates this development. This meant that for a long time, to make a program run faster, research went into designing a chip with faster single core performance.

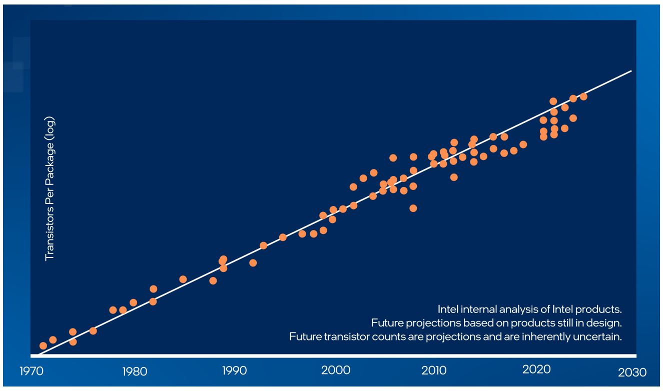


Fig. 7. Moore's law being upheld, showing the number of transistors per package, per year [12]

Parallelism did not exist on ordinary computers, the first requirement for any parallel implementation is access to a computer with multiple processor cores, i.e. a multiprocessing system, which were unavailable to the general public. However, in recent years, we have seen a major shift from high clock-speed single core processors, to somewhat lower, but still ever increasing, clock-speed multi-core processors. See Figure 8 and Figure 9.

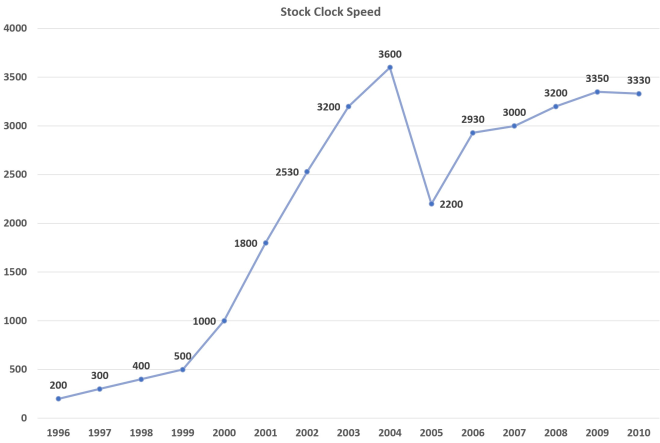


Fig. 8. Average stock clock speed of processors throughout the years [13]

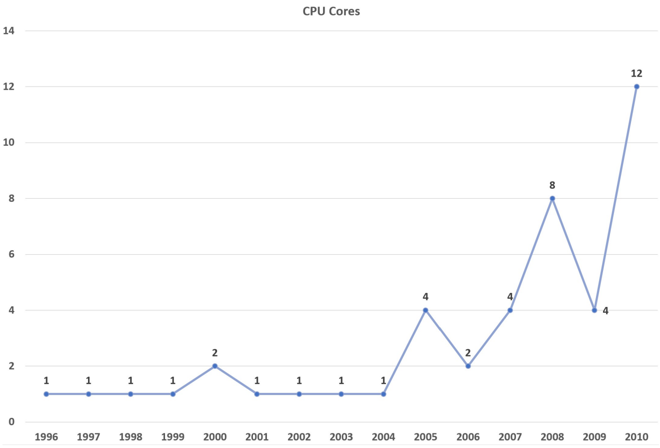


Fig. 9. Available number of CPU cores on processors throughout the years [13]

With this shift towards multiple cores per processor came a massive increase in the available cache in a processor, which ensures more optimal access speeds across the multiple cores. This change made parallelisation on smaller systems feasible. In [6, 14] gives 5 clear steps for the process of parallelising a sequential algorithm. These 5 steps also aim for the parallel algorithm running on multiple cores to have better performance than that of the sequential algorithm running on a single core. The steps are as follows:

1. **Finding concurrency design space.** The first step towards parallelising an algorithm is to understand different aspects of the sequential implementation. Think of discovering the concurrency of tasks, or groups of tasks, and the allocation of data (is it shared, or task-local).

2. **Algorithm design space.** With the extracted concurrency of the sequential algorithm, algorithm designs and design parameters need to be set. In many cases, the algorithms can be decomposed to a finite set of tasks, which can then be grouped according to several criteria, often applying the well-known patterns of divide-and-conquer.

3. **Architecture design space.** Before moving to coding, it is important to find the applicable design architecture for the algorithm. This design represents a standard classification of parallel computer systems. There are four different classifications: SISD (Single Instruction, Single Data), SIMD (Single Instruction, Multiple Data), MISD (Multiple Instruction, Single Data) and MIMD (Multiple Instruction, Multiple Data). Depending on the classification of the algorithm, different techniques provides the most effective parallelisation.

4. **Parallel implementation mechanisms.** When writing the parallel version of the algorithm, several tools can be used to aid the programmer. These tools can manage threads or processes of the algorithm, and guarantee the internal synchronization and communication.

5. **Performance metrics of parallel programs.** Once the parallel version of the algorithm is complete, several metrics need to be considered to measure its effectiveness, or speed-up. Therefore a set of measurements that quantifies the parallel code such as efficiency, scalability and portability need to be considered.

4 Topological Watershed Parallelisation Techniques

As previously introduced, the topological watershed was introduced as a version of the regular watershed algorithm. It includes grey level information in the end result in such a way that the significance of each watershed line is preserved. When the grey level of a pixel is related to a height value on a map, significance is defined as the difference in height between the watershed pixel and the basin pixel. Both papers [5, 6] introduce a parallel implementation of the topological watershed, albeit using different techniques. Specifically, [5] makes use of a Min-tree:

A Min-tree is equivalent to the Max-tree of an inverted image. Using the parallel algorithm described in [10], the authors create a Min-tree of a grey-scale image. This Min-tree easily allows one to check if two pixels are separated. Specifically, the authors mention that the separation is related to the altitude of the Lowest Common Ancestor. Using the Min-tree, after prepocessing, this Lowest Common Ancestor can be found in constant time. The key element of the algorithm is represented in Figure 10.

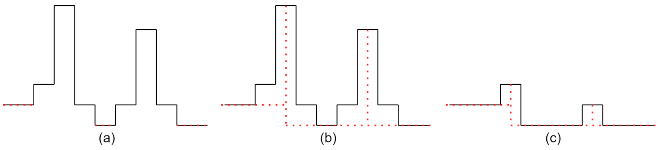


Fig. 10. Topological watershed abstraction: (a) initial height map of pixels; (b) identified watershed lines and basins; (c) topological watershed image

Using the Min-tree, the algorithm is able to identify the pixel of the watershed line between two basin pixels (Fig. 10.b). It does this using a Lowest Common Ancestor (LCA) algorithm that finds the shortest path between two basin minima. All the pixels in between are marked as *W-Destructible* and consequently have their height (or grey value) lowered to the height of the corresponding basin (Fig. 10.c). Because the function that lowers the *W-Destructible* pixels is a local function, the authors are able to parallelise the topological watershed algorithm for n threads, simply by dividing the image into n tiles and assigning one tile to each thread.

On the other hand, in [6] the parallel implementation makes use of a Split, Distribute, and Merge (SD&M) strategy. The splitting is different to the usual by pixel or block division, the source selection is instead completely random. The notion of streams, from [15], is used to calculate the watershed. First, the flow between bottoms and tops are identified, and complementary flows are grouped in so called flow families [6]. These can be arranged in different ways to aid with the partitioning of their vertex representations. By picking the unique minimum of a flow family, the set of vertices not related to this minimum can be considered as a flow-cut, which is equivalent to a watershed-cut, or simply watershed lines. The computation of streams across sources is fully parallel, then the merging step allows the computation of the full watershed with two by two streams fusion. The method is applicable in general to watershed algorithms, however, the paper gives a specific analysis of the method implemented for the parallel topological watershed. A general illustration of the flow-cut, or watershed calculation, is given in Figure 11.

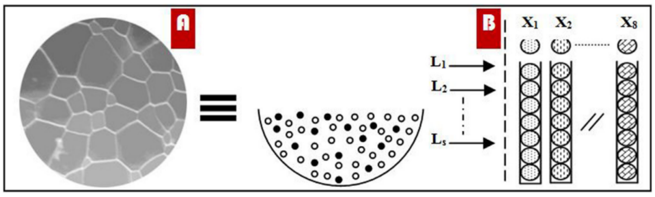


Fig. 11. Flow-cut abstraction: (a) partition of the input image into flows; (b) parallel stream computation of the flow-cut [6]

5 Comparison and Discussion

Now that the differences in implementation between the two parallel topological watershed algorithms are clear, it is important to see how these compare using real world tests. While the authors of the two papers did not use the same testing techniques, their results contain enough data, and similarities in that data, to be able to equivocally compare them.

5.1 Testing Framework

Let us start with the testing framework used in both papers. Despite the difference in publication date of six years between the two papers, [5] from 2011 uses a more powerful testing setup: a 4-socket, 6-core per sockets, AMD Opteron based machine with 128GB of memory, using 1 to 24 threads in various combinations. Figure 12 is retrieved from the paper and shows the the achieved speed-up per each processor thread used.

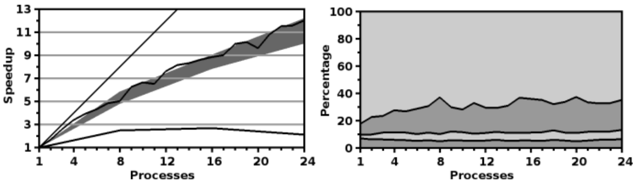


Fig. 12. Behaviour of the parallel algorithm in total speed-ups per threads in the performed tests, relative to the wall-clock time for a single thread with the same input [5]

But where Figure 12 shows the achieved overall speed-up, additionally, the paper provides another figure which shows the speed-up of the algorithm in four stages. This can be seen in Figure 13.

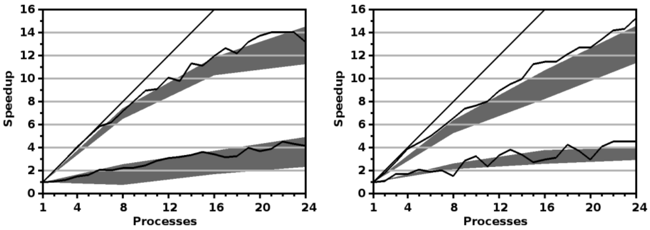
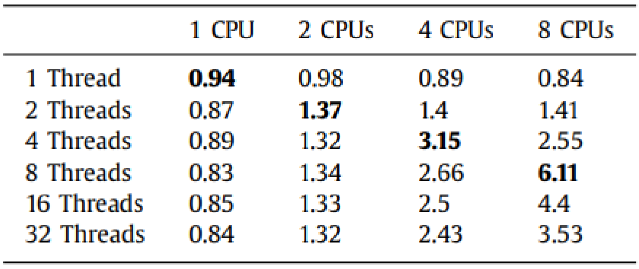


Fig. 13. Speed-ups per utilised threads in the four stages of the parallel algorithm, where each grey area represents a different stage [5]

Here, each grey area represents a stage of the algorithm. Similar to Figure 12, the diagonal line shows the ideal speed-up, where the speed-up is equal to the number of threads. Starting in the left side of Figure 13, going from top to bottom and doing the same for the right side, the four stages are: min-tree construction, tree compression, topological watershed computation, Lowest Common

Ancestor pre-processing.

In [6], 2017, the setup consists of four different processors: Intel P4-660, Intel Dual C. E8400, Intel C2 Quad E5335 and an Intel Xeon E5405. Respectively, they have 1, 2, 4 and 2 by 4 processor cores. In this way, the researchers were able to compare their algorithm at thread intervals at multiples of 2. Figure 14 shows the achieved speed-up per utilized processor thread in their version of the parallel algorithm.



	1 CPU	2 CPUs	4 CPUs	8 CPUs
1 Thread	**0.94**	0.98	0.89	0.84
2 Threads	0.87	**1.37**	1.4	1.41
4 Threads	0.89	1.32	**3.15**	2.55
8 Threads	0.83	1.34	2.66	**6.11**
16 Threads	0.85	1.33	2.5	4.4
32 Threads	0.84	1.32	2.43	3.53

Fig. 14. Performance improvement per thread utilisation [6]

5.2 Achievements

Henceforth, when referencing the papers [5, 6], for ease of readability we will be calling them the 2011 and the 2017 implementation, respectively.

We can see that the optimal speed-up in the 2017 implementation is achieved at 8 CPUs, using 8 threads. Here, adding more threads does not increase the speed-up, but rather the addition of more cores seems to be the contribution to speed-up. In contrast, in the 2011 implementation the speed-up increases with the addition of more threads, albeit with diminishing returns.

We can see that at the same thread count, in the ideal performance of the 2017 implementation with 8 threads, the two implementations perform differently. The 2017 implementation achieves a speed-up of 6.11, whereas the 2011 implementation achieves a speed-up of roughly 8, when utilizing the same 8 threads. However, Figure 12 does not show the full picture. When looking at Figure 13, we can see that the optimal speed-up of the 2011 implementation is achieved when using 6 threads. The min-tree construction stage of the algorithm even matches the ideal speed-up up to these 6 threads. At these 6 threads, the achieved speed-up equates to roughly 6.

Thus, when comparing the two implementations at their respective ideal performances, they perform nearly identical. Given the fact that the 2011 implementation continues to scale when adding more threads, this would give it a slight edge over the 2017 implementation when looking at raw speed-up performance. With the algorithm allowing for scaling above 24 threads. However, a very important detail would then be forgotten. In the 2017 implementation, the researchers used per CPU only 2GB of memory, with a maximum of 8GB for the 2 by 4 model. In terms of resource availability, this is exceeded enormously in the 2011 implementation, which has 16 times as much memory to its disposal. Communication with one of the authors of [5] specified however, that this total 128GB of memory was simply available for the system. The algorithm does not, in fact, require this entire memory. No more than 20 to 30 times the image size in memory would be required.

6 Conclusion

After an analysis of the parallelisation strategies and resulting performances of the algorithms in [5, 6] we have found that the first paper achieves similar, if not better results when compared with the newer paper. The 2017 implementation seems to achieve its best speed up at one thread per CPU, achieving an optimal speed-up at 8 threads, whereas the 2011 implementation still gains speed-up at 24 threads, with the promise of getting even faster computation the more resources are allocated to it.

Therefore, we can conclude that the 2017 algorithm is better for experimenting on personal computers with restricted resources. And that the 2011 implementation should be used in situations where hardware resources are not a constraint, and where time or speed are of the utmost importance. An example of such a situation could be where a natural disaster has taken place, and image segmentation of a huge area is needed to derive the best approach for rescue teams and medical personnel.

7 Future Work

Although the aim of this paper was to provide a complete overview of the two implementations of [5, 6], there might be room for improvement. Specifically in the parallelisation of the topological watershed implementation. The paper [7] describes multiple different implementations of the (regular) watershed algorithm, including parallel versions. With the watershed and topological watershed being so similar in their general idea, and even in their implementations, it stands to reason that parallel implementations of the watershed algorithm might give valuable insight in possible new or improved parallelisation implementations of the topological watershed. Therefore, a thorough dive into [7], analysing and quantifying the different approaches, advantages or potential improvements might be very valuable to the continuation of research into the parallelisation of the topological watershed.

Acknowledgements

The authors wish to thank Michael H.F. Wilkinson for his guidance and support.

References

[1] H Digabel and C Lantuejoul. Iterative algorithms, actes du second symposium europeen d'analyse quantitative des microstructures en sciences des materiaux, biologie et medecine, caen, 4-7 october 1977, j.-l. chermant, ed. *Riederer Forlag, Stuttgart*, 1978.

[2] Gilles Bertrand. On topological watersheds. *Journal of Mathematical Imaging and Vision*, 22(2):217–230, 2005.

[3] Visual computing for medicine. In Bernhard Preim and Charl Botha, editors, *Visual Computing for Medicine (Second Edition)*, page i. Morgan Kaufmann, Boston, second edition edition, 2014.

[4] Anton S. Kornilov and Ilia V. Safonov. An overview of watershed algorithm implementations in open source libraries. *Journal of Imaging*, 4(10), 2018.

[5] Joël van Neerbos, Laurent Najman, and Michael H. F. Wilkinson. Towards a parallel topological watershed: First results. In Pierre Soille, Martino Pesaresi, and Georgios K. Ouzounis, editors, *Mathematical Morphology and Its Applications to Image and Signal Processing*, pages 248–259, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[6] Ramzi Mahmoudi, Mohamed Akil, and Mohamed Hedi Bedoui. Concurrent computation of topological watershed on shared memory parallel machines. *Parallel Computing*, 69:78 – 97, November 2017.

[7] Jos BTM Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta informaticae*, 41(1, 2):187–228, 2000.

[8] Luc Vincent and Pierre Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(06):583–598, 1991.

[9] Arnold Meijster and Jos BTM Roerdink. A disjoint set algorithm for the watershed transform. In *9th European Signal Processing Conference (EUSIPCO 1998)*, pages 1–4. IEEE, 1998.

[10] Michael H. F. Wilkinson, Hui Gao, Wim H. Hesselink, Jan-Eppo Jonker, and Arnold Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *Ieee transactions on pattern analysis and machine intelligence*, 30(10):1800–1813, October 2008.

Relation: http://www.rug.nl/informatica/organisatie/overorganisatie/iwi Rights: University of Groningen. Research Institute for Mathematics and Computing Science (IWI); IEEE Conference on Computer Vision and Pattern Recognition ; Conference date: 17-06-2007 Through 22-06-2007.

[11] Harry Henderson. *Encyclopedia of computer science and technology*. Infobase Publishing, 2009.

[12] Moore's law - now and in the future. `https://www.intel.com/content/www/us/en/newsroom/opinion/moore-law-now-and-in-the-future.html`. Accessed: 02-03-2022.

[13] The history of a dream: How the ultimate pc has evolved in 15 years. `https://web.archive.org/web/20150418074002/http://www.maximumpc.com:80/article/home/history_dream_how_ultimate_pc_has_evolved_15_years`. Accessed: 02-03-2022.

[14] Timothy G Mattson, Beverly Sanders, and Berna Massingill. *Patterns for parallel programming*. Pearson Education, 2004.

[15] Jean Cousty, Gilles Bertrand, Laurent Najman, and Michel Couprie. Watersheds, minimum spanning forests, and the drop of water principle. 2007.

Preserving the privacy of data in autonomous cars IoT using Intel SGX

Ties Pol, Andrei Badescu

Abstract— The IoT environment is spreading to new fields and it has become more and more dependent on cloud computing services for computation. This created many new functionalities and applications but also has created potential security risks, especially when it comes to private data. For that reason, there is a global need to preserve data privacy in an IoT environment and to find new and innovative ways to keep sensitive data safe. Several methods have discussed privacy preservation, but they do not solve this problem. However, Intel Software Guard Extensions (SGX) technology is a promising candidate in this space that can ensure confidentiality in IoT environments.

This paper discusses some various methods that are used to keep the data of IoT systems private. There are many different methods to ensure data privacy, different types of encryption(e.g., anonymization, noise injection), but this paper focuses on Intel SGX. This popular technology can also be used in combination with blockchain, edge computing, and many other technologies depending on the use case. An overview and some background information about all relevant privacy ensuring methods are provided with their advantages and disadvantages. These will be compared for their security impact, scalability, and feasibility.

After the literature review about Intel SGX and IoT, we provide a system model that introduces security problems in autonomous cars. To solve these problems we introduce a system design that applies an SGX framework to autonomous cars, called Internet of Autonomous Car Things (IoACT). This framework uses Intel SGX in combination with edge computing and blockchain. The system design is based on a framework that is introduced in this paper [7], where it is applied to the Internet of Medical Things (IoMT). Finally, the design is evaluated by using experiments of several related papers that have done a simulation on SGX. This evaluation has shown that using SGX in IoACT gives an acceptable extra computing time. We could unfortunately not give an explicit answer if the system design holds or not. However, we strongly believe that this framework can be applied to autonomous cars.

Index Terms—Intel software guard extensions (SGX), Data privacy, IoT, Blockchain, Edge computing, Internet of Autonomous Cars (IoACT).

1 Introduction

In recent years, the number of Internet of Things (IoT) devices has had a strong increase in many different fields. IoT are physical objects that are equipped with for example sensors, processing ability, software, and other technologies. These objects can be placed in groups of devices and are all connected to the internet, this way they can exchange and communicate with other devices [14]. A thing can be a person with a heart monitor implant, an animal with a chip transponder, a fridge that has built-in sensors to alert the owner that it needs servicing or any other object that can be assigned an Internet Protocol (IP) address and can transfer data over a network. Advances in memory and processing resources and the urge to reduce data transmission latency have led to a rapid rise in the deployment of various Deep Neural Networks (DNNs) on constrained edge devices (e.g., wearable, smartphones, and consumer IoT devices) [16].

Many advancements in other fields such as cloud computing, machine learning, and embedded systems paved the way for IoT devices to become cheaper and more reliable. The main fields where IoT has become widespread in recent years are smart home applications, transportation, smart agriculture, medical, and healthcare applications [12].

The significant growth of IoT devices can involve critical security issues, which can lead to data privacy problems. Therefore, there is a need to preserve these problems. A solution for these privacy issues can be a trusted execution environment on the edge of an IoT environment. This can be realized by using Intel software guard extensions (SGX) technology. In addition, this technology can be combined with edge computing and blockchain too.

Intel SGX is a promising technology, developed by Intel, that could potentially solve these problems in IoT environments by providing a unique application isolation technology. While typical security measures may assist data at rest and in transit, they often fall short of protecting data while it is actively used in memory. Intel SGX helps protect data in use via application isolation technology. This creates a Trusted Execution Environment (TEE) which encapsulates the code in private regions of memory called enclaves which provides an execution space that offers a much higher level of security for trusted applications running on the device compared to classical methods. By protecting selected code and data from modification, developers can partition their application into hardened enclaves or trusted execution modules to help increase application security. The next section will go more in dept about Intel SGX and the combination of the beforementioned technologies. In addition to that background information about SGX, we discuss the vulnerabilities and threats of an IoT environment and which can be solved by using Intel SGX.

However, besides the rise of IoT devices in the above-mentioned fields, there is also the upcoming rise of IoT environments on wheels, called autonomous cars. It can have multiple IoT devices that all come with its implementation software. That means that these IoT privacy problems can happen in autonomous cars as well. So, how can we improve the security of autonomous cars, keep sensitive data away from bad actors and create a safer environment for drivers and pedestrians?

Autonomous cars have been a dream for decades, and engineers have worked towards achieving them, but only recently with advancements made in hardware, computer vision, and machine learning it's starting to become a reality [5]. Still, the average consumer doesn't account that most cars use software that might have vulnerabilities. These cars use processors or communication devices that have flaws as well in the case of autonomous cars, an attack could have tragic consequences.

Therefore, there is a growing concern regarding the security of many devices after security flaws were made public by researchers. They made clear that medical devices such as insulin pumps and pace-

- *Ties Pol is with University of Groningen, E-mail: t.g.j.pol@student.rug.nl.*
- *Andrei Badescu is with University of Groningen, E-mail: a.badescu@student.rug.nl*

makers have their vulnerabilities too. These devices are using the same chips that are used in autonomous vehicles. These advanced devices have the power to improve our lives, but at the moment some of them pose serious threats to the life of their owners, which is why there needs to be much more research conducted into making such devices as secure as possible.

These critical security problems in autonomous cars will be discussed moreover in the system model section. Thereafter, we provide a system design that explains how this problem can be solved by using a framework [7] that combines Intel SGX with edge computing and blockchain. Finally, the design will be evaluated based on related evaluations in other papers to see if the system design holds.

2 BACKGROUND

This section gives a literature review of the different technologies used in this paper. First, Intel SGX will be explained with the technologies it can be combined with. Thereafter, it gives an overview of possible IoT threats and vulnerabilities including which one can be solved using Intel SGX.

2.1 Intel SGX

Intel Software Guard Extensions (SGX) is an expansion of the instruction set provided by Intel that aims to provide integrity and confidentiality guarantees to security-sensitive computing performed on a computer where all the privileged software is potentially malicious. Secure remote computation is the issue related to executing software on a computer that is remotely accessed and owned and maintained by an untrusted party. Therefore, Intel introduced SGX, the latest invention of trusted computing designs, to secure the remote computation difficulty by using trusted hardware. Then the trusted hardware generates a secure container and the computation that has to be executed can be uploaded to this container remotely [16].

SGX makes a reserve in the memory region, called the Processor Reserved Memory (PRM) [16]. In this region, SGX creates an encrypted trusted execution area, what is called an enclave in memory, which offers confidentiality and integrity guarantees to programs running inside them [21]. This enclave is created in a hardware-protected memory, where the data in this enclave is generated by a memory encryption engine (MEE). The CPU protects the PRM from all possible non-enclave access points [4]. Moreover, SGX relies on the security properties the processor's hardware offers, to be more precise, SGX removes all other components from the Trusted Computing Base (TCB). For example, the memory hardware, firmware, and operating system [8]. In particular, SGX ensures that the memory states and processors belonging to an enclave are only accessible to the software that is running inside the enclave and cannot be used by any other enclave and software running in a separate level [21].

Also, Intel SGX provides Intra-attestation, by providing the instruction that helps another enclave to attest to another enclave within the same platform. Remote attestation is also possible where SGX provides a certificate for an enclave to prove to a remote platform that the content loaded in the enclave and the enclave are running in SGX-enabled platform. Trusted functions are also a really useful tool for maintaining security and supporting more complicated solutions.

As shown in figure 1, the enclave is on the right side under the caption "Trusted Code". This SGX visual also makes clear that the privileged system code cannot access the enclave. Besides, the untrusted code can only call the function in the enclave. The enclave will return its outcome but will remain its private data.

Intel SGX can be used in combination with multiple other technologies to ensure privacy in IoT. In the following sections, we explain how Intel SGX can be used with such a technology. The two methods that, in combination with SGX, are discussed below are Blockchain and Edge Computing. These technologies were chosen based on the relevant methods that go in combination with SGX to preserve data privacy for IoT devices. To explain more about the functionalities and privacy-preserving methods these technologies have, we provide more details on each of them below.

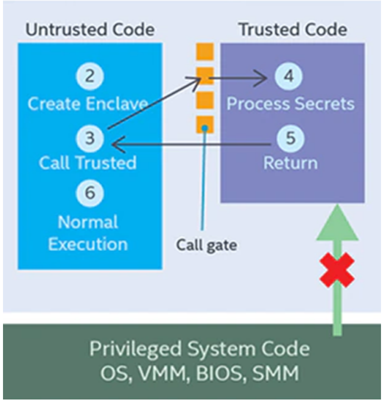


Fig. 1. Core operating principles of Intel SGX

As seen below, the enclave environment cannot be entered through classic function calls the only way is through an instruction that performs several protection checks.

2.2 Edge computing

To ensure confidentiality in IoT, SGX can be introduced into edge computing. This computation on the edge extends the cloud center and performs the computing process closer to the IoT device. This approach can reduce load interaction of the network and avoid a large amount of IoT data from entering the Internet. Therefore, edge computing has attracted a lot of attention in recent years. However, edge computing brings some security issues. Therefore, we can use SGX in edge computing to ensure more security protection.

The edge computing layer connects the IoT environment and cloud services to provide data preprocessing and storage services. In particular, edge computing integrates SGX-based trusted computing services to ensure the integrity and confidentiality of IoT data and protect the privacy of data owners [7].

2.3 Blockchain

Blockchain technology has received a lot of attention in the last years and the adoption has been growing into a myriad of fields such as Healthcare, Music, Real Estate, Banking, Education, Energy, and Insurance. It has the potential to revolutionize and create a lot of possibilities for a lot of fields. A blockchain is a growing list of records, called blocks, that are linked together using cryptography [17]. Blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks. Still, it is not impervious to malicious attacks such as the selfish miner attack [6] or an eclipse attack [8]. Therefore, mass adoption is still not as widespread as it could have been due to the lack of security that could potentially endanger really sensitive information that would be present in healthcare or finance. This is why trusted computing technologies such as SGX could come into play and provide another layer of security on top of the 6 layers that comprise the blockchain.

SGX can serve as an extra layer of security and can be implemented in a few layers of the blockchain. The Application Layer includes the extended techniques and the industrial applications of the blockchain. Implementing the technology on this layer could facilitate transactions of different cryptocurrencies while avoiding possible risks, could make distributed cloud computing a possibility through smart contracts.

2.4 Vulnerabilities

Whereas SGX offers some good advantages and protection in an IoT environment by promising an isolated execution environment, protected from other software running on the same device. Recent papers show that there are some flaws in the SGX instruction set [21] [2]. They show that SGX has a significant limitation of protection against side-channel attacks. Moreover, the paper mentioned that recent works have shown that there are arbitrary data leaks due to transient-execution attacks.

- CacheOut: This attack can retrieve the contents of an enclave. Such a scenario should not be possible as advertised by Intel. However, the side-channel attacks are acknowledged by Intel. They say that the developer is responsible for the side-channel attacks. Intel released several patches, CPU microcode updates, and even new architectures designed to mitigate SGX side-channel leakage via transient execution. However, these fixes were insufficient. For example, the CacheOut attack shows that to obtain SGX data, you first evict the data from the cache and then use TSX Asynchronous Abort (TAA) to recover it. They conclude by warning the SGX-based projects from claiming that they guarantee security even in the presence of information leakage, while there is a clear danger in deploying SGX-based protocols [21].
- Limited memory: Currently, SGX only supports secure memory that is smaller than 12MB. Although SGX offers instructions to allow system software to oversubscribe the secure memory by evicting and loading enclave pages securely, it requires additional operations to protect data privacy, which will result in significant overhead. Thus, the abuse of the enclave will incur the poor performance of the SGX-based schemes [2].
- Availability failures: The SGX-based platform is fully controlled by the platform owner. Thus, it is easy for the platform owner to terminate enclaves. Alternatively, although the secure channel can encrypt the inputs of an enclave, the platform owner can intercept the partial inputs of the enclave, which may impact the security of schemes. For example, the platform owner can transfer the old inputs to the enclave for a favorable consequence, which is called the replay attack. Thus, an SGX-based scheme must tolerate such platform failures [2].
- Side-channel attacks: SGX is implemented on the Intel CPU architecture, and it shares some computing resources (e.g., page table, cache) with normal programs. Therefore, SGX suffers from side-channel attacks, in which the attacker observes the shared resources to obtain the control flow and the data access mode of the enclave program to infer the sensitive information in the enclave. Although most of the side-channel attacks are relatively difficult to be exploited, sufficient profit may drive attackers to attack the SGX-based platforms [2].
- Single-point attacks: If an SGX-based solution relies on the credibility of a single SGX entirely, one compromised SGX will cause the solution to crash. Meanwhile, the SGX-based role in some schemes can maintain massive benefits, which stimulates the attacker to try to impose full control of the SGX, even exploiting the physical means. Therefore, a well-designed multiparty scheme should be able to tolerate the failures of one or more SGX machines [2].

2.5 Threats

As more big players in the hardware industry are producing smart home devices the average consumer started to adopt this technology. Most US households had access to 10 connected devices on average and with the emergence of the 5G networks that will speed up the process, it is becoming extremely difficult to manage an interconnected system of this scale as one vulnerability in one part could potentially jeopardize the whole network. One of the main security flaws is the broad range of connections used in IoT applications, such as Cellular, Wireless LANs, Wired Ethernet, Near Field Communication (NFC), Long Range Networks (LoRA) Bluetooth, each with their utility, standards, protocols, encryptions, etc. That makes it difficult to have a standardized security protocol. This also allows the possibility of exploitation of different vulnerabilities that could be available to attackers due to the devices not running the latest firmware. The IoT security requires IoT devices, protocols, and infrastructures to must have mechanisms to ensure confidentiality, availability, and security (CIA) services. Any IoT application should provide services based on the following requirements on securing sensitive information.

- A) Authenticity: only the authorized nodes can be involved in the communication between any two nodes.
- B) Confidentiality: leakage of sensitive information to any unauthorized user shall be avoided.
- C) Integrity: while transferring the information to IoT devices, data integrity ensures the originality of information that it is not fabricated, rewritten, copied, or replaced by the attacker.
- D) Privacy: the identity of an individual user should be protected by the secure IoT system to maintain privacy.
- E) Availability: an authorized user can use various services provided by IoT network protocols that can support the availability of the services here [1].

The IoT network is structured in a 3-layer architecture, each responsible for the communication with different nodes in the web. There is the perception layer, responsible for gathering data about the environment using the sensors. The Network Layer is in charge of delivering data between hosts in a network and finally, the Application Layers delivers data from users to the network. All of these layers are making use of different types of protocols depending on the particular use-case, each having its vulnerabilities as well.

1. The perception layer is responsible for gathering data about the surrounding environment and the communication here is mostly made between sensors in the IoT network. The most frequent protocols of communication for this layer are NFC, Wi-Fi, RFID, and ZigBee, each being implemented according to the needs, such as range, frequency, power consumption, scalability, etc. [23]

   The perception layer is often subject to attacks such as:

   - Hardware tempering: an attack that physically damages the nodes which are compromised and opens access for the sensitive information
   - Fake node injection: a new fake node is inserted and is designated to modify the original data and would share the wrong information [18]

2. The network layer is divided into two sublayers: the routing layer which handles the transfer of packets from source to destination, and an encapsulation layer that forms the packets. The protocols used in the network layer are 6LowPAN, 6TiSCH, RPL, CORPL, and CARP [23].

   This layer is mostly attacked using methods such as:

   - Man in the middle attacks: where an attacker secretly analyzes the information in between the two parties and tries to get access to the private information or alter the information.
   - Traffic analysis attacks: where an attacker would attack the network based on the analysis and prediction of the network information
   - Sinkhole attacks: in which attacker advertising the fake routing information which intent to attack the network traffic. It will cause other attacks like drops or alter the routing information, and selective forwarding.

3. The application layer is the interface between the IoT device and the network with which it will communicate. It handles data formatting and presentation and serves as the bridge between what the IoT device is doing and the network handoff of the data it produces [24]. The most commonly used protocols are AMQP, MQTT, COAP, XMPP, and DDS.

   The application layer is mostly subject to attacks such as:

   - Phishing attacks: aim to access sensitive information like credit card numbers and authentication login credentials in a banking application by masquerading as trusted parties in the electronic communication.
   - Denial-of-service (DoS) attacks: occur when a node floods the bandwidth or resources with multiple requests for a particular period. In addition to the above specified IoT layered protocols, attacks on the IoT layer and challenges on routing the information in the IoT network have to be addressed.

2.5.1 Implications

From the information provided in the previous sections, we can say that Intel SGX has some advantages and disadvantages in an IoT environment. To start with the advantages. It can be said that the biggest advantage of SGX is preventing software attacks, even when the application, Operating System, and Bios are compromised. Figure **??** shows that privileged system code cannot access the trusted code. That means that the privacy-sensitive data is protected even when the attacker has full control.

An application that uses SGX can execute its code securely inside its safe execution environment. What gives developers a great benefit to control over their application security. However, this brings us to a disadvantage. Intel added a launch control function to SGX. Therefore, each computer's owner must ask for approval from a third party. So, software developers have to enter a business agreement with Intel to be able to author software that takes advantage of SGX protections [20].

3 System Model

As we now have a understanding about which security issues can happen in an IoT environment. We will now highlight the security problems that can occur in autonomous vehicles.

A self-driving car can be seen as a complex IoT environment on wheels. It can have multiple IoT devices that all come with their corresponding software. This means that it can be the case that different third-party sources of software are running on the main computer of the car. Most of the time an autonomous vehicle has one main computer to which all features come together and are connected. As mentioned by this article [13], connected cars are highly complex systems, not quite like but not dissimilar to aircraft. The vast majority of the components found in connected cars are designed and made by companies other than car manufacturers. This represents an integration challenge since history has shown us that highly integrated IT systems are prone to failure, which is something that cannot be tolerated in cars [13]. Therefore, it can be the case that if one software distributor has a data leakage, the other connected software can also be damaged and therefore create a security breach. It therefore can happen that data becomes malicious fast.

The most notable cybersecurity challenges associated with physical components that could potentially apply to such type of vehicle include:

- Sensor jamming, blinding, spoofing, or saturation: Attackers could blind or jam sensors to gain access to autonomous vehicles. This allows malicious actors to feed AV with artificial intelligence models with wrong or incomplete data to undermine model training [11].
- DDoS attacks: hackers could execute distributed denial of service attacks blinding the vehicle to the outside world. DDoS attacks would interfere with autonomous driving leading to stalling or malfunction.
- Manipulation of autonomous vehicle's communication equipment: attackers could hijack communication channels and manipulate sensor readings or wrongly interpret road messages and signs.
- Information disclosure: Autonomous vehicles store large amounts of sensitive personal and AI data. Attackers could cause data breaches on AVs to access sensitive information.

Autonomous cars are equipped with numerous devices to provide entertainment and functionalities never before present in a car. But these can act as attack vectors, as demonstrated in 2019 at the PWN2OWN event in Vancouver, where a Tesla Model 3 was easily hacked through the browser in the ”infotainment” system. To make matters worse, even old technologies like Bluetooth and cellular radio can still pose a security risk as demonstrated by [3]. Therefore, we can confidently state that physical access is a thing of the past when it comes to hijacking cars and that there is a need for a fresh approach when it comes to security for future generations of highly digitized vehicles.

4 System Design

As shown in the previous section, preserving the privacy of data in autonomous cars is extremely important. Therefore, our system design introduces SGX-based edge computing and consortium blockchain technology on the IoT of self-driving cars. This framework is based on the work shown in this paper [7], but instead of Internet of Medical Things(IoMT), we apply it to Internet of Autonomous Car Things (IoACT).

The framework is divided into four layers as shown in figure 2. The layers that are included are described below. It gives a brief description of how the layers are communicating with each other. This communication is illustrated in the system design figure 2.

- Cloud layer: This layer is responsible for the more extensive processing of the IoACT data. This data is already preprocessed by the edge computing layer.
- Edge computing layer: This layer generates the connection between the IoACT environment and the cloud services that will be responsible for the storage and data preprocessing. Again, this framework combines edge computing with SGX-based trusted computing services to ensure the privacy of IoACT data.
- IoACT layer: The IoACT environment acts as the data source in the framework. The data is generated by all the devices that a self-driving car has to offer. As shown by this article [13], companies such as Google, Apple, and BMW forge ahead with driverless cars and apply IoT in transport, it's easy to get carried away by the relentless advances in connected devices.
- Blockchain layer: The CA in the blockchain network is responsible for the authentication of IoACT devices, and the smart contract in the blockchain manages the data generated by the IoACT devices, including metadata and access policies [7]. There is a CA in the consortium blockchain that will give certificates and private keys (PK) to the users including the IoACT devices. So, every IoT device in the car needs a private key and certificate that represents its identity and therefore has its account in the blockchain. The public key makes sure the account is unique.

Intel SGX can resolve the security issue of placing code from different software distributors on the same machine. It can give every part of code its own private and secure enclave. Hence, code parts can run completely on their own.

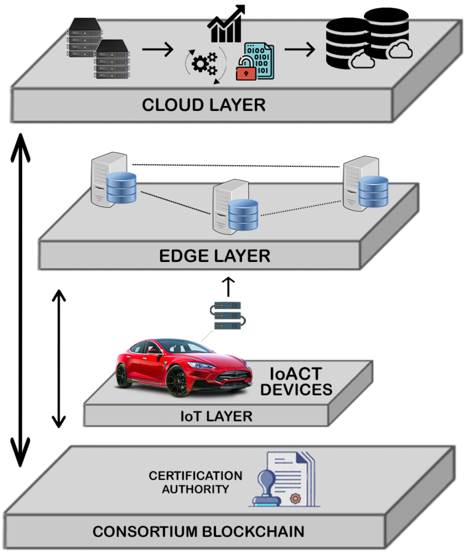


Fig. 2. System design: IoACT environment that includes SGX in combination with edge computing and blockchain technology [7]

To establish a secure communication channel in an enclave we can use a key exchange protocol. We assume that the enclave has obtained a Diffie Hellman shared key with remote attestation [7]. Hence, enclaves can setup secure communication with their IoACT data requester using their shared key. In the key exchange process, the IoACT data will never leave its enclave. If there is an analysis program, it can verify its correctness through SGX. This means that the entire process has met its data security and confidentiality.

This design approach must ensure the privacy of data that is generated by an IoACT. The following section will give an overview of why this system design can hold for autonomous cars.

5 Evaluation

The system design introduced in the previous section will be evaluated in this section. The evaluation will be done by investigating various experiments from related papers to see if the system design can hold. As previously introduced, the IoMT framework [7] can ensure data privacy. As this framework is not IoT-field specific, we think that it can also hold for IoACT environments. However, there is a need to check if the main functionalities of an autonomous car continue working by applying this framework. Therefore, we looked at some experiments that measure performance. The experiments are enumerated below.

1. Performance comparison of data computation [7]

   This experiment shows the difference in data processing performance between edge computing variants. One version uses an SGX-enabled edge node and the other one uses a conventional edge node. The experiment shows the results in figure 3. The x-axis shows the increasing datasets to the right. It can be seen that the data processing time increases when the dataset increases. As the amount of data becomes larger, the processing time of an SGX environment takes exponentially more time. This extra time can be seen in the histogram as 'Extra'.

   So, using Intel SGX in combination with the edge layer has some extra time compared to an ordinary environment. However, it is in a reasonable and acceptable range. We can imagine that some IoT devices in an autonomous car need fast computation in the edge node for safety reasons. If we look at the extra computation time shown in figure 3, we can see that there are only added some milliseconds on small datasets. If we assume that the data that has to be computed is not in large amounts, we can say that it is acceptable to use it in autonomous cars.

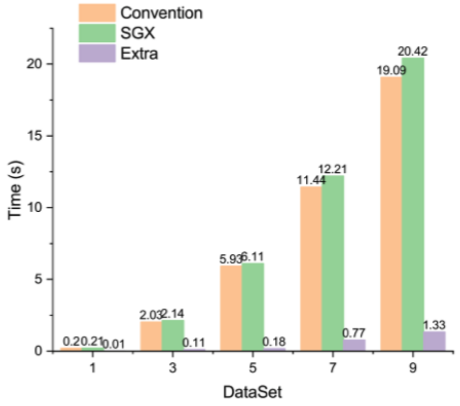


Fig. 3. Performance comparison between convention and SGX-enabled machine [7]

2. Computational overhead time [9]

   This experiment shows a measurement of the computational overhead time based on a framework that is introduced in this paper [9]. It is based on 1000 devices and uses a ruleset of 100, 400, 1000, 5000, and 10000.

   There is a need to discover how the integration of Intel SGX and the cryptographic techniques alter the time overhead of the process. There are considered three cases: no SGX, SGX without encryption, and with SGX.

   Needless to say, the execution time of the experiment with SGX takes longer than the operation when we do not include SGX. Fortunately, the time execution overhead is not that significant.

   In comparison with the previous experiment, the same conclusion applies to this experiment. Because there is not that much extra computation time, we can say that it is reasonable to apply SGX in an IoACT environment.

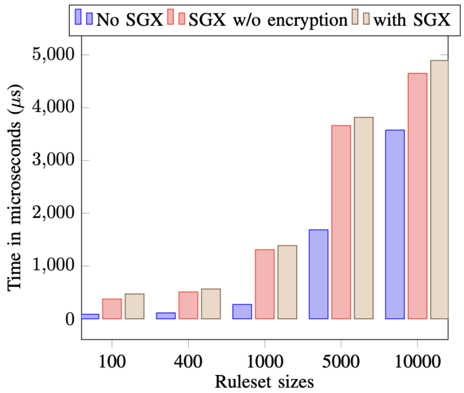


Fig. 4. Average execution time of SGX [9].

To keep in mind is that the IoACT is moving around and is provided by a cellular network, it can have some connectivity fluctuations. This would of course have an impact on the data transfer performance. Therefore, this can result in longer upload time to the cloud layer. However, because edge computing does its computations already on the edge side, it does not have to send that large amount of data to the cloud layer. Hence, there should be almost no connectivity problems with this framework.

Unfortunately, we cannot give an explicit answer if this framework holds for IoACT. The experiments that are retrieved from related papers cannot give a direct answer if the system design can be applied to autonomous cars. Therefore, the gap cannot be closed based on this evaluation. As mentioned in the future work this can be researched by doing an actual simulation on the introduced system design.

6 Discussion

Throughout the evaluation, we found out that doing a simulation on the system design is not feasible in a certain time frame. Applying this technology requires special hardware and access to the source code. Moreover, expertise is needed into the intricacies of the current state of security in on-board computer systems, communication devices and protocols, server-side security, etc., therefore it is unattainable for us to simulate and explicitly present the added benefits of introducing TEE's into the current security technology stack used by car manufacturers.

However, we wanted to prove it in some way, so we did research into other experiments of other related papers. Based on this we have done an evaluation where we use these experiments and have a detailed look at their advantages and disadvantages regarding the IoACT framework.

7 Future work

As mentioned in the discussion a simulation can be done in the future of this research subject. Intel has some documentation about running software guard extensions in simulation mode. The software development kit (SDK) can be downloaded from their website. In addition, there are some Github repositories available that provide code for the simulation, e.g. [19]. This repository provides a list with all papers that use Intel SGX simulation [22].

With this, it is possible to create a simulation in SGX enhanced application. If there is even more time and resources available, it can be checked whether an autonomous car is available to use for research purposes. It would be great to solve the gap that is discussed in the system model. This requires then more time and resources such as dedicated servers and software to create the experimental setup and test the efficacy of the proposed implementation.

8 Conclusion

The main goal of this paper was to contribute to the current IoT state-of-the-art that uses Intel SGX. Moreover, we analyzed the role and efficacy of Intel SGX in combination with two other technologies, blockchain and edge computing. These were investigated in recent research and proposed by scholars as promising applications alongside Intel SGX for increasing the security of IoT systems and explained their security issues, vulnerabilities, technologies, implementations, and results.

After, the literature review it came to our mind that autonomous cars are complex IoT environments on wheels. Thus, these environments also have their threats and vulnerabilities, because they can run different third-party software.

This paper introduces a solution for this gap by applying a framework, that is used in an IoMT environment, to the Internet of Autonomous Car Things. This framework has four layers and combines the before mentioned technologies to provide performance and security enhancements. Edge Computing advancements could be implemented to provide faster communication between the vehicles and cloud networks and provide added security through end-to-end security from the vehicle to the cloud and real-time threat detection at the edge of the network. The blockchain layer can ensure authentication of the IoACT devices by giving certificates and private keys.

We evaluated the system design by investigating different experiments on Intel SGX. These experiments show that the edge layer has some acceptable additional computing time when it uses SGX. However, it is within a reasonable range. We strongly believe that the IoMT approach can be integrated into the security infrastructure of autonomous cars.

References

[1] D. Airehrour, J. Gutierrez, and S. K. Ray. Secure routing for internet of things. *J. Netw. Comput. Appl.*, 66(C):198–213, may 2016.

[2] Z. Bao, Q. Wang, W. Shi, L. Wang, H. Lei, and B. Chen. When blockchain meets sgx: An overview, challenges, and open issues. *IEEE Access*, 8:170404–170420, 2020.

[3] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, page 6, USA, 2011. USENIX Association.

[4] V. Costan and S. Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[5] R. Dave, E. Boone, and K. Roy. Efficient data privacy and security in autonomous cars. *Journal of Computer Sciences and Applications*, 7(1):31–36, 2019.

[6] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[7] Y. Gao, H. Lin, Y. Chen, and Y. Liu. Blockchain and sgx-enabled edge-computing-empowered secure iomt data analysis. *IEEE Internet of Things Journal*, 8(21):15785–15795, 2021.

[8] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on Bitcoin's Peer-to-Peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, Washington, D.C., Aug. 2015. USENIX Association.

[9] M. S. Islam, M. S. Ozdayi, L. Khan, and M. Kantarcioglu. Secure iot data analytics in cloud via intel sgx. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 43–52. IEEE, 2020.

[10] M. S. Islam, M. S. Özdayi, L. Khan, and M. Kantarcioglu. Secure iot data analytics in cloud via intel SGX. *CoRR*, abs/2008.05286, 2020.

[11] P. A. Johnson, B. Tan, and S. Schuckers. Multimodal fusion vulnerability to non-zero effort (spoof) imposters. *2010 IEEE International Workshop on Information Forensics and Security*, pages 1–5, 2010.

[12] W. M. Kang, S. Y. Moon, and J. H. Park. An enhanced security framework for home appliances in smart home. *Human-centric Computing and Information Sciences*, 7(1):1–12, 2017.

[13] R. Kirk. Cars of the future: the internet of things in the automotive industry. *Network Security*, 2015(9):16–18, 2015.

[14] I. Lee and K. Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.

[15] I. Messadi, S. Neumann, N. Weichbrodt, L. Almstedt, M. Mahhouk, and R. Kapitza. Precursor: A fast, client-centric and trusted key-value store using rdma and intel sgx. In *Proceedings of the 22nd International Middleware Conference*, Middleware '21, page 1–13, New York, NY, USA, 2021. Association for Computing Machinery.

[16] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

[17] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[18] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn. Internet of things (iot): Taxonomy of security attacks. In *2016 3rd International Conference on Electronic Design (ICED)*, pages 321–326, 2016.

[19] pedroespindula. Sgx experiments, 2019.

[20] G. Rosinosky, S. Da Silva, S. Ben Mokhtar, D. Négru, L. Réveillère, and E. Rivière. Pprox: Efficient privacy for recommendation-as-a-service. Middleware '21, page 14–26, New York, NY, USA, 2021. Association for Computing Machinery.

[21] S. van Schaik, A. Kwong, and D. Genkin. Sgaxe: How sgx fails in practice. 2020.

[22] vschiavoni. Sgx papers, 2021.

[23] R. Yugha and S. Chithra. A survey on technologies and security protocols: Reference for future generation iot. *Journal of Network and Computer Applications*, 169:102763, 2020.

[24] K. A. M. Zeinab and S. A. A. Elmustafa. Internet of things applications, challenges and related future technologies. *World Scientific News*, 67(2):126–148, 2017.

A survey of algorithms for minimal triangulations, the fill-in problem and the treewidth problem

Jeroen Lammers and Luc Pol

Abstract— A supergraph of a graph is a triangulation if it is chordal. It can be obtained from the original graph by adding edges to this original graph. This triangulation is considered a minimum triangulation if there exist no other triangulations with fewer added edges. The problem of finding such a minimum triangulation is often referred to as the minimum fill-in problem. A relaxation of the fill-in problem is the problem of finding a minimal triangulation. This is a triangulation of which we can not remove any added edges without the supergraph losing its triangulated property. A closely related problem is determining the treewidth of a graph. For this we are interested in finding the triangulation with the smallest maximum clique size. Multiple methods have been developed in order to construct minimal triangulations and solve the minimum fill-in problem and the treewidth problem. In this survey we will summarise and condense the main contributions and evolutions in algorithms solving these problems. This will be done by looking at exact, parameterized and approximative methods. We will be looking at both general algorithms that work on any graph as well as algorithms that are restricted to specific graph classes. With this paper we give an overview of the milestones in algorithm design for minimal triangulations, the fill-in problem and the treewidth problem.

Index Terms—Chordal graphs; Minimal separators; Potential maximal cliques; Minimal triangulation; Minimum fill-in; Treewidth.

1 INTRODUCTION

Several important and widely studied problems on graphs are concerned with computing a chordal triangulation of the original graph. A chordal triangulation of a graph is a supergraph in which each cycle of length greater than three contains a chord. In this paper we will be considering simple, finite, connected and undirected graphs and their triangulations. We will be focusing on three types of triangulations. Starting with a minimal triangulation. We call a triangulation minimal if the set of added edges cannot be reduced further without the supergraph becoming non-triangulated. Second, minimum triangulation, also known as the solution to the (minimal) fill-in problem. A minimum triangulation is a triangulation where the number of additional edges is minimum. Lastly, solutions to the treewidth problem. The treewidth problem is concerned with finding a triangulation with the smallest possible maximal clique size.

Treewidth was introduced by Robertson and Seymour in [36] and plays a major role in graph algorithm design. This is because many NP-hard problems have polynomial time solutions if we restrict ourselves to graphs with small treewidth. Examples of such problems are independent sets, dominating sets, graph colouring, Hamiltonian circuits, network reliability and logic [2, 3]. The solution of the fill-in problem has use cases in sparse matrix multiplication, database management, knowledge based systems and computer vision [4, 16, 29, 37, 40]. However, finding the minimal triangulation is NP-hard [42]. Because of this we, will also focus on minimal triangulations, which are a less restrictive version of minimum triangulations and can be computed in polynomial time. They are especially useful since they are closely related to both minimum triangulations and the solution to the treewidth problem.

The goal of this paper is to provide an overview of the different methods that are currently available for each type of triangulation listed before. This will be accomplished by providing an extensive list of the state of the art algorithms that are available. In the case of the fill-in problem and treewidth problem we will also provide an overview of the techniques that are available. We will provide an overview of algorithms that provide exact solutions and algorithms which provide an approximation for arbitrary graphs. Lastly, we will also consider algorithms that are restricted to specific classes of graphs.

In Section 2 we will provide some preliminaries on graph theory and give a more detailed definition of the previously mentioned triangulations and their corresponding problems. In Section 3 we will provide an overview and corresponding background for algorithms used to compute minimal triangulations. In Section 4 we will give an overview of the evolution of minimum triangulation algorithms. In Section 5 we will provide a similar overview for the treewidth problem. In Section 6 we will give an overview of the class specific methods that were developed, as well as some more recent developments. Lastly, Section 7 provides the conclusion of the paper.

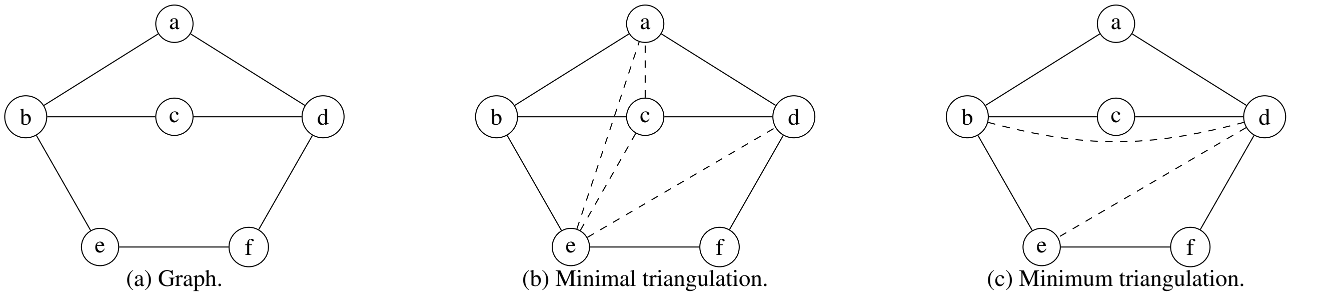
2 PRELIMINARIES

We will be considering simple, finite, connected and undirected graphs denoted by $G(V,E)$. The set of vertices V has n elements and the set of edges E has m elements. Let W be a subset of V. A *subgraph* of G induced by W is denoted by $G(W)$ and is defined as $G(W) = \bar{G}(W, \{uv \in E : u,v \in W\})$. We define a *supergraph* of a graph $G(V,E)$ by the graph $G'(V,E')$ in which $E \subseteq E'$. A *clique* is a vertex set $W \subseteq V$ such that $G(W)$ is fully connected. The process of adding edges to a graph until a graph is fully connected is called *saturating* a graph. The *neighbourhood of a vertex v* in a graph $G(V,E)$ is the set of adjacent vertices given by $N_G(v) = \{u : uv \in E\}$. The *closed neighbourhood of a vertex v* is given by the union of the vertex with its neighbourhood and is denoted by $N_G[v] = N_G(v) \cup \{v\}$. The *neighbourhood of a set* of vertices K is defined as $N_G(K) = \{u : v \in K \wedge u \notin K \wedge uv \in E\}$. Similarly, the *closed neighbourhood of a vertex set* is defined as $N_G[K] = N_G(K) \cup K$.

Let S be a subset of the set V. We denote the *difference* between set S and V as $V \backslash S = \{v \in V : v \notin S\}$. We define the *connected components* of G to be the subgraphs which are connected. A set of vertices S is called a *u,v-separator* if u and v are in different connected components of $G(V \backslash S)$. In Figure 1(a) the set $S = \{b,c,d,f\}$ is an a,e-separator. We call a u,v-separator a *minimal u,v-separator* if there is no subset $S' \subset S$ such that S' is a u,v-separator. The separator $S = \{b,c,d,f\}$ is not minimal. However, it does contain two minimal separators $S_1 = \{b,f\}$ and $S_2 = \{b,d\}$. The set of *minimal separators* of G is defined as the union of all minimal u,v-separators of G. We call two separators S and T *crossing* if S is a u,v-separator with $u,v \in T$. If two separators do not cross, then we call them *parallel*.

We define *path* in a graph $G(V,E)$ to be an ordered sequence of vertices $v_1, v_2, \ldots v_l \in V$ such that $v_i v_{i+1} \in E$ for $i \in \{1,2,\ldots,l-1\}$. We call a path a *cycle* when it starts and ends at the same vertex, i.e. $v_1 = v_l$. A *chord* of a cycle is an edge connecting two non-consecutive vertices of the cycle. A *chordal graph* is a graph in which all cycles of at least four vertices have a chord. We can see in Figure 1(a) that

- *Jeroen Lammers with RUG, E-mail: j.lammers@student.rug.nl.*
- *Luc Pol with RUG, E-mail: l.j.w.pol@student.rug.nl.*



(a) Graph. (b) Minimal triangulation. (c) Minimum triangulation.

Fig. 1. An input graph (a) with minimal triangulations (b) and (c) of which (c) is also a minimum triangulation.

none of the cycles $abefd$, $abcd$ and $bcdfe$ have a chord. Hence, the graph is not chordal. If we were to add an additional edge between ce then the cycle $bcdfe$ would become chordal. However, the newly created cycle $cdfe$ would not have a chord. Hence, by recursively adding chords we can find the chordal graph in Figure 1(b). We call a graph *triangulated* if it is chordal. We call a graph H a *triangulation* of G if H is a supergraph of G and H is triangulated. A triangulation $\bar{G}(V,\bar{E})$ is a *minimal triangulation* of the graph $G(V,E)$ if there does not exist a set of edges E' such that $E \subseteq E' \subset \bar{E}$ and $G'(V,E')$ is a triangulation. If we consider the triangulation in Figure 1(b) we can see that it is a minimal triangulation. Since if we would remove any of the added edges, the graph would have a cycle of length 4 without a chord. Thus the graph in (b) is a minimal triangulation of the graph (a).

Next we will introduce the problem of *(minimum) fill-in* or finding the *minimum triangulation* for which we need to find the triangulation such that the number of added edges is minimal. More precisely, let H be an arbitrary triangulation of G. Then we are interested in finding the triangulations such that the difference between the number of edges in H and G is as small as possible. Note that the minimum triangulation is also a minimal triangulation. However, the inverse is not necessarily true. If we consider the triangulations in Figure 1(b) and (c) then we can clearly see that (c) has fewer added edges. Hence both are a minimal triangulation of the graph (a) but only (c) is a minimum triangulation. Lastly we will introduce the problem of *treewidth*. For this we would like to find the triangulation which has the smallest maximum clique size. In our example Figure 1 we can see that both triangulations have a maximal clique size of 3, which is the smallest possible for this input graph. Note that if we would consider the complete graph with 6 vertices, then that would also be a valid triangulation of the input graph and would have a maximal clique size of 6. It has been shown that the solution to the treewidth problem is also contained in the set of minimal triangulations.

3 Minimal triangulation algorithms

In this section we will provide an overview of minimal triangulation algorithms that have been developed over the years. We will only look at a specific subset of the algorithms. Namely the algorithms that are based on the minimal separators and the potential maximal cliques of a graph. Theorem 1 is at the core of the algorithms presented here. It shows that when you have a set of minimal separators of a graph G, then you can find minimal triangulations in polynomial time. We will present various algorithms and their properties that find those minimal triangulations. If applicable, we will present how these algorithms are linked to each other.

Theorem 1 *(Parra and Scheffler [34])*

1. *Let S be a maximal set of pairwise parallel minimal separators of G. Then the supergraph H of G, generated by the saturation of the minimal separators in S, is a minimal triangulation of G. Moreover, the separators S are the only minimal separators of H.*
2. *Let H be a minimal triangulation of G. Then the minimal separators of H is a maximal set of pairwise parallel minimal separators of G. Moreover H is obtained from saturating S in G.*

3.1 Saturate minimal separators

The first algorithm we will treat is the *Saturate Minimal Separators*-algorithm (*SMS*). It was given by [21], which they deduced from the characterisation of a chordal graph from [17].

Theorem 2 *(Dirac [17]) A graph G is chordal if and only if every minimal separator of G is a clique.*

Let $\mathscr{S}$ be the set of minimal separators of our graph $G(V,E)$. The algorithm is shown in Algorithm 1. It works by enumerating all minimal separators in $\mathscr{S}$ and then checking if that minimal separator is a clique. If it is not, then the algorithm adds the fill edges to the output graph as to saturate the current minimal separator, i.e. make the minimal separator a clique. The resulting graph H is a minimal triangulation.

Algorithm 1 Saturate minimal separators (Heggernes [21])

```
Input: $G(V,E)$ and $\mathscr{S}$
Output: A triangulation $H(V,E')$ of $G(V,E)$.
$H(V,E) := G(V,E)$
for $S \in \mathscr{S}$ do
    if $S$ is not a clique then
        Add fill edges to $H$ to saturate $S$
```

In [21] the connection between minimal separators and minimal triangulation is presented by a combination of multiple results. When a minimal separator S is saturated, all of the minimal separators that cross S disappear. This is because the vertices of S cannot be separated from each other anymore. Hence, the solution of the SMS algorithm gives a triangulation.

The time complexity of the SMS algorithm is $O(n)$, as we only iterate over the set of minimal separators once, while potentially adding edges along the way. However, the SMS algorithm needs that set of minimal separators to work. While the algorithm itself is quite straightforward, the difficulty arises in generating that set of minimal separators. This is an NP-hard problem. In the next section an improved algorithm is provided that does not need the set of minimal separators.

3.2 LB-Triang

The problem with the SMS algorithm (Algorithm 1) is that it is not always straightforward to construct the set of minimal separators. Thus a new algorithm was developed based on the characterisation of a triangulation given by Lekkerkerker and Boland.

Theorem 3 *(Lekkerkerker and Boland [30]) A graph G is a triangulation if and only if for every vertex v in G, each minimal separator $S \subseteq N_G(v)$ of G is a clique.*

The LB-Triang algorithm was originally developed in [5] and summarized in [21]. It works by enumerating all vertices in a pre-defined or random order α. For each vertex the minimal separators in its neighbourhood in the graph H are saturated. Those new edges are added to the graph H.

Algorithm 2 LB-Triang (Berry et al. [5])

```
Input: $G(V,E)$
Output: A minimal triangulation $H(V,E')$ of $G(V,E)$.
$\alpha :=$ arbitrary ordering of $V$
$H(V,E) := G(V,E)$
for $v$ in $\alpha$ do
    Saturate minimal separators contained in $N_H(v)$
    Add these new edges to $H$
```

The algorithm is able to generate all minimal triangulation of the input graph. This can be done by picking an appropriate ordering α in which the vertices will be processed. The correctness of the algorithm follows from theorem 3.

Due to only considering the neighbours at each node, the time to find the minimal separators is reduced dramatically. Note that it has been shown that the search for the connected components and minimal separators can be done in $O(m)$ time in each of the n steps. To achieve this the search of $N_H(v)$ is reduced from the graph H to the graph G. This does have the problem that we can only achieve time bound $O(nm)$ if we do not explicitly compute and store the edges of H, due to the risk of adding the same edge multiple times.

3.3 Vertex incremental minimal triangulation

Next we will be looking at the Vertex Incremental Minimal Triangulation algorithm. This algorithm is based on the necessary and sufficient condition for each edge provided by Theorem 4.

Theorem 4 *(Berry et al. [6]) A graph G is a triangulation if and only if every edge uv in G has the following property: $N_G(u) \cap N_G(v)$ is a minimal u,v-separator in the graph $(V, E \backslash \{uv\})$.*

Other than solving the minimal triangulation problem, it also provides answers to the following questions. Given a triangulated graph $G = (V,E)$, a new vertex $u \notin V$ and a set of edges $D = \{ux : x \in V\}$;

- Is it possible to determine if the new graph $G' = (V \cup \{u\}, E \cup D)$ is a triangulation as well?
- If we are unable to answer the previous question, can we find a set of additional edges $F = \{ux : x \in V$ and $ux \notin D\}$ such that the resulting graph $G' = (V \cup \{u\}, E \cup D \cup F)$ is a triangulation?
- If D generates a new triangulation, can we reduce $D \supset \bar{D}$ such that $G' = (V \cup \{u\}, E \cup \bar{D})$ is a triangulation?

From Theorem 4 it can be shown that an edge uv can be added to a triangulated graph G if and only if for all edges ux such that $x \in S_{u,v}$ are either present, or also added to G. The algorithm will start with an empty set U which will denote the processed vertices. Then we will check if $G(U)$ is a triangulation. If this is not the case than we will add edges based on the previous condition in order to make the graph a triangulation.

Algorithm 3 Vertex incremental minimal triangulation (Berry et al. [6])

```
Input: $G(V,E)$
Output: A minimal triangulation $H(V,E')$ of $G(V,E)$.
$U \leftarrow \{\}$
for $v$ in $V$ do
    $U += v$
    Add edges based on the condition mentioned in the text.
```

Similar to the LB-Triang algorithm, there is no specific order in which the vertices need to be processed. Another similarity is that the vertex incremental minimal triangulation algorithm also has time bound $O(nm)$.

3.4 Fast minimal triangulation

The next method we will be considering was derived from the following characterisation of a minimal triangulation in terms of potential maximal cliques. A potential maximal clique of a graph is a set of vertices which form a maximal clique in some minimal triangulation of the graph.

Theorem 5 *(Bouchitté and Todinca [11]) Let K be a potential maximal clique of $G = (V,E)$, and let $G' = (V,E')$ be a graph obtained from G by saturating K. Let further $C_1, C_2, \dots, C_k$ be the connected components of $G(V \backslash K)$ and $S_i = N_G(C_i)$ for $1 \le i \le k$. $H = (V, E' \cup F)$ is a minimal triangulation of G if and only if $F = \bigcup_{i=1}^{k} F_i$, where F_i is the set of fill edges of a minimal triangulation of $G'(S \cup C_i)$.*

From theorem 5, the fast minimal triangulation algorithm was derived. The algorithm defines a recursive procedure. We start by considering any connected vertex subset $K \subseteq V$. Let us define $A = N_G[K]$ to be the closed neighbour set of K. Then we can use this to compute the connected components $C_1, C_2, \dots, C_k$ of the graph $G(V \backslash A)$. We can saturate each of the neighbour sets of the connected components $N_G(C_i)$ in order to saturate a set of non-crossing minimal separators of $G(V,E)$ and its triangulation. We call the graph with the saturated minimal separators $G'(V,E')$. After this is done we can recursively apply the same procedure to the subgraphs defined by $G'(N_G[C_i])$ and $G'(A)$ which only overlap at the saturated minimal separators. It has been shown in [11] that if A is a potential maximal clique, then the whole of A is automatically saturated when applying the algorithm. This results in fewer sub-problems in the recursion.

Algorithm 4 Fast minimal triangulation (Heggernes et al. [22])

```
Input: $G(V,E)$
Output: A minimal triangulation $H(V,E')$ of $G(V,E)$.
Pick any connected subset $K$ of $V$
$A := N[K]$
Compute the connected components $C_1, \dots, C_k$ of $G(V \backslash A)$
Saturate each set $N(C_i)$ with $1 \le i \le k$
The resulting graph is $G'$
Recursively compute a minimal triangulation of each subgraph $FMT(G'(N[C_i]))$ and $FMT(G'(A))$ independently in the same way
```

It has been shown that the work required at each level in the recursion tree can be bounded by the time required to multiply two $n \times n$ matrices. Hence the efficiency of the algorithm is primarily dependent on the time complexity of matrix multiplication. According to [21] the current best running time of the FMT algorithm is $O(n^{2.376})$.

4 Minimum triangulation algorithms

The fill-in problem finds the triangulation with the minimum number of edges added to make a graph a triangulation. The fill-in problem is also referred to as the minimum triangulation problem. It is applicable in various different fields as mentioned before.

A wide variety of papers have been devoted to this problem, ranging from exact methods to those that work under certain parameterizations of the problem at hand and others that merely give approximations. Some of them we will mention next.

We will start by providing a definition of the minimum fill-in problem.

Definition 1 *(Bouchitté and Todinca [11]) The minimum fill-in of a graph G is the smallest value of $|E_H| - |E_G|$, where the minimum is taken over all triangulations H of G and where $|E_F|$ is the size of the edge set of a graph F.*

4.1 Exact methods

An algorithm for finding the minimum fill-in of a graph was given in [34]. The basis for their algorithm lies in the proof that every inclusion minimal triangulation of a graph corresponds to a maximal clique of its separator graph. Moreover, they again make use of Theorem 1 mentioned in the minimal triangulations section.

In [11] it is shown that if the potential maximal cliques can be enumerated in polynomial time, then the minimum fill-in (and also the treewidth) can be found in polynomial time as well.

Broersma et al. ([12]) provide another algorithm to find a solution to the minimum fill-in problem. They started from a class-specific algorithm on at-free graphs, which they then generalized so the algorithm can be applied on all graphs.

Later, [20] reformulates the findings from [11] in such a way that it allows to compute other graph parameters – on top of fill-in and treewidth – as well. These were the fill-in distance and treelength.

4.2 Parameterized methods

Instead of considering a general solution, one might also look for restricted solutions. Instead of attempting to find the minimum triangulation we could also restrict our problem to checking if a minimum triangulation exists with k or fewer added edges. Algorithms that find such solutions will from here be referred to as parameterized algorithms.

One of the first parameterized algorithms that solves the fill-in problem was given in [23]. They showed that if a graph can be triangulated by adding at most k edges then the fill-in problem can be solved in $O(k^2mn+f(k))$, where n and m represent the number of vertices and edges, respectively. The first term – $O(k^2mn)$ – comes from a three-step partitioning process (exact details can be found in [23]), where the graph G is partitioned in two subsets A and B, such that the size of A is $O(k^3)$ and there are no chordless cycles in G containing vertices in B. The second term – $f(k)$ – results from obtaining a $(k-a)$-triangulation of A for some $a \geq 0$. The function f is defined as $f(k) = k^6 2^{4k}$.

Later, [19] improved on the algorithm shown above. Their algorithm solves the fill-in problem in time $O(2^{O(\sqrt{k}\log k)} + k^2nm)$. It is the first algorithm to solve the fill-in problem in subexponential time. The approach is based on the theory given in [11] regarding the minimal triangulations and potential maximal cliques.

4.3 Approximative methods

An example of an approximation to the fill-in problem is given in [18]. The author presents an algorithm that can find k-triangulations of size $O(k\log k)$ which they claim is within a factor $O(\log k)$ of the optimal solution. This finding was a byproduct of them developing kernelization algorithms for the minimum fill-in problem. As explained in their paper, a kernelization is a systematic research of preprocessing heuristics within the framework of parametrized complexity.

5 Treewidth algorithms

In this section we will provide algorithms that solve the treewidth problem. As explained in [10], the treewidth problem of a graph G means finding a triangulation with smallest maximum clique size. The treewidth itself is then given by the smallest maximum clique size over all triangulations of G, minus 1.

Similar to the previous section about fill-in problem we will provide treewidth algorithms that can be categorized as exact methods, parameterized methods and approximative methods.

5.1 Exact methods

The result from [34] was already mentioned in the section of minimum fill-in above. However, the algorithm is able to find the treewidth of a graph as well.

In 2006 [41] showed that the number of potential maximal cliques for an arbitrary graph G with n vertices is bounded by $O(1.8135^n)$, and that those potential maximal cliques can be listen in time $O(1.8899^n)$. Hence, as a consequence, the treewidth can be computed in the same time bound as well.

5.2 Parameterized methods

A parameterized method for determining the treewidth is given by [7]. It says that for some constant k, a linear time algorithm determines whether the treewidth of a graph G is at most k. If it did find that it was true, a tree decomposition of G with a treewidth of at most k can be found.

5.3 Approximative methods

In 1993, [8] found an algorithm that can compute an approximation of the treewidth problem that is at most $O(\log n)$ times the optimal solution. This works by computing a tree decomposition of the graph. The tree decomposition itself then has a treewidth of at most $O(k\log n)$, where k is the treewidth of the original graph.

In [1] multiple algorithms for finding the treewidth of a graph are presented. One algorithm approximates the solution by a factor $O(k)$, where k is the treewidth. It was the first algorithm able to solve the problem in time independent of the number of vertices in the graph.

6 Graph class specific algorithms

In this section we will provide an overview of algorithms designed for specific classes of graphs. We will mainly be discussing the information depicted in Table 1. This section is meant as an overview and hence we will not be going into the definitions of each of the classes of graphs discussed. In general, these algorithms will have a much better time complexity due to exploiting properties of the more restrictive graph classes. The main reason why this is possible is because most of these graph classes have a polynomially bounded number of minimal separators [10, 11, 28]. We would like to note that the d-trapezoid$^{(1)}$ algorithm also requires additional information as an input. More specifically, it needs a d-trapezoid diagram of which it has been shown that constructing such a diagram is NP-complete [43]. Because of this, the d-trapezoid$^{(1)}$ algorithm is limited in use. In the case of the asteroidal triple-free graphs (AT-free) we can see that the complexity of the algorithm is dependent on the number of minimal separators R of the graph. Hence, this algorithm will only provide polynomial time results in case we have a polynomial bound on the number of minimal separators, similar to the general algorithms. Due to the class of d-trapezoids being a subclass of the AT-free graphs with a polynomial bound number of minimal separators, we can apply the AT-free algorithm in order to obtain a solution to the fill-in problem and the treewidth problem without explicitly computing the d-trapezoid diagram. This results in the d-trapezoid$^{(2)}$ algorithm.

Graph class	$O(mfi())$		$O(tw())$	
AT-free	n^5R+n^3R	[27]	n^5R+n^3R	[27]
Biconvex bipartite	n	[35]	n	[35]
Bipartite permutation	n^5	[38]	-	
Chordal bipartite	n^5	[25]	$p(n)$	[26]
Circle	n^3	[28]	n^3	[24]
Circular-arc	n^3	[28]	n^3	[39]
Cograph	n	[14]	n	[14]
d-trapezoid$^{(1)}$	n^d	[9]	$n * tw(G)^{d-1}$	[9]
d-trapezoid$^{(2)}$	n^{3d+3}	[27]	n^{3d+3}	[27]
Distance hereditary	n	[14]	n	[14]
HHD-free	n^6	[13]	n^6	[13]
Multitolerance	n^5	[33]	-	
Permutation	n	[32]	n	[32]
Weakly triangulated	$n^2\bar{m}^2$	[10]	$n^2\bar{m}^2$	[10]

Table 1. The time complexities of each of the class specific algorithms designed to solve the fill-in problem or the treewidth problem.

Other research focused on more robust methods for solving the fill-in problem and treewidth problem. More specifically, Mancini [31] looks at split graphs, which are a subclass of chordal graphs. Since the problem of fill-in and treewidth are well defined on split graphs, due to them being chordal, the author looks at the possibility of finding

Graph class	$O(mfi())$	$O(tw())$
Split + 1v	NP-hard	n
Split + ke	(2k)! * nm	(2k)! * nm
Split + kv	NP-hard	NP-hard

Table 2. The complexity results of from [31]

algorithms for graphs which are almost split graphs. Their results are summarized in Table 2. More precisely, the class of split graphs with k additional edges called split + ke and the class of graphs which become split graphs after removing k vertices called split + kv. This concept of extending graph classes was originally introduced by Cai in [15] where it was used to find solutions to the graph colouring problem on the extended split and bipartite classes. Mancini shows that the minimum fill-in and treewidth can be computed in $O((2k)! * nm)$. They also show that in general, finding a solution for the fill-in problem and the treewidth problem is NP-hard for split + kv graphs. However, they find that for $k = 1$ the class of split + 1v has an NP-hard solution for the fill-in problem, but a linear time solution for the treewidth problem. This is one of the first results in which there is a significant difference in the complexities in the computation of the fill-in problem and the treewidth problem.

7 Conclusion

In this paper we provided an overview of the different methods for finding minimal triangulation, minimum triangulation and treewidth. We presented how the minimal triangulation methods evolved over time with their respective complexities. Multiple approaches were considered to solve the minimum fill-in problem. We looked at exact methods as well as parameterized methods and approximative methods. Similar approaches were treated related to the treewidth problem. Finally, we provided an overview of class-specific algorithms that where developed over the years. Moreover, we looked at an extension of class specific methods in which one looks at graphs which are 'almost' part of the graph class.

We have seen that we can find a minimal triangulation in case we have a maximal set of parallel minimal separators. It might be interesting to see if it is possible to generate such a set of parallel minimal separators from a smaller set of minimal separators that could act as a generating set for the maximal set of parallel minimal separators. If this is the case we might be able to characterise each minimal triangulation based on these generating minimal separators, which may give a tighter bound on the number of possible minimal triangulations as well as methods that could expedite the process of finding the minimal separators of a graph.

References

[1] E. Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

[2] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[3] S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete applied mathematics*, 23(1):11–24, 1989.

[4] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM (JACM)*, 30(3):479–513, 1983.

[5] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms*, 58(1):33–66, 2006.

[6] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for dynamically maintaining chordal graphs. In *International Symposium on Algorithms and Computation*, pages 47–57. Springer, 2003.

[7] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[8] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[9] H. L. Bodlaender, T. Kloks, D. Kratsch, and H. Müller. Treewidth and minimum fill-in on d-trapezoid graphs. In *Graph Algorithms And Applications I*, pages 139–161. World Scientific, 2002.

[10] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in of weakly triangulated graphs. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 197–206. Springer, 1999.

[11] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

[12] H. Broersma, T. Kloks, D. Kratsch, and H. Müller. A generalization of at-free graphs and a generic algorithm for solving triangulation problems. *Algorithmica*, 32(4):594–610, 2002.

[13] H. J. Broersma, E. Dahlhaus, and T. Kloks. Algorithms for the treewidth and minimum fill-in of hhd-free graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 109–117. Springer, 1997.

[14] H. J. Broersma, E. Dahlhaus, and T. Kloks. A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. *Discrete Applied Mathematics*, 99(1-3):367–400, 2000.

[15] L. Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

[16] F. R. Chung and D. Mumford. Chordal completions of planar graphs. *Journal of Combinatorial Theory, Series B*, 62(1):96–106, 1994.

[17] G. A. Dirac. On rigid circuit graphs. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 25, pages 71–76. Springer, 1961.

[18] F. V. Fomin, G. Philip, and Y. Villanger. Minimum fill-in of sparse graphs: Kernelization and approximation. *Algorithmica*, 71(1):1–20, 2015.

[19] F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM Journal on Computing*, 42(6):2197–2216, 2013.

[20] M. Furuse and K. Yamazaki. A revisit of the scheme for computing treewidth and minimum fill-in. *Theoretical Computer Science*, 531:66–76, 2014.

[21] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[22] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time o (nalpha log n)= o (n2. 376). *SIAM Journal on Discrete Mathematics*, 19(4):900–913, 2005.

[23] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.

[24] T. Kloks. Treewidth of circle graphs. In *International Symposium on Algorithms and Computation*, pages 108–117. Springer, 1993.

[25] T. Kloks et al. *Minimum fill-in for chordal bipartite graphs*, volume 93. Department of Computer Science, Utrecht University, 1993.

[26] T. Kloks and D. Kratsch. Treewidth of chordal bipartite graphs. *Journal of Algorithms*, 19(2):266–281, 1995.

[27] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theoretical Computer Science*, 175(2):309–335, 1997.

[28] T. Kloks, D. Kratsch, and C. Wong. Minimum ll-in on circle and circular-arc graphs. *J. Algorithms*, 28(2):272–289, 1998.

[29] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[30] C. Lekkeikerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

[31] F. Mancini. Minimum fill-in and treewidth of split+ ke and split+ kv graphs. *Discrete applied mathematics*, 158(7):747–754, 2010.

[32] D. Meister. Treewidth and minimum fill-in on permutation graphs in linear time. *Theoretical computer science*, 411(40-42):3685–3700, 2010.

[33] A. Parra. Triangulating multitolerance graphs. *Discrete Applied Mathematics*, 84(1-3):183–197, 1998.

[34] A. Parra and P. Scheffler. How to use the minimal separators of a graph for its chordal triangulation. In *International Colloquium on Automata, Languages, and Programming*, pages 123–134. Springer, 1995.

[35] S.-L. Peng and Y.-C. Yang. On the treewidth and pathwidth of biconvex bipartite graphs. In *International Conference on Theory and Applications of Models of Computation*, pages 244–255. Springer, 2007.

[36] N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

[37] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Elsevier, 1972.

[38] J. Spinrad, A. Brandstädt, and L. Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

[39] R. Sundaram, K. S. Singh, and C. P. Rangan. Treewidth of circular-arc graphs. *SIAM Journal on Discrete Mathematics*, 7(4):647–655, 1994.

[40] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing*, 13(3):566–579, 1984.

[41] Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In *Latin American Symposium on Theoretical Informatics*, pages 800–811. Springer, 2006.

[42] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

[43] M. Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

Welcome to Cloud City:
An Overview of Networking Techniques within the Cloud Data Center

David Visscher, Erwin de Haan

Abstract— The problems faced by large cloud data-centre networks are different than those faced by smaller-scale networks, like those run by single organisations. This review paper will create an overview over modern technologies within the context, in order to bolster understanding by a broader group, by doing a literature review and highlighting important concepts in an understandable manner.

The topics covered by the paper are Topologies, Routing, Virtual Network Functions and Quality of Service. The topology of the network describes the broad shape of the network. Routing concerns figuring out the paths data packets should take between two endpoints, keeping the route as efficient as possible. Virtual Network Functions are virtualized versions of traditionally physical network devices. Virtualizing network functions has the advantage of flexibility and maintainability, just like with virtualization of machines. However, even when a network is optimally laid out, the issue of sharing and dividing resources still remains, which is where quality of service comes in.

For each of these topics, the issues at hand are made plain and possible approaches to the problems are highlighted in this paper. An important pattern that is observed is that, the more is known about the traffic, the more the network can be optimised for that traffic's performance.

Index Terms—Software defined networking, Cloud, Quality of Service, Routing.

1 Introduction

Public cloud infrastructure is quickly becoming the backbone of modern information systems, and with that, scaling challenges grow. Ever more organisations are building their systems around cloud infrastructure, whether public or private, putting pressures on cloud providers to keep expanding and diversifying. With that, networking becomes more and more complex. This review covers recent developments in networking within the cloud data centre, and what unique challenges are faced within that space.

The problems faced by large cloud data-centre networks are different than those faced by smaller-scale networks, like those run by single organisations. The amount of traffic can be enormous, with many different organisations deploying their infrastructures in as little space as possible. The providers of cloud services also cannot know the nature of the traffic beforehand, as anyone can deploy anything at any time. Because of this, the networks in the cloud data centre need to be intelligently designed, dynamic, and adaptable to many different situations.

Software Defined Networking (SDN) is a range of technologies that allow the dynamic reprogramming of networks. This review paper will cover many different techniques that make use of SDN, and the considerations that go into its use.

1.1 Goals

This review paper will create an overview over modern technologies within the context of cloud data centres, in order to bolster understanding by a broader group. By doing so, it is hoped that the reader will come away with new insights that may even be useful in other situations.

We will attempt to create such understanding by doing a literature review and highlighting important concepts in an understandable manner, supported by research questions. We have formulated one main research question, and subdivided that question into four sub-questions. For each section in this paper, we will evaluate how this relates to the questions. The conclusion at the end will then take all this knowledge into account, and formulate answers to the questions.

The questions are formulated as such:

Main Question: Which novel technologies are being developed to underpin the massive scaling of public cloud infrastructure?

Sub-Question 1: How do recent developments handle the field of tension between achieving maximum throughput vs. minimising latency, what are the trade-offs?

Sub-Question 2: How is the allocation of networking resources kept fair and efficient?

Sub-Question 3: How have topologies developed to fit the problems faced by cloud data centres, what are the trade-offs made?

Sub-Question 4: What methodologies have been developed to (dynamically) adapt to changing demands?

1.2 Overview

In this paper we will go through relevant technologies and the papers in which they are proposed, ordered by topic, in order of scale. We will each time, going from section to section, zoom in and look at a more specific level.

The first and most broad topic covered, is that of Topologies (section 2). This is followed by Routing, a look into the behaviour within such a topology (section 3. After this, follows the topic of Virtual Network Functions (VNFs), a topic closely related to SDN (section 4). The fourth and final topic will be that of Quality of Service (section 5).

1.3 Prior Knowledge

While the paper's goal is to explain the topics as clearly as possible, we must assume some familiarity with networking and network virtualisation techniques. This is to keep the paper concise and readable.

2 Topologies

When starting to design a system, we begin with a high-level overview before having to think about the details. Therefore, the first topic we will discuss is topologies, since this is on one of the higher levels of abstraction in a data centre network.

The topology of a network describes the broad shape and architecture of the network. We will discuss a few examples later in this section, but for now it may be good to think about the topology as a schematic map of the network. Since topologies are a high-level overview of the network, it impacts the performance and behaviour of the network.

- *David Visscher, E-Mail: d.j.visscher@student.rug.nl.*
- *Erwin de Haan, E-Mail: e.w.de.haan@student.rug.nl.*

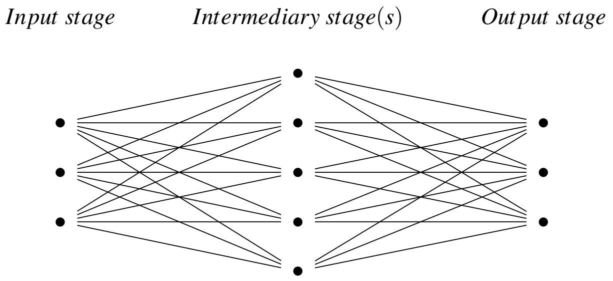


Fig. 1. The Clos topology consists of multiple layers of parallel switches. The number of switches in each layer can be calculated by using formulae from Clos' paper [5]. Each switch in a layer connects to each other switch from the neighboring layers.

2.1 Regions

Before we dive further into the topic of topologies itself, we must first establish some context and terms, one of those being regions. Regions are multiple data centres, in close proximity, operating as though they are a single massive data centre. This has influence on the possible choices and trade-offs in the used topologies, as not every layout would be favourable when clusters of machines are moved away from each other.

The demand for cloud infrastructure keeps growing, but the difficulty of constructing very large data centres – especially in high density areas – is also increasing [6]. Data centres are usually not an attractive sight, but businesses renting services might want the service providers – the data centres – to be close to the businesses themselves, usually in urban areas. Apart from the space considerations, other reasons to move towards regions are that they are more failure-resilient: in the case of a catastrophe; there is less of a single-point-of-failure.

2.2 Optical networking

This rise in demand also comes with a rise in supply in the form of data carriers. Fibre cables have more capacity and are more efficient on longer runs, but there is overhead of translating between the optical and electrical domain[6]. As nearly all machines operate using electrical signals, this poses some overhead problems, but there are some developments to minimise these problems, which we cover in this subsection.

The move to optical networking – with its lower latency over larger distances – changes how networks can be laid out, which in turn enables the creation of regions. The reduction of the latency penalty allows larger distances between core components of the same network, and the enormous potential capacity of fibre cables allow high-throughput interconnects between the regions.

Recent developments mainly concern optical switching or routing methods. As said before, a big part of the overhead – both in time and energy – is translating between optical and electrical signals, which motivates the development of techniques to keep data in the optical domain for longer. The goal here is to minimise the amount of translations needed. There are multiple techniques for this, but there is a trade-off between faster fine-grained switching and cost [6].

2.3 Overview of topologies

Modern-day topologies can be broadly categorised into two families: Clos-based topologies and alternatives to Clos, according to Namyar et al. [11].

The range of Clos-based topologies are founded upon a foundational principle that each endpoint should be able to communicate with all other endpoints in a non-blocking manner. The original architecture was developed by Charles Clos in 1953 as a layout for a telephone network system [5] and consisted of an input stage, an output stage, and one or multiple intermediary stages, all comprised of multiple smaller switches, shown in Figure 1. These stages can still be distinguished in modern-day descendants, as can be seen in Figure 2.

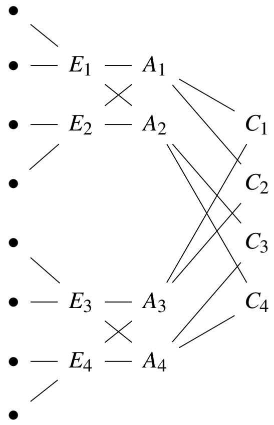


Fig. 2. A small example of the Fat-Tree topology [1]. Note that the aggregation switches are connected to half of the core switches, but each edge switch is in this way connected to all core switches. Under each pair of edge switches is a pod of four network devices.

The depicted older example of a Clos-based topology is the Fat-Tree topology, introduced in 2008 by Al-Fares et al. [1]. In this topology another layer is added to the basic Clos structure, the aggregation layer. It solves a problem in other Clos-based topologies, in which the bandwidth supported by the whole network is lower than the bandwidth supported by the edge switches.

In recent years more economical topologies have been developed [11]. The Jellyfish topology [13] is an example of such a topology. When looking at the connections made between the top-of-rack switches, the network topology is essentially a random graph, with a certain minimum degree for each of the nodes, meaning that each switch connects to a minimum amount of other switches. This topology has been developed to be able to incrementally adjust to changing situations, but also provides 25% more supported servers with the same hardware in contrast to the Fat-Tree topology, and also a lower amount of hops between servers[15].

Other recent examples that utilise graph theory, include the Xpander topology created by Valadarsky et al. [14] focusing on optimal performance of a network, and the FatClique topology created by Zhang et al. which focuses on management complexity, while keeping performance equal [10].

Traditional Clos-based topologies are still widedly used, but other types of layouts can be used in situations where economic concerns are valued higher than management complexity. There is no one-topology-fits-all solution.

2.4 Summary

We have given an overview of the various topologies that exist, the context in which they were developed, and the trade-offs that must be considered. As we have shown, the topic of topologies relates to the first question on throughput vs. latency, and the third question on topological innovations that power cloud data centres.

3 Routing

Routing concerns figuring out the paths that data packets should take between two endpoints. Here we have the goal of keeping the route as efficient as possible. Efficiently routing traffic in cloud data centres is constantly becoming more and more difficult.

Routing becomes more complex when the number of endpoints becomes higher, making this a challenging topic working on the scale of a cloud data centre. On top of this, modern data centres often have to scale to a very large number of servers, virtual machines (VMs) are becoming more and more ephemeral, VMs can be migrated from hypervisor to hypervisor or even across data centres in a region, and entire architectures are redeployed many times per day. This leads to quite a difficult situation for keeping the way we route our traffic optimal.

The concept of routing is tightly related to that of topologies, as the way the network is layed out determines how information can be routed through it. It is intuitive that the routes our information can take is determined by the way the topology is designed.

A brief aside on switching vs. routing (clarifying terms)

Switching relates to forwarding packets connected to a certain switch, while routing concerns the delivery of packets through a whole network. Switches forward frames within a specific OSI layer-2 network. Routers route traffic across networks, using the internet protocol on OSI layer 3. While the lines have certainly blurred with the advent of SDN – because these are often not identifiable physical boxes anymore – it is worthwhile to note the difference.

3.1 Problems specific to the cloud data centre

Ports, firewalls, routers and other virtual network functions migrate across the data centre or region, so the most efficient paths can change quickly, which need fast network response. When virtual machines migrate, placing IP forwarding entries in switches correctly is a main challenge, as a directory of VMs needs to be somehow maintained. The traditional approach to this would be some form of broadcasting or flooding, or to require the querying of a centralised system which maintains the directory. These approaches are relatively not very efficient.

3.2 PARIS

PARIS, proposed by Arora et al. [2], is a proactive system which pre-positions IP forwarding entries in switches. *Proactive* in this case means that the rules for forwarding traffic are already present in switches before traffic first arrives. This reduces the latency for traffic and prevents having to actively learn the correct forwarding last-minute.

PARIS mitigates the issues posed by the traditional approach, no longer requiring broadcasting, flooring or a centralised directory, which expands upon earlier ideas by Changhoon et al. [4]. Forwarding rules are proactively placed in the switch by a logically-centralised controller; this method is chosen to prevent very large forwarding tables.

3.2.1 Topologies

The No-Stretch Topology (as illustrated in Figure 3) variant optimises for low-latency, sacrificing some throughput between endpoints to achieve such latency. In this case, the aggregation switches store reachability information about all the VMs in their pod. Each aggregation switch is then connected to each core switch, this is a requirement and keeps paths short. The topology inside the aggregation and edge layer is not restricted.

The High-Throughput Topology variant (see Figure 4) organises the core switches to form a full mesh, enabling a high throughput at the cost of some extra latency. The aggregation switches no longer have to store all virtual prefix reachability information as in the previous topology. Traffic is intelligently balanced over this mesh, enabling high throughput at the cost of some extra latency.

Considering the two topological options for PARIS, we can formulate a table of pros and cons (see Table 1).

No-Stretch	High Throughput
- Lowest Latency - Switches store all reachability information - Many connections between core and aggregation layers	- Maximum Throughput - Switches store information for only their pod. - Full mesh in the core layer, allowing less connections.

Table 1. Comparison of PARIS topological options.

Multi-tenancy is not yet supported, though the authors have indicated the intention to implement this in the future. This would be an important requirement for many service providers, especially in the cloud data centre.

3.3 In summary..

We have elaborated on the problem of routing, and the shortcomings of the traditional approach with solutions by a novel approach. The topic of routing is connected to our first question, where we need to consider our workload to make the tradeoff between low latency and high throughput. The topic is also very relevant to the fourth question we pose, on how we can dynamically adapt to changing demands.

4 Virtual Network Functions

Now that we have covered routing, we move on to the topic of Virtual Network Function (VNF) placement.

Virtual Network Functions are virtualised versions of traditionally specialised physical network devices. This process of virtualisation is also called Network Function Virtualisation (NFV). Examples of these network functions are firewalls, gateways, proxies, intrusion detection systems, compression, encryption, and load balancing. These functions provide a service for the network infrastructure, just like their physical counterparts normally would.

Virtualising these network functions has the advantage of increasing flexibility and maintainability, just like with virtualisation of machines. This means it combines very well hand-in-hand with Software Defined Networking principles [9]. VNFs are ubiquitous in cloud data centres, being sold as a service to customers as part of an Infrastructure-as-a-Service offering [16].

4.1 Placement problem

The flexibility and freedom provided by this virtualisation does pose a new problem or opportunity: with so many options, where do you place the VNFs? Knowing the best place where to place VNFs is an NP complete problem, depending on the constraints [9].

The placement of these functions plays a role in the performance of the network. There is a strong connection between how we place our VNFs and how data is routed through the network. This is because traffic still must reach the physical device on which the function is being virtualised.

This problem can become more important if there is a series of VNFs that data must be routed through before it gets to the destination. Data centres can have various policies in place, determining that certain chains of VNFs are mandatory in certain situations [7]. Therefore we need some way to mitigate these issues and figure out a way to discover where to place the VNFs.

4.2 Mitigation strategies

An NP complete problem has no clear solution in a reasonable amount of time, so either the amount of input parameters must be limited to narrow the problem space, and/or an optimisation strategy must be used to find a local optimum. Lots of approaches just place a set of

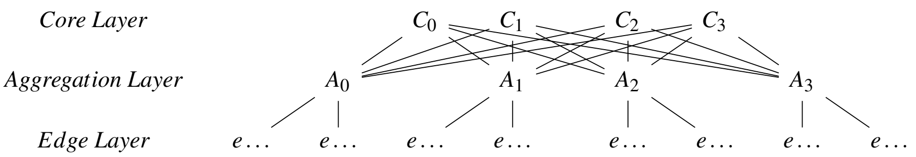


Fig. 3. No-Stretch Topology for PARIS [2]. Note that all aggregation layer switches are connected to all core layer switches. Here, A_0 & A_1 and their edge-nodes form a pod, and A_2 & A_3 and their edge nodes are another pod.

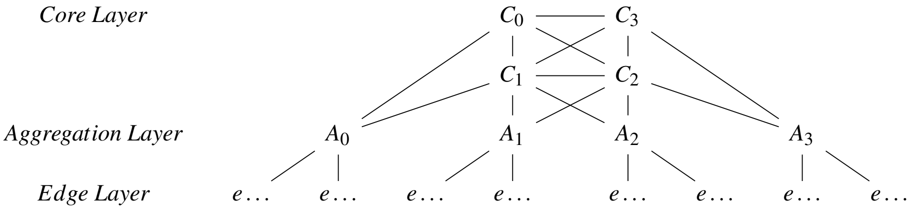


Fig. 4. High-Throughput Topology for PARIS [2]. Note the full mesh in the core layer. Here, pods are configured as in Figure 3.

VNFs and consider those a given when calculating the optimal routing paths [3]. These approaches are called placement-led, but we are interested in other solutions.

As said previously, the routing in a network is related to the placement of VNFs. This means both routing and this placement can be considered at the same time. Luizelli et al. apply a strategy of fixing-and-optimising which considers this whole problem space, and iteratively try to get to an optimal solution [9]. This approach scales to hundreds of VNFs, but is not yet fit for fast adaptation to fluctuation in demand or changing situations.

Another approach is the narrowing of the problem space. Billingsley et al. propose a method for routing-led placement, considering how data should be routed as an immutable constraint; the ideal placement of VNFs is then calculated bearing that in mind [3]. This approach provides better calculation performance in the case of very large networks as compared to placement-led routing.

Keeping even more constraints fixed, quite like placement-led approaches, migrating Virtual Machines is an alternative to migrating the VNFs: keeping paths optimal, not by moving the VNFs, but moving the VMs. Flores et al. apply this approach in a data centre policy aware manner [7]. This approach reduces communication cost of VM pairs, however, considering the placement of VMs simultaneously with the placement of VNFs has not yet been achieved.

4.3 Summary

We have attempted to make plain the issues at hand, and shown possible approaches to optimise the placement of network functions. VNFs and the problem of their placement touch on the sub-questions of Topological Innovations and Adapting Dynamically.

5 Quality of Service

Even when a network is optimally laid out, the issue of sharing and dividing resources still remains. An optimised network is of little use to a tenant when a noisy neighbour can still disrupt their service, just by hogging all of the available resources.

Tenants demand to be able to count on a guarantee of quality aspects of the service, known as a level of service. While the demands can vary, common ones are: minimum downtime, a reserved bandwidth being available, consistency in the service provided. To be able to guarantee this as a service provider, we must make sure that the network is optimised to achieve the level of service for the tenant.

Optimising the flow of traffic, generally, requires us to have knowledge of the nature of that traffic. We can use this knowledge to determine in which ways to best optimise the the flow. This is possible using a number of methods, examples include:

- We can set quotas to limit the bandwidth usage of a certain virtual machine. That way, we can make sure that there is no excessive use. These can be hard quotas: you can't use more than x Mb/s. They can be soft quotas: you can't use more than x Mb/s over a period of t seconds.
- We can set traffic priorities. If we –for example– know a certain type is time-sensitive and another is not, we can let the time-sensitive one go first.

We will now go into several different approaches to quality of service.

5.1 Quality of experience in video streaming

As also explored earlier, insight into the workload can be used to improve the perceived performance of a system.

We can imagine, for example, a Software-as-a-Service offering that provides streaming video. In their paper [12], Shimokawa et al. study such a case, and propose a method for estimating the quality of experience - and not just plain throughput - in the case of streaming video. They then show how such an estimation can be used to optimise performance using Openflow programmable systems.

They propose a system that works in a two-stage approach:

Stage One. A rough estimate is made of the quality of experience (QoE). This is not necessarily optimal, but generating it is fast. That way, the arriving flow can be forwarded onto an appropriate route quickly, avoiding any unnecessary delays.

This can be difficult though, as nothing is known about the flow when it first arrives. The flow is quickly placed along the route with the largest residual bandwidth. Measurements of the flow can then start.

With initial measurements of the flow, an estimation can be made using the bitrate of the video(which is a result of the resolution and frame rate of the video). That way, there is a theoretical number of packets that can be regularly transmitted to achieve optimal flow (disregarding the first few moments, because there's generally some buffering ahead). This info is then used to place the flow along a more sensible route, until the QoE can be more precisely estimated in the second stage.

Stage Two. Based on long-duration measurements (the authors use the most recently recorded eight seconds) of the ongoing flow, it's possible to get a more accurate picture of nature of the traffic. Using those measurements, more precise quality of experience estimations can be made. That information can then be used to optimise in such a way that the QoE for all flows is maximised. These estimations can even be adjusted in real-time. Measurement errors and packet loss does need to be taken into account for this second stage, and the authors' methods allow for estimation, even despite these occurring.

5.2 Dynamic network scheduler

In [8], Hauser et al. propose a solution for the fairer division of bandwidth in situation where multiple virtual machines must share it, and a best-effort approach is not sufficient. An example of what might go wrong with a traditional, best-effort, approach is that one machine might start using a lot of bandwidth, keeping it reserved, another machine then has difficulties getting its traffic through. This is a situation which, of course, can happen all the time in cloud data centres.

Their proposed solution is their dynamic network scheduler, which is enabled by software defined networking: This system identifies the requirements, in terms of bandwidth, for individual virtual machines. It then assigns each virtual machine a bandwidth limitation based on two factors:

- Firstly, It should provide *fairness*. Ensuring the machines do not need to compete for bandwidth.
- Secondly, It should be fully *efficient*, meaning that there should be no left-over bandwidth (the available capacity is used up as much as possible).

The algorithm used to attain this selects two ports where this scheduling is applied: a recipient port and a donor port. It then requires the donor port to lower its quota to make sure that the recipient gets its deserved rate, of course making sure the donor also has sufficient resources to give.

It does this periodically, thus ensuring that no one virtual machine can hold onto all the network resources, starving others.

5.3 In Summary

We have shown that the level of service in a data centre is a concern for the service provider, and that careful consideration of the specific problem is required. However, using knowledge about the traffic involved, a lot can be done to mitigate the issues faced.

The topic of Quality of Service is strongly related to the second subquestion regarding fairness, and the fourth subquestion about dynamic adaptation to changing demands.

6 Discussion

All topics were discussed in a level of depth that should not discourage novice readers to read this paper. We have tried to give a good overview of the topics and a starting point for further research.

The topic of topologies is a bit of an abstract one. Although many more topologies exist or may exist which we are not able to cover, we have highlighted some interesting cases, and tried to give an understanding of the considerations behind them.

We have outlined the problem setting – and considerations one needs to make – when talking about routing on a cloud data centre scale, but many more undertakings have been done attempting the problem, which were are not able to cover.

Virtual Network Functions and their usage have been explained, and some viable strategies to the sketched problem related to them have been highlighted, but research into this topic is still active.

The topic of Quality of Service is optimisation at a detailed level, and highly situational, but we have tried to cover some of these situations to demonstrate the breadth of the issue at hand.

As stated in earlier sections, we have attempted to focus on the considerations that surround the topics covered. We've paid special attention to the state of the art, but this can come at a cost of breadth.

This paper is not an in-depth overview of all possible topics, that would simply not be feasible because of the breadth of information available. Each of the topics that have been covered are interesting and buzzing with new developments, and would therefore merit their own review paper, for which this paper could serve as an entry-point.

7 Conclusion

We will now answer all the questions posed in the introduction to get a final conclusion to the main question, and show our contributions for each topic.

Sub-Question 1: How do recent developments handle the field of tension between achieving maximum throughput vs. minimising latency, what are the trade-offs?

We have discussed topologies, and the trade-offs between latency vs. throughput, in section 2. We have also covered this question within the context of routing, looking at the decisions to be made there, in section 3.

Taking these sections into account we can safely conclude the answer: It depends on the type of traffic you are dealing with. However, in a public cloud data centre we cannot usually make such assumptions about the traffic type since this is at the behest of the tenants.

Sub-Question 2: How is the allocation of networking resources kept fair and efficient?

We've seen, in section 5, that there are many different approaches to dividing network resources fairly.

The more specific information one has about the nature of the traffic, the fairer and more efficient the allocation can be. If not much is known beforehand, then the best approach is to divide as evenly as possible in a best-effort approach.

Sub-Question 3: How have topologies developed to fit the problems faced by cloud data centres, what are the trade-offs made?

This question has – of course – been extensively covered in section 2, where topologies of networks are discussed. We have also seen this topic come up in section 4, where the placement of VNFs is concerned.

In the recent years, there is a move towards regions and optical networking techniques, for which the topologies have to be adapted. There have been multiple possible approaches proposed, each choosing different priorities.

Sub-Question 4: What methodologies have been developed to (dynamically) adapt to changing demands?

The methodologies covered in this paper show the ability to leverage the advantages of SDN in order to dynamically adapt to changing circumstances, we've seen this particularly in section 4. The challenge here has been to keep the layout of the components as efficient as possible, while constantly changing. Virtual Network Functions can be re-placed or scaled in order to better suit specific traffic requirements. Traffic quotas can be dynamically adapted to ensure all parties get a fair share, while maximally utilising available resources, as described in section 5.

Main Question: Which novel technologies are being developed to underpin the massive scaling of public cloud infrastructure?

We highlighted interesting and novel technologies in this paper that have been developed and allow new levels of optimisation in the cloud data-centre. These new techniques can empower the cloud data centre to be utilised even more optimally, supporting cloud providers as they face scaling challenges.

An important pattern we have seen is that, the more is known about the traffic, the more the network can be optimised for that traffic's performance.

Acknowledgements

Many thanks to the reviewers and the expert reviewer for their comments, and everybody involved in the organisation of the course and the colloquium.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication - SIGCOMM '08*, page 63, New York, New York, USA, 2008. ACM Press.

[2] D. Arora, T. Benson, and J. Rexford. ProActive routing in scalable data centers with PARIS. In *DCC 2014 - Proceedings of the ACM SIGCOMM 2014 Workshop on Distributed Cloud Computing*, pages 5–10. Association for Computing Machinery, 2014.

[3] J. Billingsley, K. Li, W. Miao, G. Min, and N. Georgalas. Routing-Led Placement of VNFs in Arbitrary Networks. 1 2020.

[4] K. Changhoon, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecturefor Large Enterprises. page 439, Seattle, Washington, USA, 8 2008. Association for Computing Machinery.

[5] C. Clos. A Study of Non-Blocking Switching Networks. Technical report, 1953.

[6] V. Dukic, G. Khanna, C. Gkantsidis, T. Karagiannis, F. Parmigiani, A. Singla, M. Filer, J. L. Cox, A. Ptasznik, N. Harland, W. Saunders, and C. Belady. Beyond the mega-data center: Networking multi-data center regions. In *SIGCOMM 2020 - Proceedings of the 2020 Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 765–781. Association for Computing Machinery, 7 2020.

[7] H. Flores, V. Tran, and B. Tang. PAM & PAL: Policy-Aware Virtual Machine Migration and Placement in Dynamic Cloud Data Centers. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2549–2558. IEEE, 7 2020.

[8] C. B. Hauser and S. R. Palanivel. Dynamic network scheduler for cloud data centres with SDN. In *UCC 2017 - Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 29–38. Association for Computing Machinery, Inc, 12 2017.

[9] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary. A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining. *Computer Communications*, 102:67–77, 4 2017.

[10] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. Understanding Lifecycle Management Complexity of Datacenter Topologies. In *NSDI'19: Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, pages 235–254, 2019.

[11] P. Namyar, S. Supittayapornpong, M. Zhang, M. Yu, and R. Govindan. A throughput-centric view of the performance of datacenter topologies. In *SIGCOMM 2021 - Proceedings of the ACM SIGCOMM 2021 Conference*, pages 349–369. Association for Computing Machinery, Inc, 8 2021.

[12] S. Shimokawa, Y. Taenaka, K. Tsukamoto, and M. Lee. SDN Based in-Network Two-Staged Video QoE Estimation with Measurement Error Correction for Edge Network. *IEEE Access*, 9:39733–39745, 2021.

[13] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking Data Centers Randomly. Technical report, 2012.

[14] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira. Xpander: Towards optimal-performance datacenters. In *CoNEXT 2016 - Proceedings of the 12th International Conference on Emerging Networking EXperiments and Technologies*, pages 205–219. Association for Computing Machinery, Inc, 12 2016.

[15] W. Xia, P. Zhao, Y. Wen, and H. Xie. A Survey on Data Center Networking (DCN): Infrastructure and Operations. *IEEE Communications Surveys & Tutorials*, 19(1):640–656, 2017.

[16] Z. Xu, H. Ren, W. Liang, Q. Xia, W. Zhou, G. Wu, and P. Zhou. Near Optimal and Dynamic Mechanisms Towards a Stable NFV Market in Multi-Tier Cloud Networks. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10. IEEE, 5 2021.

Deep Contrastive Learning For Object Detection– A Review

Rohit Yadav and Chrysoula-Maria Nampouri

Abstract—Over the past few years, contrastive self-supervised learning has shown remarkable results in many vision tasks including object detection in-the-wild. The establishment of many pre-trained models and datasets has encouraged researchers to modify and fine-tune existing methods and propose novel solutions on diverse downstream tasks without relying on explicit external annotation. In fact, learning efficient visual representations from unlabeled data is a topic of continual research in computer vision, considering the cost of annotation and the sensitivity of models to its quality. Till now, most architectures utilize convolutional neural networks at the patch level of images even for pixel-level labeling problems. However, there is a recent trend of developing self-supervised learning methods that directly work at the level of pixels and incorporate local features representations. This has been shown to bridge the gap between pre-training and pixel-level prediction, achieving state-of-the-art results on target dense prediction tasks. Inspired by recent advances in local feature contrastive learning, we frame our investigation to some of the most prominent deep self-supervised models available today for object detection. Specifically, we will focus on architectures that are pre-trained on benchmark datasets and optimized by a contrastive loss function. The aim is to provide a literature review of the current state-of-the-art, to compare and contrast the performance of models specifically designed for dense prediction and to assess their suitability according to their strengths and weaknesses. During our analysis, we will quantitatively and qualitatively consider methods in terms of accuracy, time efficiency, and complexity.

Index Terms— Object detection, self-supervised learning, dense contrastive learning, contrastive loss, global features representation, local features representation.

1 Introduction

Object detection has been a key area of research for decades due to its potential applications in various fields including face recognition, activity recognition, object tracking, and video object co-segmentation. The straightforward solution relies on fully-supervised learning on labeled target dense datasets. However, annotating images is an immensely time-consuming and expensive process, and in some domains, is still even infeasible due to the absence of relative information. To circumvent the need for explicit supervision, self-supervised approach has aroused extensive attention from researchers in recent years. Specifically, unsupervised pre-training on ImageNet [3] is the dominant choice, where models are pre-trained to acquire global feature representations that can be then transferred to downstream tasks. Nonetheless, the gap between image classification pre-training and target dense prediction tasks (e.g., object detection) is substantial. Contrastive learning showcase a promising alternative by learning proper visual representations directly at the level of pixels (or local features).

This study aims to provide a review analysis of the deep contrastive learning techniques for object detection. Since the field is still emerging, we focus on three out of the most prominent methods up to now, namely DenseCL [22], DetCo [24], and deep contrast learning (DCL) [12]. The reason for choosing these specific algorithms is that DetCo and DenseCL rely on the same experimental setup, allowing fair comparisons, while all three operate in the pixel space and achieve state-of-the-art results at certain detection objectives. Our goal is to provide answers to the following questions: (1) Do these dense methods outperform the earlier global contrastive approaches in object detection? (2) Do they perform better than their supervised counterparts? (3) Are they adaptable to other end-tasks?

During our analysis, the proposed methods are considered quantitatively, by providing a clear view of the accuracy improvements and efficiency, as well as qualitatively with respect to visualization results and model's complexity. In order to ensure consistency of our study's contribution, the results obtained from these methods are judged taking into account their experimental setup and baseline counterparts.

The rest of this paper is structured as follows: Section 2 presents the related work on contrastive learning. Section 3 introduces the common contrastive learning pipeline for global representation learning. Section 4 provides an overview of the state-of-the-art techniques for dense prediction tasks. This is followed by their visual results in Section 5 and a detailed comparative analysis in Section 6. Finally, Section 7 summarizes the key aspects of our work and concludes the paper.

2 Related Work

In the last decade, self-supervised learning has witnessed remarkable progress in learning representations from unlabeled data in two directions: generative modeling and contrastive learning. The former aim at learning representations through modeling density and is typically relying on either auto-encoding of images or adversarial learning [6].

In contrast, contrastive learning is a discriminative approach that produces impressive transferable visual representations by learning to be invariant to different augmentation compositions. Specifically, the main idea is to pull representations of different views of the same image (positive pairs) close and push representations of views of different images (negative pairs) apart. The breakthrough approach is SimCLR [1] which is trained to maximize the agreement between differently augmented views of the same image while introducing a nonlinear projection of representations to the space where contrastive loss is applied. One severe drawback of this method is that it requires large batch sizes to guarantee a sufficient number of negative pairs during training. Following that, He et al. (2020) and Chen et al. (2020) proposed Momentum Contrast (MoCo) [7] and MoCo-v2 [2] respectively that restrain computational requirements by using a large memory bank of samples for computing the contrastive loss.

Earlier methods rely on many sorts of pretext tasks to learn visual representations and eventually transfer them to downstream tasks [19, 25, 17, 10, 5]. Nonetheless, most of them are specifically designed for learning global features for images, thus neglecting pixel-wise tasks like object detection. Recent studies address this issue and propose methods that directly work at the level of pixels achieving state-of-the-art results in dense prediction tasks [22, 24].

Even further, saliency detection has recently attracted great interest due to its powerful ability to highlight the most noticeable object regions in an image. The multi-context deep learning framework (MC) [26] is one of the first methods that incorporate both global and local features. Inspired by that, Li et al. (2015) proposed MDF [11] that adopts a refinement model for the better spatial coherence of the

- *Rohit Yadav is with University of Groningen, E-mail: r.yadav.2@student.rug.nl.*
- *Chrysoula-Maria Nampouri is with University of Groningen, E-mail: c.m.nampouri@student.rug.nl .*

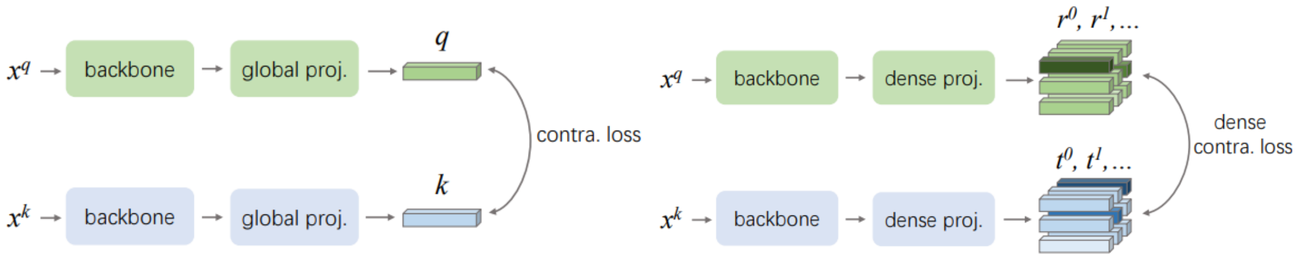


Fig. 1. DenseCL architecture. (Left) The global contrastive loss is computed between the global feature vectors outputted by the global projection head. (Right) The dense contrastive loss is computed between the dense feature vectors outputted by the dense projection head, at the level of local features. Two augmented views x^q and x^k can be encoded by the same encoder or different ones. Image taken from [22].

extracted multi-scale features. The same authors in 2016 presented an improved version of MDF called DCL [12] that discovers high-level visual contrast in an end-to-end mode.

3 Contrastive Learning Pipeline

The state-of-the-art (SOTA) contrastive learning pipeline relies on learning global representations of features and consists of four major components: data augmentation, backbone network, projection head and contrastive loss.

Pre-training pipeline. Given an unlabeled set of images, different *views* for each image are generated by applying data augmentations techniques. Two augmented views of the same image are called positive pairs, while views of different images are negative pairs. Each pair of views is initially fed into a backbone network, e.g., ResNet [9] or any other convolutional neural network, to create global vector representations for the whole views. These representations are then forwarded to a set of dense layers called *projection head* and transformed into a non-linear space. The final goal is to generate similar representations for the positive pairs by optimizing a pairwise contrastive (dis)similarity loss at the level of global features. The overall pipeline from images to global representations is illustrated in Figure 1 (Left).

Global contrastive loss. Following the principle of MoCo-v2, contrastive learning can be considered as a *matching queries to keys* process. To be more precise, for each encoded query q, a set of key representations $\{k_0, k_1, \ldots\}$ are encoded by a (different) moving average network (momentum encoder) that maintains their consistency on every training update. Among these keys, there is one positive key k_+ that matches query q and encodes different views of the same image. The rest keys are treated as negative and encode views of different images. Then, InfoNCE [18] contrastive loss, with a temperature hyper-parameter τ as in [23], is employed to pull q close to k_+ while pushing it away from the rest negative keys k_-:

$$\mathscr{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{k_-} \exp(q \cdot k_-/\tau)} \tag{1}$$

Fine-tuning pipeline. Once the pre-training process is complete, the encoded feature representations are used as input to the so-called *end-task network*. End-task networks take the input embedding and convert it into the desired target task output. In this study, we focus on object detection, and thus, detector networks [20, 8, 14] are assumed.

4 Methods

The paradigm for learning global features has been discussed in Section 3. This section presents the three cutting-edge architectures tailored for dense prediction tasks: DenseCL, DetCo and DCL.

4.1 DenseCL Framework

DenseCL is a novel self-supervised contrastive learning framework customized for dense prediction tasks [22]. It follows the general pipeline presented in Section 3, with the core differences lying in the encoder and loss function.

4.1.1 DenseCL Encoder Pipeline

Features extracted by backbone networks are forwarded to a sophisticated projection head scheme consisting of two complementary subheads; one global projection head and one dense projection head. The global projection head can be instantiated as any of the existing projection heads that generate global feature representations of views.

To generate dense feature vectors as well, Wang et al. (2021) proposed an additional projection head that takes the same input with the global but outputs a dense format instead —Figure 1 (Right). Specifically, the global pooling layer is removed from the backbone network, while the multi-layer perceptron is replaced by the identical 1 × 1 convolution layers with ReLu activation function in between. The number of parameters in both projection heads is the same.

The second main contribution of this method concerns the learning process. In fact, the backbone and the two parallel projection heads are end-to-end trained by optimizing a pairwise contrastive (dis)similarity loss not only at the level of global features but also of the local features.

4.1.2 Dense Contrastive Learning

Dense contrastive learning extends the conventional InfoNCE loss described in Section 3 by incorporating the loss of both global and local representations. Global contrastive loss term derives from the output of global projection head and is given by Equation 1.

Regarding the local contrastive learning term, a set of encoded keys $\{t_0, t_1, \ldots\}$ is defined for each encoded query r. However, in this case, each query represents a local part of the view generated by the dense projection head. Then, each negative key t_- is the pooled feature vector of a view from a different image. The positive key t_+ is another view of the same image and is assigned by taking into account its visual correspondence across views (described in Section 4.1.3). Eventually, the dense contrastive loss is derived as follows:

$$\mathscr{L}_r = \frac{1}{S^2} \sum_s -\log \frac{\exp(r^s \cdot t^s_+/\tau)}{\exp(r^s \cdot t^s_+) + \sum_{t^s_-} \exp(r^s \cdot t^s_-/\tau)}, \tag{2}$$

where r^s denotes the s^{th} out of the S^2 feature vectors generated by the dense projection head and corresponds to a local part of a view. Overall, the total loss of DenseCL is given by:

$$\mathscr{L} = (1-\lambda)\mathscr{L}_q + \lambda \mathscr{L}_r, \tag{3}$$

where λ is a hyper-parameter tuned to balance the contribution of global and local loss. Note that $\lambda = 0$ corresponds to MoCo-v2.

4.1.3 Dense Correspondence across Views

To find the positive key t^+ of an encoded query q, the correspondence across views is extracted. Specifically, a matching is built between the dense feature vectors of two views, i.e., Θ_1 and Θ_2, using the corresponding backbone feature maps F_1 and F_2; From F_1 and F_2, the cosine similarity matrix is calculated. Then, for all the feature vectors of Θ_1, the correspondence with Θ_2 is obtained by applying an argmax operator that indicates the maximum similarity.

4.2 DetCo Framework

Up to now, most self-supervised contrastive learning algorithms were end-task oriented. In contrast, DetCo [24] is a more universal contrastive learning framework that demonstrates superior performance on dense prediction tasks while maintaining competitive image classification transfer accuracy. This is achieved by designing (1) multi-level supervision that keeps features at multiple stages of the backbone network discriminative, (2) global and local contrastive learning to mutually enhance global and local representations. The core architecture of DetCo is mainly based on MoCo-v2 meaning that the different augmented views of an input image are trained by different encoders, namely encoder$_q$ and its momentum-updated encoder$_k$. The overall pipeline is depicted in Figure 2.

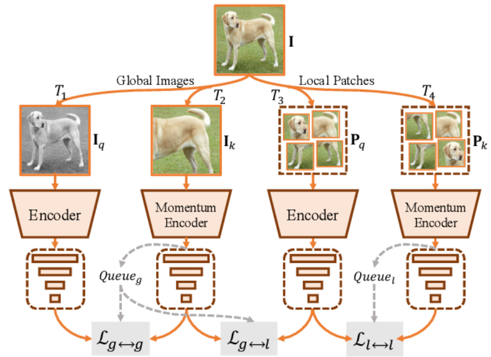


Fig. 2. DetCo architecture. Besides the global views, DetCo adds two additional local-patch sets for input, building contrastive loss across both the global and local views. Representation learning is ultimately deployed through multi-level supervision. Image taken from [24].

4.2.1 Multi-level Supervision

The first main modification of this method relies on the way features are extracted; while most models reach the final level of a backbone network to learn a proper visual representation, DetCo yields features from four intermediate stages.

To be more precise, given an input image $I \in R^{H\times W\times 3}$ and two augmented views I_q and I_k of this image, a standard backbone network is fed with each of the views and extracts features $\{f_2, f_3, f_4, f_5\}$, where f_i is the feature vector from the i^{th} stage. The obtained multi-level backbone features are then forwarded to four projection heads respectively, from which four global representations $\{q/k_2^g, q/k_3^g, q/k_4^g, q/k_5^g\} = \text{encoder}_{q/k}(I_{q/k})$ derive. Then, the contrastive loss function of DetCo extends the conventional InfoNCE to multi-level contrastive losses for multi-stage features, formulated as:

$$\mathscr{L}_{g\leftrightarrow g}(I_q, I_k) = \sum_{i=1}^{4} w_i \cdot \mathscr{L}^i_{g\leftrightarrow g}, \tag{4}$$

where w is the loss weight and $\mathscr{L}^i_{g\leftrightarrow g}$ indicates the global↔global contrastive loss at the i^{th} stage as derived from Equation 1.

4.2.2 Global and Local Contrastive Learning

The idea of employing multi-level feature representations for global features, as described in Section 4.2.1, can also be adopted in local features representation learning.

Specifically, given an input image $I \in R^{H\times W\times 3}$, nine local patches $\{p_1, p_2, ..., p_9\}$ are generated from each of the two different views P_q and P_k and pass through a backbone network. At each stage, nine feature vectors $F_p = \{f_{p1}, f_{p2}, ..., f_{p9}\}$ are extracted and concatenated to be fed into a projection head. The output is the final representation q/k^l at a certain stage. Then, the local contrastive loss is evaluated in terms of both global↔local and local↔local losses as follows:

$$\mathscr{L}_{g\leftrightarrow l}(I_k, P_q) = -\log \frac{\exp(q^l \cdot k^g_+/\tau)}{\sum_{i=0}^{K} \exp(q^l \cdot k^g_i/\tau)} \tag{5}$$

$$\mathscr{L}_{l\leftrightarrow l}(P_q, P_k) = -\log \frac{\exp(q^l \cdot k^l_+/\tau)}{\sum_{i=0}^{K} \exp(q^l \cdot k^l_i/\tau)} \tag{6}$$

Eventually, the complete loss function of DetCo derives from:

$$\mathscr{L}(I_q, I_k, P_q, P_k) = \sum_{i=1}^{4} w_i \cdot (\mathscr{L}^i_{g\leftrightarrow g} + \mathscr{L}^i_{l\leftrightarrow l} + \mathscr{L}^i_{g\leftrightarrow l}) \tag{7}$$

4.3 Deep Contrast Learning

DCL is an end-to-end deep network tailored for salient object detection [12]. It consists of two parallel streams, which are a multi-scale fully convolution network (MS-FCN) and a segment-level saliency stream, as shown in Figure 3.

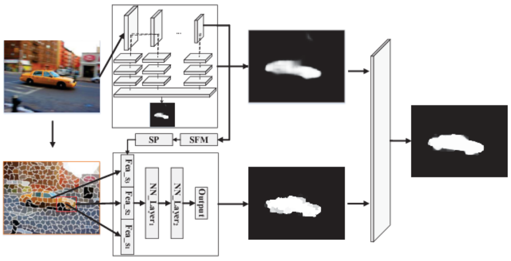


Fig. 3. Streams of the DCL network. Image taken from [12].

4.3.1 Multi-Scale Fully Convolution Network

The fully-convolution stream aims to create a mapping from an input image to a pixel-level saliency map. Li et al. (2016) employ the VGG16 network [21] as the pre-trained model, since after certain modifications it proves to be (1) deep enough to produce multi-level features, and (2) able to infer semantic properties of the objects as well as capture subtle visual contrast.

To be more precise, the first main modification relies on replacing all the fully-connected layers of VGG16 into 1 × 1 convolution ones. However, this yields a very sparse prediction map with a 32-pixel stride. To overcome this issue, sub-sampling is skipped in the last two pooling layers, thus maintaining an 8-pixel stride.

The second contribution concerns the generation of multi-scale saliency maps. Specifically, after each pooling layer (four), the corresponding extracted feature map is forwarded to three external convolution layers. The four feature maps derived from these layers are then stacked along with the last true output map of VGG16 (5 channels) and fed into a final convolution layer with 1 × 1 kernel and single channel. The generated output is called resized saliency map S_1. The entire architecture of the modified VGG16 is presented in Figure 4.

4.3.2 Segment-Level Saliency

In most cases, salient objects have a rather irregular shape. The saliency map based only on MS-FCN possesses discontinuities and thus, may fail to cover their fine boundaries. To overcome this issue, DCL employs an additional segment-wise pooling stream that models visual contrast between regions. In this case, the input image is initially divided into segments. For each segment, the output generated from the true convolution layer (Conv5_3) of MS-FCN is masked to either ’1’ or ’0’ labels depending on whether the pixels are within the segment or not. Since segments of Conv5_3 have variable size, spatial pooling is then applied within a number of sub-cells of the general bounding box of each segment. Finally, to obtain segment-level visual contrast, three spatially accumulated feature vectors are deployed, i.e., boundary box of current segment, boundary box of neighbouring segment, and the saliency map from Conv5_3. These three feature vectors

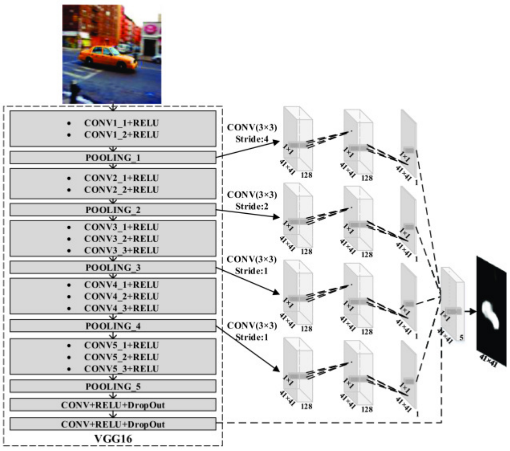


Fig. 4. The architecture of MS-FCN. Image taken from [12].

are further fed to three layers, as shown in Figure 3, with the first two being fully connected and the last being the output layer that produces the saliency map S_2.

4.3.3 Spatial Coherence

Given the two output saliency maps S_1 and S_2 from MS-FCN and segment-level respectively, one option is to merely combine them which results in DCL. However, in this way, we miss to check if the saliency score for the current segment is consistent with its neighbors or not. Since both streams S_1 and S_2 give saliency scores without considering spatial coherence, this needs to be refined for better contour localization. Thus, a fully connected conditional random field (CRF) is applied to minimize the energy function $E(L)$:

$$E(L) = -\sum_{i} logP(l_i) + \sum_{i,j} \theta_{ij}(l_i, l_j), \tag{8}$$

where L represents a binary label (salient – non-salient), $P(l_i)$ is the probability of a pixel x_i having the label l_i, and θ_{ij} is the pairwise potential. The advantage of CRF output (DCL$^+$) is that it not only produces smooth results with pairwise accuracy but also preserves salient object contours, which up to now were a trade-off configuration.

5 Results

Deep self-supervised contrastive learning algorithms for object detection have been presented in Section 4. Table 1 gives the summary of this review study.

DenseCL framework is used mainly for dense prediction tasks. The pre-training experiments were conducted on two large-scale datasets, designed for different end-tasks: ImageNet [3], which is the benchmark for image recognition tasks, and MS COCO [15] which is for object detection. In the fine-tuning stage, PASCAL-VOC [4] and MS COCO were employed to fine-tune popular detectors. Several decisions influence the quality of the result, i.e., selection of augmentation techniques, encoder configuration, detector configuration (e.g., Faster R-CNN, Mask R-CNN), dataset configuration, training process, and hyper-parameters (e.g., weight factor λ, temperature τ). The paper proposed certain settings and used the average precision (AP) as an evaluation metric, summarizing the precision-recall curve. In fact, DenseCL-200ep (200 epochs pre-training on ImageNet) managed to achieve 58.7 and 40.3 AP on the VOC and COCO datasets respectively, outperforming MoCo-v2. The result of visual correspondence achieved by DenseCL-200ep is shown in Figure 5 against MoCo-v2.

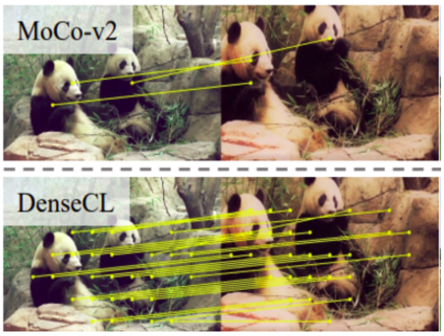


Fig. 5. Visualization of dense correspondence across two views of the same image. DenseCL-200ep extracts more high-similarity matches (≥ 0.9) than MoCo-v2. Image taken from [22].

DetCo framework was pre-trained on ImageNet and evaluated on PASCAL VOC and COCO datasets using the AP metric. Again, Xie et al. (2021) experimented with various detectors (e.g., Faster R-CNN detector, Mask R-CNN, RetinaNet) and parameter settings. DetCo-800ep (800 epochs pre-training on ImageNet) established a new SOTA, achieving 58.2 and 46.5 AP on VOC and COCO respectively. The result of this method is demonstrated in Figure 6 using once again MoCo-v2 as the baseline model.

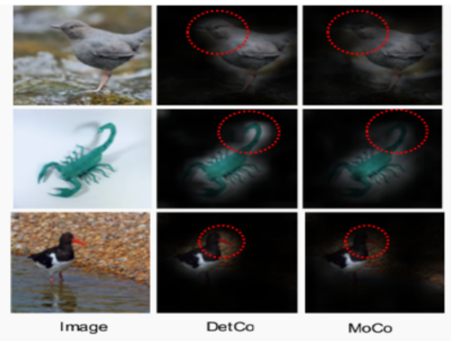


Fig. 6. Attention maps generated by DetCo and MoCo-v2. DetCo has a wider localization spectrum than MoCo-v2. Image taken from [24].

DCL evaluates the performance on the basis of datasets tailored for salient object detection task: MSRA-B [16], PASCAL-S [13], HKU-IS [11]. The MSRA-B is a pixel-accurate dataset of 5,000 images with mainly single salient objects, while HKU-IS consists of 4,447 images of either low contrast or multiple salient objects. PASCAL-S is a subset of PASCAL VOC containing 850 natural images. To evaluate the performance of DCL, the maximum F-measure (maxF) is used that corresponds to the harmonic mean of precision and recall. Figure 7 illustrates some indicative results of DCL against SOTA methods.

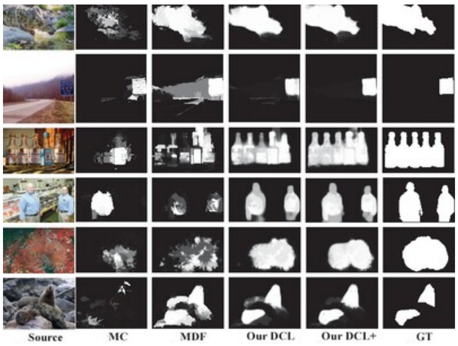


Fig. 7. Visual comparison of saliency maps generated from DCL and DCL$^+$ over SOTA methods. Image taken from [12].

Paper	Framework	Pre-training Datasets	Fine-tuning Datasets	Evaluation	Outcome
Wand et al., 2020	DenseCL	ImageNet / COCO	PASCAL VOC / COCO / ImageNet	AP	New SOTA in object detection
Xie et al., 2021	DetCo	ImageNet	PASCAL VOC / COCO / ImageNet	AP	Golden mean btw. image cls. & object detection
Li et al., 2016	DCL	ImageNet	MSRA-B / HKU-IS / PASCAL-S	maxF / MAE	New SOTA in salient detection

Table 1. Summary. DenseCL outperforms SOTA methods (i.e., both self-supervised and supervised pre-training) in object detection; DetCo is designed for object detection, but is also robust and competitive on image classification; DCL established a new SOTA in salient object detection.

6 Comparison and Discussion

In this section, we discuss the methods presented in Section 4 from a quantitatively and qualitatively point of view using their respective results. These methods are compared with respect to their accuracy performance, time efficiency, visualization results, and complexity.

6.1 Quantitative comparison

Initially, we quantitatively compare the aforementioned deep contrastive learning methods in terms of accuracy and time efficiency.

6.1.1 Accuracy

The performance of DenseCL is evaluated in terms of AP improvement over the baseline MoCo-v2 and supervised counterpart. Specifically in Table 2, we notice that DenseCL outperforms MoCo-v2 in object detection regardless of the nature of the pre-trained dataset. However, the increase in AP is higher when the pre-training dataset is similar to the downstream's one, which confirms that DenseCL adapts better to dense datasets, like COCO, than MoCo-v2. In addition, the performance between DenseCL and supervised pre-training on ImageNet is compared, proving that a self-supervised model might be indeed a better solution in many scenarios. Finally, when dealing with image-level end-tasks, i.e., image classification, MoCo-v2 outperforms DenseCL for 3.9% AP on ImageNet, indicating that DenseCL is not that adaptable to other end-tasks beyond detection.

Pre-training model	**MoCo-v2**			**Super. IN**	
	VOC	COCO	IN Cls.	VOC	COCO
DenseCL IN	+1.7%	+0.5%	−3.9%	+4.5%	+0.6%
DenseCL CC	+2.0%	+1.1%	N/A	+2.5%	-0.1%

Table 2. DenseCL AP-change over MoCo-v2 and supervised counterpart. 'IN' and 'CC' indicate the pre-training datasets ImageNet and COCO respectively. Both methods are pre-trained either on ImageNet for 200 training epochs or on COCO for 800 epochs. Faster R-CNN detector is utilized. 'IN Cls.' stands for ImageNet classification end-task.

Likewise, the performance of DetCo-800ep is evaluated in terms of AP improvement over the baseline MoCo-v2 and supervised counterpart. The results after fine-tuning and testing on PASCAL VOC dataset are presented in Table 3 ($3^{rd} column$). In fact, DetCo-800ep outperforms both MoCo-v2 and supervised pre-training but lags in performance over DenseCL-200ep. Moreover, in the 4^{th} column of this table, we compare the performance of DetCo-800ep with the best performance achieved on COCO by the same methods, while in the 5^{th}, the performance of DetCo on ImageNet classification is shown.

Overall, DenseCL may perform better on object detection, but to do so sacrifices classification performance. In contrast, DetCo is more adaptable to different end-tasks and seems to have achieved the golden mean between them. Surprisingly, both self-supervised methods improve by far the final detection accuracy over supervised pre-training, indicating that working on unlabeled data can prove valuable.

DCL and DCL$^+$ results are also compared with their SOTA methods on salience detection, but in this study, only the two methods that leverage deep learning are considered: MC [26] and MDF [11]. Table 4 illustrates the percentage increase of the maxF measure when DCL and DCL$^+$ are compared with these methods. It can be clearly seen that DCL itself improves the SOTA especially in datasets like PASCAL-S. In addition, using the CRF component in the DCL$^+$ variation improves even further the results.

Method	Epochs	PASCAL VOC	COCO	IN Cls.
super. IN	90	+4.7%	N/A	–
super. IN	200	+4.0%	+1.5%	–
MoCo-v2 IN	200	+1.2%	+6.7%	+1.1%
MoCo-v2 IN	800	+0.8%	N/A	N/A
DenseCL IN	200	−0.5%	+6.2%	5.0%

Table 3. DetCo-800ep AP-change over MoCo-v2, supervised pre-training, and DenseCL-200ep. For the PASCAL VOC all methods are pre-trained on ImageNet and tested on VOC07+12 using Faster R-CNN detector. For COCO object detection, different settings were chosen for each method. 'IN Cls.' stands for ImageNet classification end-task.

Fine-tuning datasets	DCL		DCL$^+$	
	MC	MDF	MC	MDF
MSR-B	+1.2%	+2.2%	+2.5%	+3.5%
HKU-IS	+10.5%	+3.6%	+13.3%	+5.0%
PASCAL-S	+10.1%	+6.7%	+11.1%	+7.6%

Table 4. DCL and DCL$^+$ percentage change of maxF over MC and MDF.

6.1.2 Efficiency

Next, we consider the efficiency of the deep contrastive learning methods with respect to their training time.

For DenseCL, each pre-training model is optimized on 8 NVIDIA TESLA V100 GPUs. A model pre-trained on ImageNet for 200 epochs needs approximately 55 hours, whereas on COCO for 800 iterations around 20 hours. According to Wang et al. (2021), the overhead compared to MoCo-v2 is less than 1% in both cases. Therefore, considering that the former works on the level of local features, whereas the latter on the level of global features, DenseCL is negligible slower taking into account its superior transfer performance.

DetCo utilizes 8 NVIDIA TESLA V100 GPUs as well. Although training time is not mentioned for this method, it is noted that with only 100 pre-training epochs, DetCo achieves almost the same performance as MoCo-v2 at 800 epochs, proving its high learning capability. However, due to its complex architecture, discussed in Section 6.2.2, we certainly expect more training time per epoch.

Lastly, DCL takes 25 hours to train on the MSRA-B dataset using a single NVIDIA Titan Black GPU and a 3.4GHz Intel processor. Then, the trained model needs 1.5 seconds to detect salient objects in a testing image of 400 × 300 pixels, while DCL$^+$ only 2.3 seconds. In contrast, MC with its unified multi-context model takes 31 hours to train and 1.6 seconds to generate the output salient map under similar computing power, while MDF needs 20 hours of training and 8 seconds for detection. In fact, DCL is significantly more efficient than the SOTA methods and this can be easily perceived considering that earlier methods process overlapped patches for the same image.

6.2 Qualitative comparison

Next, we qualitatively compare the presented deep contrastive learning methods in terms of visualization results and models complexity.

6.2.1 Visualization Results

In Figure 5, the high-similarity matches between two views of the same image are visualized. It can be clearly seen that DenseCL is a highly confident method that gives strong responses and extracts more matches than its baseline MoCo-v2 even under a high threshold value. Although the final visual detection of this method is not provided, we can safely conclude that this framework is capable of realizing more objects compared to the baseline model.

In addition, Figure 6 demonstrates the attention map of DetCo and MoCo-v2. In fact, DetCo can activate larger object regions than MoCo-v2, and its attention map seems to be more accurate in objects boundaries, thus proving its stronger localization capability.

Finally, Figure 7 demonstrates through different input images how DCL outperforms MC and MDF in terms of incomplete (first two rows), disconnected (middle two rows), and low contrast (last two rows) salient objects. DCL$^+$ improves even further the result, bringing it close to the ground truth.

6.2.2 Complexity

As we already discussed, common contrastive learning methods acquire global feature representations by processing whole images through one projection head and optimizing one global loss function.

DenseCL aims at incorporating both global and local representations by using two projection heads. The dense projection head has the same number of parameters as the global head, resulting in double learnable parameters. Moreover, the output from these heads is not only a global feature vector for each view but also dense feature vectors, thus increasing significantly the complexity of the model. All these vectors are finally, used in pairwise cosine similarity operations. However, according to Wang et al. (2021), by using efficient matrix operations, the introduced latency overhead tends to be negligible.

The complexity of the DCL method can be attributed to two key design parts of the architecture: (1) The modification of VGG16, which results in multi-level feature maps fed again into additional layers of convolution to attain multi-scale saliency maps —instead of one; (2) The segment-level saliency stream, where the generated segment-level visual contrast map requires three feature maps for each segment.

The most complex method is DetCo. As described in Section 4.2, DetCo pipeline consists of several computations, considering two views at the global level, 18 sub-views at the local level, multi-scale feature vectors (i.e., four) for each (sub)-view in both global and local representations, and three contrastive loss functions.

7 Conclusion

In this work, we presented three deep contrastive learning methods designed for object detection and saliency detection: DenseCL, DetCo, and DCL. DenseCL outperforms both self-supervised baseline and supervised pre-training on object detection but it is not adaptable to other end-tasks. DetCo manages to find the golden mean between image-level and dense prediction tasks by learning more universal representations. Finally, experimental results on DCL demonstrated that it can significantly improve the SOTA methods in salient object detection.

All in all, self-supervised learning seems to break new ground in computer vision by learning efficient representations without relying on explicit annotation and by incorporating local features as well. Even further, we firmly believe that learning universal representations able to adapt well to various tasks is now a laudable goal. One possible extension to this study would be to provide an analysis of these methods in respect to various end-tasks and quantify the exact level of sensitivity and adaptability of the contrastive learning approach.

Acknowledgements

The authors wish to thank the expert reviewer Dr. J. Guo for her valuable feedback, and Prof. R. Smedinga and Prof. M. Biehl for giving the required guidance in writing this paper.

References

[1] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proc. ICML*, pages 1597–1607, 2020.

[2] X. Chen, H. Fan, R. Girshick, and K. He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255, 2009.

[4] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[5] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.*, 27, 2014.

[7] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proc. CVPR*, pages 9729–9738, 2020.

[8] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proc. ICCV*, pages 2961–2969, 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016.

[10] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. CVPR*, pages 4681–4690, 2017.

[11] G. Li and Y. Yu. Visual saliency based on multiscale deep features. In *Proc. CVPR*, pages 5455–5463, 2015.

[12] G. Li and Y. Yu. Deep contrast learning for salient object detection. In *Proc. CVPR*, pages 478–487, 2016.

[13] Y. Li, X. Hou, C. Koch, J. M. Rehg, and A. L. Yuille. The secrets of salient object segmentation. In *Proc. CVPR*, pages 280–287, 2014.

[14] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proc. ICCV*, pages 2980–2988, 2017.

[15] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, pages 740–755, 2014.

[16] T. Liu, Z. Yuan, J. Sun, J. Wang, N. Zheng, X. Tang, and H.-Y. Shum. Learning to detect a salient object. In *IEEE PAMI*, pages 353–367, 2011.

[17] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *Proc. ECCV*, pages 69–84, 2016.

[18] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[19] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proc. CVPR*, pages 2536–2544, 2016.

[20] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.*, 28, 2015.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[22] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li. Dense contrastive learning for self-supervised visual pre-training. In *Proc. CVPR*, pages 3024–3033, 2021.

[23] Z. Wu, Y. Xiong, S. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *arXiv preprint arXiv:1805.01978*, 2018.

[24] E. Xie, J. Ding, W. Wang, X. Zhan, H. Xu, P. Sun, Z. Li, and P. Luo. Detco: Unsupervised contrastive learning for object detection. In *Proc. ICCV*, pages 8392–8401, 2021.

[25] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *Proc. ECCV*, pages 649–666, 2016.

[26] R. Zhao, W. Ouyang, H. Li, and X. Wang. Saliency detection by multi-context deep learning. In *Proc. CVPR*, pages 1265–1274, 2015.

Why You Should Use A Multi-GPU Platform

Marios Souroulla, Anton Bredenbals, *MSc, FSE*

Abstract— In the last couple of decades, sophisticated algorithms have been developed to solve various problems, like optimization tasks, machine learning, and object detection. However, most of these algorithms were developed to run sequentially on a CPU, and despite their impressive performance, they take a long time to execute. A method to achieve a boost in the performance would make them suitable for real-time applications, or in other cases getting results in reasonable times.

This is where the papers we are reviewing come in. They propose multi-GPU (Graphic Processing Unit) parallelizations of different algorithms, namely *k*-NN, the Viola-Jones algorithm, and a linear equation solver for topology optimization. We introduce the methodologies of the papers, their respective optimization tricks and their impact. Further, we will work out the different speed-ups that are achieved and set them into perspective of each other with regards to the different experimental setups. Finally we asses, whether multi-GPU computing is a viable option to deal with the aforementioned problems. We come to the conclusion that multi-GPU computing is a great option in situations, where time is of importance. Compared to CPU implementations, speed-ups are claimed to be between 4x and 800x, and the scaling from one to multiple GPUs also works efficiently in most cases. Although one need to keep the more complex implementation and optimization with its associated costs in mind.

Index Terms—Parallelization, GPU, Multi-GPU, Optimization, Time-sensitivity.

1 Introduction

Many parts of modern human life, like driving, shopping or even most tasks we do at our workplaces, especially in science, cannot be imagined without the support of various computational systems that make our lives easier. These systems are backed by strong algorithms that get increasingly complex and thus require longer computation times. But many of the applications are time-sensitive, because end-users are waiting to access the results, or even because real-time applications, like autonomous driving rely on them. Further, the volume of data that needs to be processed increases at high rates, for example the datasets for scientific studies or the data that is taken into account when modifying algorithms behind your favorite online shop.

The engineering of faster CPUs cannot keep up with this, so it is critical to find other ways to increase the algorithms' performance. A promising approach has been to parallelize the algorithms when possible. This allows multiple cores to make simultaneous computations and thus allows for multi-threading on the CPU or even GPU.

Multi-threading on a CPU, often referred to as multi-core, is restricted to using a few (usually 4-16) computing cores. These cores are quite powerful on their own and versatile, as they stay efficient even when ”working” on completely different threads. A GPU is a many-core architecture, consisting of a couple of hundred to a couple of thousand computing cores. An individual core is not as powerful, but the sheer number of cores makes up for that. Many-core architectures are more suited to computations that need to be performed on a huge number of individual elements, i.e. pixels in an image (for example in real-time graphics) or in general big data sets of different sorts. A common disadvantage of GPU-implementations is the higher implementation and optimization overhead that is inevitably associated with them.

Now that these possibilities are well known and widely used, the world has adapted these possibilities. But what is the next step? Datasets will not stop growing, real-time applications will get more complex or more data-hungry. So what can we do if even the CPU or GPU parallelized algorithms are too slow for our problem and we don't have one of these handy super-computers available? We say the answer is probably multi-GPU computing.

This paper focuses on discussing multiple approaches on Multi-GPU implementations for widely used algorithms. We will discuss if the speed-ups that can be achieved by using multiple GPUs is significant enough to warrant the increased implementational overhead, the costs associated with it and the additional required hardware. In the process, we will also introduce the optimization tricks used. We are also going to compare the different speed-ups achieved in the vastly different experimental setups. In the following section, we will briefly introduce each one of them. After that, the remainder of the paper is organized as follows: Section 3 introduces the relevant papers in more detail, by giving an overview over their respective methods, optimization tricks and experimental setups, Section 4 provides the results found by each of the researches. In Section 5, we will discuss the results of the different papers and point out their differences, and lastly, Section 6 summarizes our comparison and suggests future research.

2 Related Work

In 2015, Masek et al. [4] proposed a multi-GPU implementation of the well-known *k*-Nearest Neighbor (*k*-NN) algorithm. According to them, *k*-NN is a widely used and influential algorithm for classification, especially in the field of data mining. The data used is oftentimes high dimensional and consists of many data points; these properties make it well suited for many-core parallelization (in the algorithm the data samples are parallelized). The main contribution of the paper was a Java-based Multi-GPU implementation using the OpenCL library, which achieves very substantial speed-ups on moderate hardware when compared to a sequential CPU implementation. In 2016, Masek et al. [5] added an implementation using the CUDA API, which performs slightly better than the OpenCL implementation in most tests.

In 2017, Trompouki et al. [6] proposed a hybrid multi-CPU and multi-GPU implementation of the Viola-Jones algorithm [7] meant to be used for real-time pedestrian detection for Advanced Driving Assistance Systems (ADAS). They focused on providing an open-source Java implementation using the CUDA API, which meets the necessary ISO safety standard ASIL-D and runs fast enough to meet the strict requirements of volatile real-time situations.

- *Marios Souroulla, E-mail: m.souroulla@student.rug.nl.*
- *Anton Bredenbals, E-mail: a.bredenbals@student.rug.nl.*

In 2021, Herrero-Pérez et al. [3] proposed a multi-GPU based acceleration for Solid Isotropic Material Penalization (SIMP), a popular iterative density-based topology optimization method. For these methods, a fine mesh resolution is indispensable for reaching an accurate result. This leads to large linear systems of equations which make up for the vast majority of the computation time. Exploiting the data locality of this method, the authors tried to tackle two problems at once: the long computation times of these methods, which can reach well into multiple days using an optimized CPU implementation, as well as the limited available memory space on single systems and GPUs in particular.

This paper will provide an overview of the mentioned papers and compare these applications with respect to their corresponding speedups and discuss the different optimization tricks they use. To do this, we will also take the experimental setups into account and discuss their impact on the speed-ups.

3 Overview of Methods

In this section, we describe the various methods proposed in the literature, as well as the tricks they used.

3.1 Multi–GPU Implementation of *k*-Nearest Neighbor Algorithm

The first set of papers by Masek et al. [4],[5] focuses on an OpenCL and CUDA implementation of the *k*-NN clustering algorithm. In short, using *k*-NN, the label of a new point is the label of the majority of its *k* nearest points. Figure 1 shows the principle of the *k*-NN algorithm with an example. In that example, the star represents the new point we want to classify, if we choose the parameter *k* to be 3, then the new point is classified as ’red’ (because the majority of the 3 neighbors is red), but if we choose k=7, it is classified as ’blue’. In the naive brute-force implementation, the first step is to calculate all the distances between the new point and all the other points. The second step is to sort those distances, and the last step is to find the label of the majority of the *k* points that have the lowest distance.

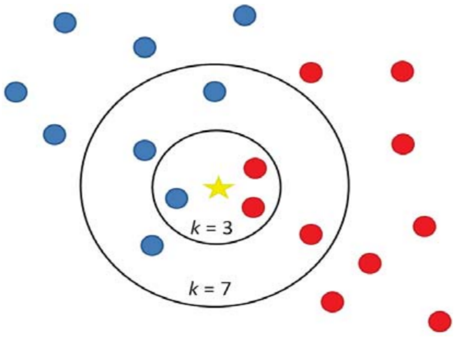


Fig. 1. The principle of the k-NN algorithm [5].

The algorithm is parallelized with respect to the training samples. The main change of this implementation is that the distances are mutually compared during their computation, and the *k* lowest distances are kept as the nearest neighbors. This change offers better memory usage as we do not need to store the whole list, but instead store only the current *k* nearest neighbors. Another trick the authors used to boost the performance is the use of the float4 vector format to save every training and testing example. This vector format contains four float values that are processed in one step. They also optimized the kernel by using local memory. These modifications aim to reduce the run-time.

3.2 Multi-GPU Implementation of Viola-Jones Algorithm

The second paper by Trompouki et al. [6] introduces a Multi-GPU and Multi-CPU implementation of real-time pedestrian detection that is based on the Viola-Jones algorithm [7]. An efficient implementation of the Viola-Jones algorithm is of great importance, because this will allow the use of it as a pedestrian-detection software in real-time applications, such as car navigation systems.

The algorithm uses so-called Haar features in order to detect certain patterns. The way these features are used is simple, each feature has two types of regions, a black one and a white one, the output of a feature is the sum of the pixels in the white region minus the sum of the pixels in the black region. This value is then compared to a (learned) threshold to get the final output of this feature.

Another important aspect of this algorithm is the classifier cascade, where the features (or weighted sums of features) are put into a cascade. The idea is to evaluate the next feature only if the previous features indicated the pattern. Figure 2 shows an example of a classifier cascade with K classifiers. Moreover, the algorithm transforms the image to an intermediate representation called *Integral Image* in which every pixel is replaced by the sum of pixels above and to the left. This representation allows for a more efficient evaluation of the features.

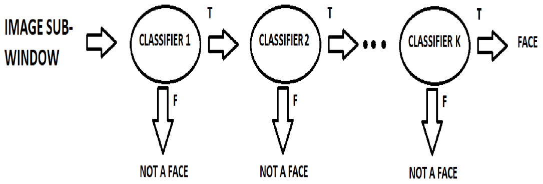


Fig. 2. Classifier cascade example with K classifiers.

The basic steps of this algorithm are to first read the image, compute the intermediate representation of different scales of the image (integral images), and then for each scale to go through all the subregions and apply the classifier cascade to detect the patterns you are looking for. The multiple scales are essential because the object we want to detect, pedestrians in this case, can be found in many different sizes. Taking this into account, we need to examine various scales. The corresponding Haar features need to be scaled as well.

The aforementioned paper [6] deploys a number of tricks to achieve a performance boost. The first modification has to do with the classifier cascade data structure. This structure is an array of stage classifiers, each stage classifier consists of an array of classifiers, and each classifier holds the set of features used for the detection. This structure leads to numerous problems like dynamic memory allocation on GPU, many small Direct-Memory-Access (DMA) transfers etc. The main modification regarding this was to allocate a single memory chunk and partition it internally, making sure that the smaller data structures representing the classifier are in contiguous memory locations. As a result, the entire classifier structure can be transferred to the GPU memory in one transfer. For this to be possible, the pointers need to be adjusted in order to be valid.

The next trick refers to the kernels. The image is scanned at various scales and each thread (all threads are launched in parallel) is assigned a sub-window and calculates all the features regardless of a fail in an earlier stage. This is done in order to avoid divergence among the threads; further, all threads are synchronised before proceeding to process detections.

Moreover, the authors propose yet another trick that deals with the sequence of the algorithm. In the sequential version of the algorithm, each sub-region is evaluated for each scale, so in pseudocode this translates into two nested loops, the outer loop goes through all the scales, and the inner loop through all the sub-regions of the corresponding scale. The modification the authors propose is to swap those two loops to increase the level of parallelism, since the sub-regions are much more than the scales and the algorithm is parallelized along its outermost loop. Moreover, with this modification, neighboring threads belonging in the same warp process sub-regions of the same size, because now they correspond to the same scale. This results in a more efficient use of cache, providing a speed-up this way.

Apart from these tricks, the authors perform some optimizations to better exploit a multi-CPU, multi-GPU architecture. The first such optimization is overlapping CPU and GPU operations. This is critical because in the CUDA programming model the jobs are launched synchronously, which means that the CPU waits for the GPU to finish its task before continuing with the execution of the program. However, if the jobs are instead launched asynchronously (which is also provided by CUDA), the CPU is able to perform other time-consuming tasks (e.g. read the next image) while it waits for the GPU to finish its task.

The next such optimization is overlapping GPU transfers and GPU computations. Even though asynchronous jobs allow us to overlap CPU and GPU operations as discussed above, the GPU computations are still serialized. The optimization in this case is to overlap GPU memory transfers and GPU computations. This is possible because the CUDA device that the authors use is equipped with a DMA (Direct Memory Access) controller. A CUDA stream represents a set of GPU operations that are to be executed in the given order. Such operations are executions of kernel functions or memory transfers from and to GPU memory. The idea here is to use two different streams in the same GPU device. While one stream is interacting with memory, the other stream can execute the kernel function, and vice-versa. This way, the authors achieved overlapping memory transfers and computation in a single GPU unit.

Furthermore, the authors propose yet another optimization trick to improve the CPU and GPU overlapping. The bottleneck currently is the reading and writing of the image as we will see in the next section. This is limiting the performance, since these operations are very slow and also blocking. This means that the process has to wait for the reading and writing to be done, in order to proceed. For the reading of the image not much can be done, but the writing phase can certainly be addressed. The authors use asynchronous writing operations which allow the process to make the I/O request and continue the execution without needing to actually wait for the write to be complete. A different approach would be to eliminate writes completely and not save the image at all, just display the image to the user. This is more efficient, however this is strictly application-dependent, since some applications might require saving the images for legal purposes. Nevertheless, by making the I/O requests asynchronous, a performance boost is achieved.

The last optimization performed is getting to use two GPUs with Message Passing Interface (MPI). For this to be possible, a second CPU is needed which will be controlling the other GPU. The way this was implemented, was to use MPI to create two separate processes running on each of the GPUs, and assigning each process to a different CPU, such that the CPU is the closest to the corresponding GPU. This was done to make the memory transfers as efficient as possible. This allows us to use multiple GPUs if they exist in the given architecture (two, for the case of the authors), boosting the performance even more.

Figure 3 shows two GPUs working in parallel, each of them having two streams that overlap memory operations and kernel computations.

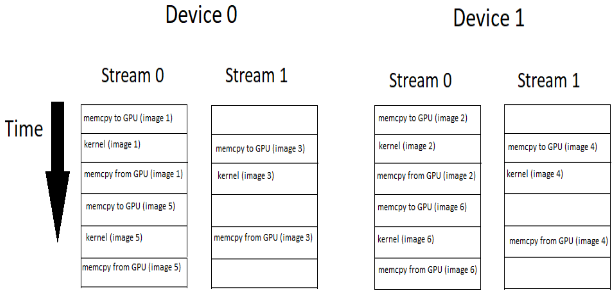


Fig. 3. Two GPUs, each of them having two streams, working in parallel.

3.3 Multi-GPU Acceleration of Large-Scale Density-Based Topology Optimization

The third paper examined by Herrero-Perez et al. [3] deals with density-based topology optimization. Topology optimization is performed to find the optimal distribution of material over a domain. This is done by solving a binary programming problem in such a way that a cost function is minimized while satisfying a number of constraints. A specific category of topology optimization is density-based topology optimization, which relaxes the integer-based problem with an interpolation scheme that penalizes a continuous density variable. This paper focuses on density-based topology optimization.

The main challenges in this area are the adaptation of proper algorithms to exploit the computing power and limited device memory of GPUs. In order to tackle these problems, the authors propose some optimizations. The basic idea is to split the problem into smaller and more easily manageable sub-problems which can be solved by different computing units, GPUs in our case. The main aspect is to balance the workload of each sub-problem and to minimize the communication and data transfers between processes handling different sub-problems.

The first optimization refers to domain partitioning, which is essentially the splitting of the whole domain into multiple non-overlapping sub-domains. This splitting is carried out once during the initialization phase and is kept throughout the optimization process. It is done by following some optimization criteria, more specifically the minimization of the number of surface elements in order to reduce the communication between processes.

The second optimization deals with distributed solving of the optimization problem. Let us consider the following linear system of equations:

$$Ax = b$$

where A is the coefficient matrix, x is the solution vector, and $\mathbf{b}$ is the right-hand side. A parallel strategy would require splitting the coefficient matrix A along p processes dealing with its corresponding sub-matrix of A. However, some parts of A need to be global and that requires some level of communication between the processes. Taking this into account, each process has an A_{loc} to store the local part of A, and A_{rem} to store the remote part of A which needs to be communicated with other processes. Moreover, each process calculates the global columns required for itself and the other processes. This results in the fact that each process knows where to send, and from where to receive data, so the communication is performed directly between processes. This also scales well and reduces the computational complexity and memory requirements because the number of neighbors and the amount of data exchanged is independent of the number p of processes.

The third optimization is the parallel computation of the objective function, the sensitivities, and the update of the solution vector x. These are trivial, due to the fact that they only require data that are stored locally. Hence, the computational unit responsible for a sub-domain can perform those computations independently and in a parallel fashion.

4 Results

In this section, we provide the results and experimental setups as they were presented in the corresponding papers.

For the first set of papers by Masek et al. [4], [5] which were about a multi-GPU implementation of the k-NN algorithm, the author performed their experiments using a CPU - Intel Core i7 3770@4.1GHz, L3 cache - 8192kB, 4 GPU - 2x3072 cores, mem. 2x4096MB@6 GHz, GPU - 1019 Mhz. The results they provide are shown in Tables 1 and 2. More specifically, Tables 1 and 2 show the speed-up obtained from the OpenCL and CUDA implementation respectively, compared to the single CPU approach. To put that into context, for the 4 million elements each having 10 attributes, it took the single core CPU over 31 hours to compute the result. Figure 4 shows the comparisons between the speed-ups achieved by using the CUDA and OpenCL implementations using up to 4 GPUs for various numbers of input vectors and number of attributes.

Table 1. Speed-ups vs single core CPU for k=5 neighbors, OpenCL.

	1 GPU	2 GPU	3 GPU	4 GPU
0.4 million inputs, 1000 attr.	75	146	215	278
2 million inputs, 100 attr.	89	178	266	353
1 million inputs, 10 attr.	172	318	454	600
4 million inputs, 10 attr.	176	341	508	671
1 million inputs, 4 attr.	206	389	576	745

Table 2. Speed-ups vs single core CPU for k=5 neighbors, CUDA.

	1 GPU	2 GPU	3 GPU	4 GPU
0.4 million inputs, 1000 attr.	77	151	221	290
2 million inputs, 100 attr.	92	184	275	359
1 million inputs, 10 attr.	146	289	434	548
4 million inputs, 10 attr.	150	301	450	578
1 million inputs, 4 attr.	240	465	675	882

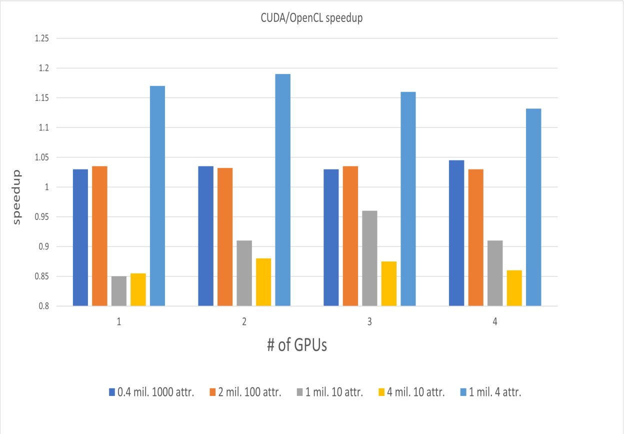


Fig. 4. Comparisons between the speed-ups achieved by CUDA and OpenCL for the k-NN algorithm

For the second paper, which is dealing with an efficient implementation of the Viola-Jones algorithm, all the experiments have been conducted on a Quad Core AMD with 2 Dual-Core AMD Opteron Processors 2222 @3GHz, with 1MB cache. The host is equipped with two NVIDIA Tesla C2050 graphic cards. The results are provided in Table 3 which shows the execution times for each of the different versions of the benchmark for 100 images. To put those numbers into context, the 2 GPUs 2 CPUs, no write version achieves processing of 21.2 frames-per-second (fps). This is easily calculated by dividing the given number by 100 (since it is for 100 images/frames) and then dividing 1 second with that. For example, if it takes 4.708 seconds for 100 images, then each image needs 0.04708 seconds, so each second we can process 21.2 frames. The resolution of the tested images was 640x480, and they have been selected from well-established datasets (Caltech [2] and INRIA [1]).

Table 3. Execution times of different versions for 100 images.

Setting	Time(sec)
1 CPU, no write	6m 56.626s
1 GPU 1 CPU, no overlap, no write	10.335
1 GPU 1 CPU, no write	8.331
2 GPUs 2 CPUs, no write	4.708
1 CPU, write	7m 8.314s
1 GPU 1 CPU, no overlap, write	19.771
1 GPU 1 CPU, asynchronous, write	15.863
1 GPU 1 CPU, write	16.164
2 GPUs 2 CPUs, write	13.125

Regarding the third paper [3], which examined performance boosts for density-based topology optimization, the authors used four compute nodes with an Intel Xeon W 2145 CPU @ 3.70GHz and 96GB of RAM. Also, two of the compute nodes use four low-cost AMD Radeon VII graphics cards.

Moreover, the authors did not provide any details of the performance of a 'naive' implementation. However, they did provide the execution time for the Finite Element Analysis (FEA) solving stage which is the most time-consuming stage, since it takes 91%-99% of the total time spent.

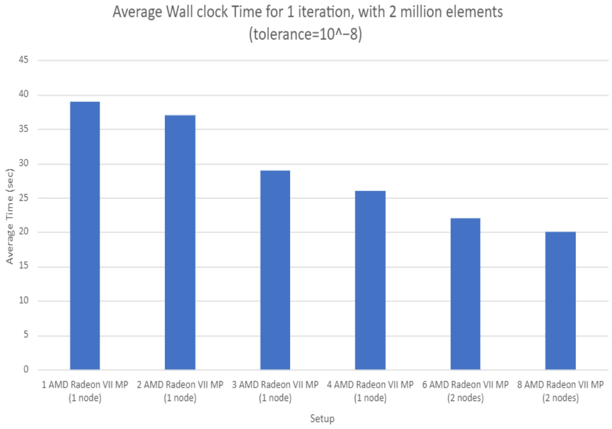


Fig. 5. Average Wall clock Time for 1 iteration, with 2 million elements and a tolerance of 10^{-8}.

Figure 5 shows the average wall clock time for one iteration of the topology optimization for one of their experiments (cantilever). The authors used two million hexahedral finite elements with tolerance of 10^{-8}.

Figure 6 shows the average wall clock time for one iteration of the topology optimization for one of their experiments (cantilever). The authors used one million hexahedral finite elements per GPU, with tolerance of 10^{-6} and reusing the solution of the last iteration.

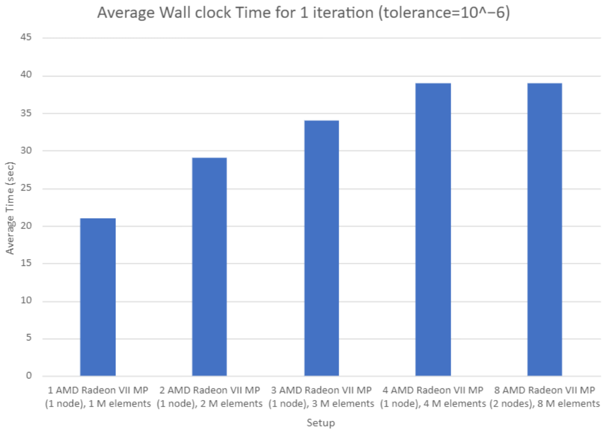


Fig. 6. Average Wall clock Time for 1 iteration and a tolerance of 10^{-6}.

Furthermore, the authors reported a speed-up of a factor of around 3.9 when using a multi-core with 32 cores in four computing nodes, versus many-core that uses 8 GPUs in two computing nodes. These numbers are regarding the Cantilever experiment, using eight million finite elements, with tolerance 10^{-8}.

4.1 Hardware Specifications Overview

Table 4 shows the hardware specifications for all the papers included in analysis.

Table 4. Hardware Specifications for all the papers.

	Setup	
Paper	CPU	GPU
K-NN [4],[5]	Intel Core i7 3770	NVIDIA GeForce GTX 690
	4 Cores	3072 Cores
	4.1 GHz	1019 Mhz
	8MB L3 Cache	1 MB L2 Cache
	32 GB Memory	4GB Memory
Viola-Jones[6]	AMD Opteron 2222	NVIDIA Tesla C2050
	2 Cores	448 Cores
	3.0 GHz	574 MHz
	1 MB L3 Cache	768 L2 Cache
	-	3 GB Memory
Topology	Intel Xeon W 2145	AMD Radeon VII
Optimization[3]	8 Cores	3840 Cores
	3.70 GHz	1750 MHz
	11 MB L3 Cache	4 MB L2 Cache
	96 GB Memory	16 GB Memory

5 Discussion

In this section, we will compare and discuss the results and improvements each of the previously introduced papers had to offer. This is not going to be a straightforward comparison because the experimental setups are immensely different. First of all, the papers by Masek et al. [4], [5] are the only ones who provide a reference given by sequential single core CPU computation. The other papers compare their findings to a parallelized single or even multi-CPU version. The latter is probably the most fitting comparison, as we want to discuss the impact of multi-GPU implementations in particular and not the impact of parallelization in the first place, as parallelization has been a staple for a while now and proven to be essential in many tasks. We began this chapter with these remarks, as they are essential to understanding the rather large gaps in the achieved speed-ups between the papers.

Looking at Tables 1 and 2, we were able to see that for all the given setups the use of multiple GPUs scaled almost linearly in efficiency, just dropping barely below a relative speed-up of 3.5 for four GPUs compared to one GPU in some cases of the OpenCL implementation. That is over 83% effective speed-up per GPU. What is interesting to note is that the factor of the speed-up mainly depends on the number of attributes the data has. Comparing 1 and 4 million inputs, both for OpenCL and CUDA, does not dramatically change the speed-up, although the speed-ups consistently increase with more inputs. The most probable cause for this is that the extra overhead for using GPUs decreases, comparatively to the computing time. Different numbers of attributes have a bigger impact, as we can see for both OpenCL and CUDA the speed-ups more than halve going from 4 to 100 attributes, even though the number of inputs is doubled.

It was interesting to see that OpenCL performed slightly worse in all cases but the ones with 10 attributes, where it had an edge compared to CUDA. This phenomenon can be seen in Figure 4 and is similar, no matter the number of GPUs. This can be attributed mainly to the use of the float4 data structure. Both the CUDA and the OpenCL implementations use it, but we assume that the CUDA compiler deals with empty fields in the data structure less efficiently, which results in comparatively slower computation times in the cases where the attributes are not divisible by 4. Furthermore, we can see that CUDA has the biggest performance lead over OpenCL in all cases with 4 attributes and just a minor but consistent lead with high attribute counts (100 and 1000). As mentioned before, these values were comparisons to a sequential CPU implementation, but nonetheless they show hugely impressive performance and great scalability to multiple GPUs. The authors optimized the algorithm and the GPU memory allocation but did not provide intermediate results, so we are unable to comment on their impact independently.

Comparing the results from Masek et al. [5] to the results of Trompouki et al. [6] in the second paper, on first glance the speed-ups for the Viola-Jones algorithm looked slightly less impressive. Disregarding the writing times, the speed-up from one CPU to 2 CPUs and 2 GPUs was around 104x, while the others were able to push it up to a 465x speed-up in one case (with two GPUs). Comparing the no write timings of the setup with 1 optimized GPU and CPU to 2 GPUs and CPUs we saw a decent performance increase of above 75%. This led us to believe that this approach can be scaled to more than two GPUs with a similar performance gain. Now, although not clearly stated in their paper, we believe that a parallelized CPU version on four cores was used to base their comparison on. Taking that into account, the results of the two papers look very similar in the achieved speed-ups.

Looking at the hardware of the two experiments, we found that Masek et al. [5] used a more powerful computing setup regarding the CPU and GPU, but the exact differences can not be considered here. In their paper, Trompouki et al. [6] mention that they achieved the minimum required fps for safe pedestrian detection in autonomous cars, complying with the ISO standard ASIL-D. Our first intuition led us to believe that these were unrealistic settings for autonomous cars, as having a multi GPU setup for just this one task is not really suitable for mass production of affordable cars. Looking into the used hardware however, this might be different now in 2022, as the technology used in the experiments was rather low-end, even in 2017.

In the experiments of Herrero-Pérez et al. [3], the scaling to multiple GPUs does not work as well as in the other cases, as we can see in Figure 5, with the 8 GPU setup being about twice as fast as the single GPU setup. Looking at the 4 and 8 GPU setups in Figure 6, we can see a constant time. In this particular case, the computing is split onto two separate compute nodes, connected via highspeed ethernet. This seems to scale perfectly, at least to two nodes, as the number of elements and the number of GPUs get doubled and the computing time stays constant. Looking at the similar setup in Figure 5, we can see a mere 25% speed-up. This difference can be attributed mainly to the different tolerance levels, while the smaller amount of elements could have had an impact, too.

Further, when they compare the GPU versions utilizing two GPUs to their multi-core (and multi-CPU) version, it achieves a large speed-up, while certainly being the more economic choice as well. In detail, they achieve speed-ups between 4x and 8x in different experiments, when using between two and eight GPUs compared to four 8-core CPUs (32 cores). Unfortunately, we cannot infer if the used CPU version scales better to multiple CPUs from their experiments as we do not have references with only one CPU. Seeing how much better the GPU version performs in comparison, this is definitely the most economic approach, even though it might not scale as well as the other algorithms into multiple GPUs.

Our analytics could not take the different hardware that was used into account to a satisfactory level. Comparing the technical details of the hardware, we can say with confidence that the CPU and GPU used for the topology optimization are the most modern and strongest, followed by the hardware used for *k*-NN and while Viola-Jones implementation was tested on the weakest hardware. However to get a more accurate and comparable result, one would have to use the exact same hardware (and software/drivers) for all three experiments. As the hardware and the experiment go together, we had no way to infer relative power between the components any further than this.

6 Conclusion

In this paper, we presented and discussed three different studies about highly parallelized Multi-GPU implementations. All of which were able to achieve significant speed-ups in their respective field when compared to sequential or parallelized CPU implementations. Nonetheless, the speed-ups range from 882x in the case of *k*-NN to about 6x in some cases of topology optimization. These differences can mainly be contributed to different experimental setups, as the *k*-NN experiment compared single core (sequential) execution times to the Multi-GPU version, while for the topology optimization, a 32 core multi-threaded CPU version is compared to the Multi-GPU approach.

Our findings mean that parallelizing algorithms which work with a large amount of data and/or very high dimensional data has a tremendous effect on the execution time. Furthermore, we found that using multiple GPUs over multiple CPUs is another huge step up in performance, and that splitting the work onto multiple compute nodes can be very valuable as well. If applicable, using both multiple CPUs and multiple GPUs leads to promising results, too, and might be the most economic option to get the maximum parallel computing power. Moreover we found that in many cases, GPU-optimized algorithms scale very well to multiple GPUs. We found an impressive effective speed-up of over 83% for parallelizing to multiple GPUs in the first paper (*k*-NN).

We think that in the future, we will see more and more algorithms being parallelized and optimized to run on multiple GPUs. Especially if the downsides of more implementation and optimization overhead and the associated higher cost is accepted. In many fields the amount of data that needs to be processed will steadily grow, while the available computing power of a single CPU or GPU will not be able to keep up as Moore's Law starts to slow down.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[2] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *2009 IEEE conference on computer vision and pattern recognition*, pages 304–311. IEEE, 2009.

[3] D. Herrero-Pérez and P. J. M. Castejón. Multi-gpu acceleration of large-scale density-based topology optimization. *Advances in Engineering Software*, 157:103006, 2021.

[4] J. Masek, R. Burget, J. Karasek, V. Uher, and M. K. Dutta. Multi-gpu implementation of k-nearest neighbor algorithm. In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, pages 764–767. IEEE, 2015.

[5] J. Masek, R. Burget, L. Povoda, and M. K. Dutta. Multi–gpu implementation of machine learning algorithm using cuda and opencl. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*, 5(2):101–107, 2016.

[6] M. M. Trompouki, L. Kosmidis, and N. Navarro. An open benchmark implementation for multi-cpu multi-gpu pedestrian detection in automotive systems. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 305–312. IEEE, 2017.

[7] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. Ieee, 2001.

GPU-Accelerated Frequent Itemset Mining: An In-depth Evaluation of GMiner

Willem Meijer and Leon Visscher

Abstract— The most valuable asset for modern businesses and industries is information, making efficient information generation methods invaluable. Frequent Itemset Mining is a form of data mining that is used to find non-trivial itemsets that frequently co-occur in a transactional dataset. Although many algorithms have been created to perform this task, over the last decade the amount of generated data has skyrocketed, making these algorithms no longer suitable. Multiple GPU-accelerated solutions have been proposed for improving the execution time of frequent itemset mining. One of those solutions is GMiner, an Apriori-based algorithm that uses the CPU for candidate generation and the GPU for support counting. The comparison performed in the original paper shows that GMiner outperforms several alternative solutions, however, one limitation of the comparison is that only real datasets are used. Therefore, it is unknown what the impact is of specific dataset features on the execution time of GMiner. The goal of this work is to investigate the relationship between dataset features such as number of transactions, average transaction length, number of unique items and data density, and the GMiner execution time. Additionally, the impact of the chosen minimum support value on the GMiner execution time is investigated. An experiment is performed in which GMiner is executed using various different dataset configurations. Our results show that the execution time of GMiner increases when the amount of input data increases and when the minimum support threshold decreases. Contrary to existing literature, we did not find a significant relation between data density and execution time.

Index Terms—Frequent itemset mining, GMiner, Data mining, GPU acceleration, CUDA.

1 Introduction

The most valuable asset for modern businesses and industries is information – data that is processed to be useful [1]. Information can be used to make informed decisions to, for example, improve product quality, increase revenue or decrease costs. Data mining is the process of extracting previously unknown and hidden information from large sets of data [2].

Frequent Itemset Mining (FIM) is a form of data mining in which frequently co-occurring itemsets of a transactional dataset are identified. An itemset is any subset of distinct items from a dataset. FIM was originally used for market basket analysis to identify what products of a store are frequently bought together. However, over the years it has been applied in a wide range of different areas such as bioinformatics, education, and sociology [14].

An itemset in a transactional dataset is considered frequent if it occurs in at least ξ percentage of transactions, where ξ is a user-determined minimum support value. The problem of mining frequent items has been shown to be computational intensive [5] because, in order to find all frequent items of a dataset, each subset of items must be scanned. This results in a time complexity of $O(2^n)$ where n is the number of items in the dataset. Popular algorithms for improving the time complexity of FIM are Apriori [6], FP-Growth [11], and Eclat [19]. Besides the number of items in a dataset, several other features of a dataset can impact the execution time. The number of transactions, the average length of the transactions, the density of the data, and the specified minimum support value can all influence the execution time of the algorithm. In a dataset with high density, items occur in a high number of transactions. Because of the impact of dataset features, existing comparison studies of FIM solutions often do not announce a clear winner because which algorithm performs best depends on the type of data.

In addition to the computational intensity of the frequent itemset mining problem, the amount of data produced by the world in recent decades has grown exponentially resulting in original algorithms such as Apriori, FP-Growth and Eclat no longer being suitable for processing these large datasets in a realistic amount of time [16]. This becomes clear when observing the results of Chee *et al.* [9], where analysing a small dataset using these algorithms already costs 30 seconds on average. Since the first appearances of Apriori, FP-Growth, and Eclat, no algorithms have been proposed that have resulted in significant performance improvement. Therefore, recent attempts for improving the performance of FIM focus more on how the algorithm is executed instead of on the algorithm itself.

One strategy for improving the performance of FIM is using hardware acceleration. Three high-performance computing platforms that can be used for hardware acceleration are Application-Specific Integrated Circuits (ASIC), Field-Programmable Gate Arrays (FPGA), and Graphical Processing Units (GPU), all of which can be used for frequent itemset mining. However, ASICs and FPGAs require a high degree of domain expertise, causing development processes to generally take longer compared to programming a GPU [8].

One promising solution for GPU-accelerated FIM algorithm is GMiner [10]. GMiner is an Apriori-based algorithm that uses the CPU for candidate generation and the GPU for support counting. In their work, they show that GMiner outperforms several existing FIM algorithms. However, one limitation of their comparison is that they only use real datasets, resulting in them having little control over the specific features of the datasets. The contribution of this study is to evaluate GMiner by performing an experiment using datasets generated with the IBM Almaden Frequent Itemset Generator [4] using different configurations for the total number of transactions (D), average transaction length (T), number of unique items (I) and (implicitly) data density which is calculated by T/I. Specifically:

The goal of this work is to investigate the relationship between dataset features and GMiner execution time.

The rest of this document is organised as follows. First, necessary background information and relevant related literature is presented and discussed in Section 2. This is followed by Section 3 where we discuss our methodology, research questions, and experimental setup. Section 4 shows the acquired results, which are further discussed in Section 5. After that, the threats to the validity of our work are considered and described in Section 6. Section 7 describes our conclusions and finally, in Section 8 we discuss what research directions could be explored in the future.

- *Willem Meijer is an M.Sc. Computing Science student at the University of Groningen E-mail: w.meijer.5@student.rug.nl.*
- *Leon Visscher is an M.Sc. Computing Science student at the University of Groningen E-mail: l.visscher@student.rug.nl.*

2 Background

The frequent item/pattern mining problem addresses the challenge to discover non-obvious patterns inside a large number of transactions. Given a dataset containing transactions, FIM can be used to determine what items frequently occur in the same transaction. This field has over 20 years worth of literature attached to it. In this section, an elaboration of the fundamental theory, a tertiary review of the FIM sub-fields, as well as a description of the GPU-accelerated state-of-the-art solutions are given.

2.1 Theoretical Basis

FIM uses a minimum support parameter that determines how frequently itemsets must occur in order for them to be considered frequent [5]. The naive solution to this problem is to simply generate all possible itemsets, and then calculate their support. However, the number of possible itemsets grows exponentially with the number of unique items in the dataset. Therefore, this approach is unsustainable for larger datasets. To resolve this problem, almost thirty years ago algorithms such as Apriori [6], FP-Growth [11], and Eclat [19] were proposed. Most modern state of the art solutions are in some shape or form based on these traditional algorithms. Therefore, this section briefly explains these algorithms. All three of these solutions are exhaustive, meaning that they are able to find all frequent itemsets.

2.1.1 Apriori

Agrawal and Srikant were the first to propose an algorithmic solution for FIM, known as the Apriori algorithm [6]. Their solution improves on the naive algorithm by excluding itemsets from computation that cannot have sufficient support. This is done according to the Apriori principle which states that an itemset can only be frequent if all its non-empty subsets are frequent as well. The Apriori algorithm is a breadth-first solution that consists of three steps: 1) candidate itemset generation, 2) support calculation, and 3) itemset pruning. At each stage of the algorithm, candidate itemsets of size k are generated by intersecting all itemsets of stage $k-1$. Next, all candidate itemsets for which a subset of size $k-1$ exists that does not have sufficient support are excluded. The initial itemsets are sets containing singular items. After the candidate itemset generation, the support is calculated naively by iterating through the entire dataset. Finally, candidates that have insufficient support are removed from the list of candidates, reducing the number of itemsets that is used to generate new itemsets for stage $k+1$. This sequence continues until no new candidates can be generated.

2.1.2 FP-Growth

Han *et al.* presented an alternative to Apriori, the FP-Growth algorithm [11] which uses an FP-tree data structure (an altered version of a prefix-tree) that is used to condense critical information of a dataset. It improves on Apriori by highly condensing the dataset, removing the costly generation of candidate itemsets, and applying a divide-and-conquer technique to reduce the search space. The algorithm consists of two parts: 1) FP-tree generation, and 2) frequent pattern mining. To generate an FP-tree, the entire dataset is traversed exactly twice, first, to identify what individual items are non-frequent, and second, to remove all non-frequent items from transactions and order the transactions. These transactions are inserted into the FP-tree, similar to how this is done in a prefix-tree. For every node three values are stored: the item it represents, a count, and a node-link that points to some next node that represents the same item (so they form a linked list). Every time a node is traversed in the insertion process, its count is incremented. This process condenses the entire data by a significant amount, as many transactions share the same prefix and they do not have to be redundantly stored in the FP-tree.

To mine frequent patterns, an FP-tree header is used which points to the first entry of the linked list for every individual itemset in the tree. For every individual item α, a traversal through this linked list is made, identifying all related potential patterns. All of these itemsets (minus the currently evaluated item) are then used to generate a new FP-tree, referred to as the conditional FP-tree of α. If this tree only has one path, all possible combinations of items are generated (unified with α). Their support is equal to the minimum support of an element in the itemset. If this tree does not have one path, the same procedure as described above is performed until a tree with only one path is reached.

2.1.3 Eclat

Zaki proposed another alternative to the Apriori algorithm, Eclat [19]. Eclat differs from Apriori and FP-Growth as its data is processed in vertical format (itemset → transaction-ids; $\alpha \rightarrow T_\alpha$) rather than horizontal (transaction-id → itemset); i.e. an item points to all of the transactions it is contained in. The vertical data structure that Eclat uses is fundamentally different from other FIM algorithms. Therefore, Eclat-inspired solutions commonly copy this approach. Regular datasets generally do not have this format. Therefore a translation step must be performed prior to running the algorithm. The algorithm itself repeatedly performs a number of steps that are very similar to Apriori: 1) support is calculated for each itemset, 2) itemsets are pruned, and 3) new candidate itemsets are generated. The support of an itemset is equal to the number of transactions it is contained in, and therefore, the cardinality of itemset α is equal to the cardinality of its respective entry in the dataset. Itemsets with insufficient support are removed from the considered candidate itemsets, and new candidates are generated by intersecting the resulting itemsets. Given two itemsets α and β, the resulting itemset is defined as $\alpha \cup \beta$ and their respective set of transactions as $T_\alpha \cap T_\beta$. After that, the algorithm continues at the first step until no new itemsets can be generated.

2.2 Related Work

Improving the performance of FIM solutions is a popular problem that many researchers have addressed. Because the number of solution directions, and therefore the number of algorithms, is incredibly large, describing the state-of-the-art for each of them is near-impossible. To capture the essence of the FIM sub-fields, a tertiary literature review is performed using various recent literature reviews [8, 9, 12, 14]. Additionally, two alternative state-of-the-art solutions [13, 18] are introduced.

Bustio-Martinez *et al.* [8], performed a review on various GPU- and FPGA-accelerated algorithms. These algorithms attempt to speed up the FIM process by parallelising various components of the process. GPU-accelerated solutions attempt to do so by using the GPU of a machine that allows processes to be parallelised with over a thousand threads. FPGA-accelerated solutions use the Field Programmable Gate Array (FPGA) to parallelize tasks. FPGA implements programmable logical gates, allowing programmers to implement algorithms at a closer-to-hardware level, making dedicated and more efficient builds possible. In their review, Bustio-Martinez *et al.* do point out that programming an FPGA solution requires more expertise than building a GPU solution.

Chee *et al.* [9] created a list of multiple state-of-the-art sequential algorithms. In their work, they categorise these algorithms into one of the three categories defined by Aggarwal *et al.* [3]: *join-based algorithms, pattern growth algorithms*, and *tree-based algorithms*; groups that are similar to the three fundamental algorithms explained in section 2.1. They conclude that the majority of modern algorithms extend FP-Growth as it only needs to scan the complete dataset twice. However, new algorithms must still be continued to be developed as the computation time needed to perform FIM still significantly increases as the amount of data grows.

Kumar and Mohbey [12] present a literature review on parallel and distributed solutions for FIM. They state that although distributed solutions scale much easier in terms of computing power and memory compared to parallel, single-machine solutions, these solutions introduce a significantly larger amount of communication overhead and introduce difficulties regarding fault handling, security, quality of service, transparency etc., giving single-machines an advantage in scenarios where a smaller amount of data is handled.

Luna *et al.* [14] performed a literature review exploring the field of FIM. FIM solutions are divided into sequential solutions, multi-thread solutions and distributed computing solutions. An elaborate overview and explanation of existing algorithms is given. Based on the number of citations for each paper they conclude that although there are some non-exhaustive search solutions that do not necessarily find all existing frequent itemsets, the FIM community is more focused on exhaustive search solutions. Apriori, FP-Growth, Eclat, and dEclat are identified as the most well-known solutions in the field. Other existing solutions are adaptations of these algorithms. Additionally, they emphasise that there is no clear winner in terms of performance because no objective comparison has been performed.

The various literature reviews introduced a significant number of sequential, hardware-accelerated, multi-threaded, and distributed FIM solutions. Opposed to the exhaustive algorithms some heuristic-based solutions have been proposed, which use heuristics to trade accuracy for performance [17]. However, these remain relatively disregarded as they cannot guarantee that the entire search space is explored [15]. Many studies that introduce new GPU-accelerated FIM solutions are evaluated in isolation, for which it is unclear what their position is within the literature. Because of this, only two other algorithms were found that can are considered competitive state-of-the-art alternatives: MCVG [13] and GFPG-LLMA [18].

Li *et al.* [13] introduce a closed FIM algorithm based on a vertical data structure similar to the Eclat algorithm. In order to preserve memory space on the GPU, they introduce a *Multi-Layer Vertical* (MLV) data structure. In this structure, each transaction id list is partitioned into multiple sub-layers, each of which can be represented by a much smaller object. This speeds up the support counting process significantly as heuristics can be used to accurately determine whether an itemset is frequent or not.

Wu *et al.* [18] introduce an FP-Growth-based algorithm in which memory access latency is reduced significantly. To achieve this, they change the underlying data structure as well. They replace the FP-Tree with an FP-Map that allows the algorithm to utilise coalescing memory whilst representing the tree data structure in a GPU-friendly manner. Additionally, they replace the recursion step that is used for support counting by replacing it with an iterative solution that is distributed across GPU blocks. In this solution, multiple sub-header tables are used (instead of one global header) to generate candidate itemsets for the next iteration.

2.3 GMiner

Chon *et al.* [10] proposed the Apriori-based GMiner algorithm which uses the CPU for candidate generation and the GPU for support counting. They introduce the *Traversal from the First Level* (TFL) algorithm to parallelise the counting procedure of the Apriori algorithm. They introduce two methods: *nested loop streaming* and *transaction blocks* to achieve this behaviour. Chon *et al.* use a vertical bitmap layout to represent a dataset. In this layout, transactions are represented as a binary number where the *i*th bit represents the *i*th item. As GPUs adopt a grid-like architecture, they partition the dataset into smaller chunks and distribute these across GPU blocks, making each GPU block responsible for counting the support of that partition, which is stored in a local hash table. The partial results of each block are aggregated using summation reduction, generating a single global hash table containing all candidate itemsets and their support. Because GMiner partitions the dataset across GPU blocks it scales relatively well with the amount of input data by distributing the additional transactions to nodes that were not used before. To maximise computation time and minimise communication overhead, they introduce *nested loop streaming* in which data transfer between the CPU and GPU is done in parallel with GPU computation.

One of the most common GPU bottlenecks is its limited memory space, which poses a risk when using the Apriori algorithm. During the candidate generation phase, the memory usage of Apriori increases significantly due to the large number of candidate itemsets that can be generated. To battle this problem, the GMiner algorithm exclusively uses 1-itemsets instead of $(k-1)$-itemsets to generate candidate k-itemsets. On top of reducing the memory requirements, they argue that this method speeds up the algorithm as well.

To account for datasets that have more items than there exist bits in GPU registers, they introduce the *HIL strategy* which splits individual transactions up into multiple *fragment blocks* that are handled asynchronously on different GPU threads.

3 Methodology

As stated in section 1, the goal of this work is to evaluate the work of Chon *et al.* [10] and gain deeper insights into the effect of dataset parameters on its execution time. The efficiency of algorithms can depend on a number of factors: the amount of data, the support threshold, and dataset densities.

3.1 Research Questions

The following research questions are defined:

RQ1 How does the performance of GMiner change when the amount of processed data changes?

RQ2 How does the performance of GMiner change when the support threshold changes?

RQ3 How does the performance of GMiner change when the dataset density changes?

To answer RQ1, GMiner is executed with a different number of transactions and a different number of transaction lengths. To answer RQ2, GMiner is executed with different minimum support values. To answer RQ3, GMiner is executed with two types of different parameters, 1) using datasets that have different transaction lengths, and 2) using datasets with different numbers of unique items. In each of these cases, the time used to complete the FIM task is recorded. After each run, the GMiner tool produces a report with information about the run, including the execution time. We use this reported execution time in our experiments.

3.2 Experimental Setup

The primary goal of this work is to measure the time performance of the GMiner algorithm. To increase replicability of this work

		# Items	**Avg. Trans.**	**Dens.**
Dataset	**# Trans.**	**(I)**	**Length (T)**	**(T/I)**
D100KT10I1K	98K	976	10.1	1.03%
D100KT25I1K	100K	976	24.9	2.55%
D100KT40I1K	100K	977	39.9	4.08%
D100KT55I1K	100K	977	54.8	5.61%
D250KT10I1K	246K	977	10.1	1.03%
D250KT25I1K	250K	977	24.9	2.55%
D250KT40I1K	250K	977	39.9	4.08%
D250KT55I1K	250K	977	54.8	5.61%
D500KT10I1K	492K	977	10.1	1.03%
D500KT25I1K	500K	977	24.9	2.55%
D500KT40I1K	500K	977	39.9	4.08%
D500KT55I1K	500K	977	54.8	5.61%
D1000KT10I1K	984K	977	10.1	1.03%
D1000KT25I1K	1M	977	24.9	2.55%
D1000KT40I1K	1M	977	39.9	4.08%
D1000KT55I1K	1M	977	54.8	5.61%
D500KT10I2	492K	1890	10.1	0.53%
D500KT10I3	492K	2768	10.1	0.36%
D500KT10I4	492K	3593	10.1	0.28%
D500KT10I5	492K	4337	10.1	0.23%
D500KT10I6	492K	5077	10.1	0.20%

Table 1: Evaluated Datasets

and to minimise background noise, the High-Performance Computing (HPC) Peregrine Cluster[1] is used, which is equipped with different computing nodes. In this work, the V100 nodes are used, which are equipped with 24 or 28 Intel Xeon 2.5 GHz cores or 64 AMD EPYC 7601 cores and a single virtual Nvidia V100 GPU. Because background noise can never be completely eliminated, especially on a shared computing cluster. As recommended by Beiranvand *et al.* [7], to further increase the reliability of the results, each individual experiment will be run multiple (3) times, after which the mean and standard deviation is presented in the results.

GMiner uses `CUDA` (version `10.2.89-GCC-8.3.0`), a general GPU programming environment that extends `C/C++` with several GPU-related functionalities. The code used for GMiner is acquired from their publically available repository[2], and the experimental code used in this work can be found in our repository[3]. Within the field of FIM, a wide variety of datasets is used for evaluation, differing in transaction length, number of unique items, and data density. To ensure a greater amount of control over the various variables required to answer the posed research questions, the commonly used IBM Almaden Frequent Itemset Generator [4] is used to generate a number of datasets (shown in Table 1). Datasets generated with this tool are created with several parameters. T is the average length of a transaction, I is the number of different items, and D is the number of transactions. The experiment is executed with all datasets represented in Table 1, and executed with all $\xi \in \{0.02\%, 0.06\%, 0.1\%, 0.14\%, 0.18\%, 0.22\%\}$.

4 Results

As introduced in Section 3.2, a large number of experiments was performed. For some configurations of dataset parameters, the experiment could not finish because of the process running out of memory. This was the case for the experiments with (T,ξ) combinations $(40, 0.02\%)$, $(50, 0.02\%)$, $(50, 0.06\%)$, and $(50, 0.1\%)$. The number of transactions in the dataset did not have an impact on whether the experiment ran out of memory. The charts shown in Figures 1, 2, 3, and 4 are an indication of the experiment results. All experiments have been replicated 3 times, for which the average observed standard deviation is 0.003 seconds, indicating that the time performance of the algorithm is stable.

Figure 1 shows the execution time of executing GMiner for four different values of average transaction length and number of transactions ranging from 100K until 1000K. It can be observed that execution time increases significantly with larger transaction lengths. Additionally, linear growth can be observed when the number of transactions in the dataset increases.

In Figure 2 the execution time is shown relative to the minimum support value. An inverse exponential distribution can be observed, indicating the high impact the minimum support value has on the execution time of GMiner. With execution time values ranging from 12.89 until 27.36 seconds for a support threshold $\xi = 0.00022$, rapidly declining to values all below 1 second for a support threshold $\xi = 0.0018$.

Figure 3 and 4 show the average transaction length and the unique number of items in a dataset respectively, relative to the execution time. Because there are two ways of altering the data density, 1) changing the transaction length and 2) changing the total number of unique items these results can be consulted to investigate the impact of the data density on the execution time.

5 Discussion

RQ1: How does the performance of GMiner change when the amount of processed data changes?

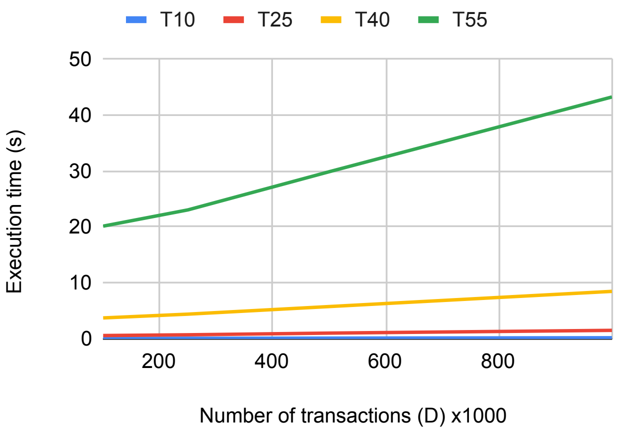


Fig. 1: Execution time of GMiner given datasets of different sizes where support $\xi = 0.14\%$.

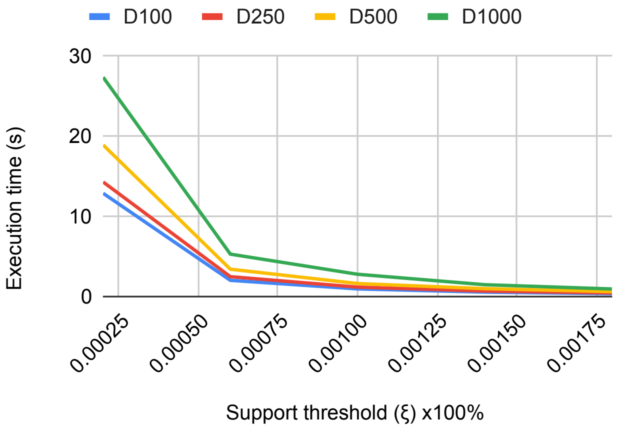


Fig. 2: Execution time of GMiner given different minimum support where the average transaction length $T = 25$.

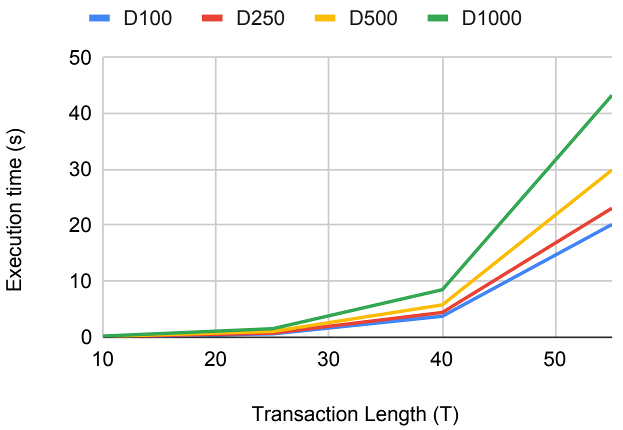


Fig. 3: Execution time of GMiner given different transaction lengths where support $\xi = 0.14\%$.

[1] Peregrine HPC Cluster: https://www.rug.nl/society-business/centre-for-information-technology/research/services/hpc/facilities/peregrine-hpc-cluster

[2] GMiner [10] repository https://github.com/coderbond007/GMiner

[3] Experiment repository: https://github.com/wmeijer221/colloquium

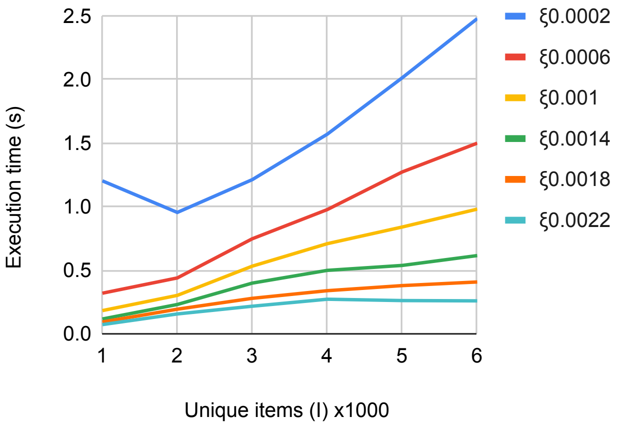


Fig. 4: Execution time of GMiner given different numbers of unique items where the average transaction length $T = 10$.

For evaluating RQ1 we are specifically interested in two dataset parameters. The amount of input data can increase by 1) increasing the number of transactions in a dataset or 2) increasing the average length of transactions.

As shown in Figure 1 the execution time of GMiner increases linearly with an increase in the number of transactions in the dataset. This can be explained because the GMiner algorithm uses the TFL strategy to distribute a larger number of transactions across a larger number of GPU blocks. This increases the communication overhead, as well as the hash table summation reduction overhead, giving a mild increase in execution time.

The vertical distance between the lines in Figure 1 as well as the results from Figure 3 clearly show an exponential increase in computation time when the average transaction length increases. This is expected because the amount of candidate itemsets increases exponentially with an increasing average transaction length. The algorithm battles memory access overhead during the candidate generation phase by only considering 1-itemsets instead of $(k-1)$-itemsets to generate k-itemsets. However, the overhead added during this phase is still significant enough to notice this exponential growth.

There seem to be two different relationships between the dataset size and execution time. The number of transactions causes the execution time to grow linearly while increasing the average transaction length causes the execution time to grow exponentially.

RQ2: How does the performance of GMiner change when the support threshold changes?

Figure 2 shows that GMiner has an exponentially larger execution time when the support threshold decreases. This can be explained according to the nature of the Apriori algorithm as it implements a pruning step during which the minimum support is used. During this step, all candidate itemsets with insufficient support are removed as any superset of this itemset would never be able to generate enough support due to the anti-monotone property. As the minimum support decreases, more itemsets will acquire sufficient support, for which they are not pruned.

As shown in Section 4 using a support threshold value of $\xi \leq 0.001$ in combination with a transaction length $T \geq 40$ introduces out of memory errors. As the number of candidate itemsets increases, the memory required to store these itemsets increases as well. In case the support threshold is decreased significantly enough, the number of candidate itemsets will outgrow the available memory.

In the gathered data, we, therefore, see an inverse exponential increase in execution time when the minimum support value is increased.

RQ3: How does the performance of GMiner change when the dataset density changes?

A higher density means a higher average support for itemsets. The density of a dataset is calculated by dividing the average length of a transaction by the total number of unique items and can therefore be increased in two ways: 1) the average transaction length is increased, or 2) the number of unique items is decreased.

While answering RQ1, we discussed that an increase in the average length of a transaction results in an exponential increase in execution time. However, it is not clear whether this is because of the increased size or the increased density. In Figure 4 it can be observed that as the number of unique items increases, the execution time of the algorithm also increases while the dataset density decreases significantly. In this case, the number of transactions and average number of items in a transaction stays the same, for which this difference can be fully contributed to the change in density.

It stands out that although density is commonly used as a descriptive parameter of a dataset, the results shown in this work cannot contribute to its value. The results show that both increasing and decreasing the dataset density can cause an increase in execution time. For this reason, the density of the dataset should be interpreted with some level of care as this value itself might not have a significant impact on the mining process, whereas the factors it is computed with might have.

In the results shown here, there seems to be no direct relation between dataset density and the execution time of the GMiner algorithm.

6 Threats to Validity

This section describes the threats to the validity of our research and the steps we took to mitigate these threats.

6.1 Method Validity

One drawback of measuring the performance of a running piece of software is that the actual measuring can have a direct impact on the performance. However, because the same behaviour is evaluated multiple times, this measurement overhead affects all results in the same manner. Additionally, the conclusions drawn do not consider absolute time and instead emphasise the relative time growth (time complexity) of the algorithm, making a minor overhead irrelevant to the drawn conclusions. To mitigate any additional noise, the experiments have been replicated a number of times.

The amount of FIM literature is very extensive, whereas the time available for this research is relatively limited. Because of this, literature that might have been relevant to this work might have been missed.

6.2 Conclusion Validity

Using standardised datasets allowed us to investigate the impact of specific dataset properties. Additionally, it makes it easy to replicate our study. However, a drawback is that the used datasets are synthetic datasets that might not be representative of real data. Therefore, we cannot conclude anything about how our results can impact real-life FIM applications.

7 Conclusion

In this work, we performed an experiment with the goal of investigating the relationship between dataset features and GMiner execution time.

To achieve this goal we have formulated three research questions: *1) How does the performance of GMiner change when the amount of processed data changes? 2) How does the performance of GMiner change when the support threshold changes?, and 3) How does the performance of GMiner change when the dataset density changes?*

By answering research questions 1 and 2 we have shown that the performance of GMiner behaves as expected when the amount of processed data or the defined support threshold changes.

Answering RQ3 resulted in the interesting find that, opposed to as expected by consulting existing literature, the data density has no significant impact on the execution time of the GMiner algorithm.

The experiment performed in this study extends the work of Chon *et al.* [10] by comparing their algorithm in a controlled environment using synthetic datasets. The results found for RQ1 and RQ2 are relatively unsurprising, whereas RQ3 yielded unexpected results. During this research, we have laid a basis for a standardised evaluation environment in which FIM algorithms can be compared. This framework can be used in the future for further evaluation of state-of-the-art FIM solutions.

8 Future Work

As mentioned in the previous section, because all datasets used in the experiment are standardised and publicly available we believe this work can serve as a framework for comparing other state of the art FIM solutions.

Based on the literature we have identified three GPU-accelerated FIM algorithms, GMiner [10], MVCG [13], and GFPG-LLMA [18] that all have outperformed earlier methods. However, it is not clear which of these methods has the best performance. Therefore, an objective comparison study would be invaluable. Unfortunately, the code/software for MVCG is not publicly available and the published version of the prototype for GFPG-LLMA does not work flawlessly, making executing and evaluating these solutions in a fair way impossible.

Additionally, by using the same datasets used in this experiment, any newly proposed methods can directly compare their results with the existing state of the art.

Acknowledgements

We would like to thank professor Asad Shahbahrami for the support and feedback provided during this research. Additionally, we would like to thank the various reviewers that helped point out opportunities for improvements in our work.

References

[1] R. L. Ackoff. From data to wisdom. *Journal of applied systems analysis, 16(1)*, pages 3–9, 1989.

[2] P. Adriaans and D. Zantinge. *Data mining*. Addison-Wesley, Harlow, England ; Reading, Mass, 1996.

[3] C. C. Aggarwal, M. A. Bhuiyan, and M. A. Hasan. Frequent Pattern Mining Algorithms: A Survey. In C. C. Aggarwal and J. Han, editors, *Frequent Pattern Mining*, pages 19–64. Springer International Publishing, Cham, 2014.

[4] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, Dec. 1993.

[5] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, SIGMOD '93, pages 207–216, New York, NY, USA, June 1993. Association for Computing Machinery.

[6] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, volume 20, pages 487–499. Morgan Kaufmann, 1994.

[7] V. Beiranvand, W. Hare, and Y. Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848, Dec. 2017.

[8] L. Bustio-Martínez, R. Cumplido, M. Letras, R. Hernández-León, C. Feregrino-Uribe, and J. Hernández-Palancar. FPGA/GPU-based Acceleration for Frequent Itemsets Mining: A Comprehensive Review. *ACM Computing Surveys*, 54(9):1–35, Dec. 2022.

[9] C.-H. Chee, J. Jaafar, I. A. Aziz, M. H. Hasan, and W. Yeoh. Algorithms for frequent itemset mining: a literature review. *Artificial Intelligence Review*, 52(4):2603–2621, Dec. 2019.

[10] K.-W. Chon, S.-H. Hwang, and M.-S. Kim. GMiner: A fast GPU-based frequent itemset mining method for large-scale data. *Information Sciences*, 439-440:19–38, May 2018.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, May 2000.

[12] S. Kumar and K. K. Mohbey. A review on big data based parallel and distributed approaches of pattern mining. *Journal of King Saud University - Computer and Information Sciences*, Sept. 2019.

[13] Y. Li, J. Xu, Y.-H. Yuan, and L. Chen. A new closed frequent itemset mining algorithm based on GPU and improved vertical structure: A NEW CLOSED FIM ALGORITHM BASED ON GPU AND IMPROVED VERTICAL STRUCTURE. *Concurrency and Computation: Practice and Experience*, 29(6):e3904, Mar. 2017.

[14] J. M. Luna, P. Fournier-Viger, and S. Ventura. Frequent itemset mining: A 25 years review. *WIREs Data Mining and Knowledge Discovery*, 9(6):e1329, 2019.

[15] J. M. Luna, F. Padillo, M. Pechenizkiy, and S. Ventura. Apriori Versions Based on MapReduce for Mining Frequent Patterns on Big Data. *IEEE Transactions on Cybernetics*, 48(10):2851–2865, Oct. 2018.

[16] S. Masih and S. Tanwani. Data Mining Techniques in Parallel and Distributed Environment - A Comprehensive Survey. *International Journal of Emerging Technology and Advanced Engineering*, 4(3):453–461, 2014.

[17] S. Ventura and J. M. Luna. *Pattern Mining with Evolutionary Algorithms*. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2016 edition, 2016.

[18] Y.-C. Wu, M.-Y. Yeh, and T.-W. Kuo. Fast Frequent Pattern Mining without Candidate Generations on GPU by Low Latency Memory Allocation. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1407–1416, Dec. 2019.

[19] M. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May 2000.

Applications of Linux Extended Berkeley Packet Filter (eBPF)

Sina Rezagholipour and Iulia-Cristina Tomoescu

Abstract—For creating networking, security or observability features, the Linux kernel is the best choice. But there are a few issues with it, most notably the difficulty of debugging when we have a complex infrastructure or many abstract layers. So this is why the Extended Berkeley Packet Filter (eBPF) was developed. In our paper we will dive into the numerous application of Linux eBPF and how its evolution impacted different fields in Computer Science, network security being the most important one. The main focus is reviewing and classifying the four given papers by their use cases.

Index Terms—Extended Berkeley Packet Filter, eBPF, Network Security

1 Introduction

The eBPF (Extended Berkeley Packet Filter) is a technology used to efficiently extend the capabilities of an operating system kernel without the need of changing kernel source code. Some of the applications will be discussed in further detail in the next sections of this paper. The Berkeley Packet Filter has a wide range of applications, which may be broadly classified into four categories: Networking, Security, Tracing and Profiling, and Observability and Monitoring.

eBPF was introduced in 2014 with Linux 4.0. It was initially implemented in kernel version 3.18 and essentially enables us to run applications at kernel level. It may be thought of as a sandbox (a software testing environment that enables the independent execution of applications for the purpose of evaluating, monitoring, or testing them one by one) virtual machine embedded within the Linux kernel. eBPF elevated BPF's ideals to new heights by adapting the technique to be more efficient while also taking use of newer hardware generations. It is critical to emphasize that eBPF has applications beyond packet filtering. Additionally, performance analysis, debugging, tracing, and security are covered. Due to its features, eBPF is still regarded as an intriguing technology in the Linux and cloud communities today.

We can add eBPF to several different kernel functions, giving them access to the data that the function is currently processing and allowing them to modify that data. eBPF was developed to improve the Linux tracing tools and is available primarily on the BSD and Solaris OS, with dynamic It was inspired by dtrace, a tracing tool; unlike dtrace, Linux did not provide an overview of the running system. Instead, it was limited to a specific frame of library calls, function calls, and system calls.

We will review and classify a number of four papers, namely Fingerprint-Based Automated Rule Generation for DDoS Mitigation using the Berkeley Packet Filter[5], Combining System Visibility and Security Using eBPF[3], Leveraging eBPF for programmable network functions with IPv6 Segment Routing[10] and eBPF-based Content and Computation-aware Communication for Real-time Edge Computing[1].

In our paper we will answer two questions:

RQ1: What are some of the application of Linux eBPF and how its evolution impacted different fields in Computer Science?

RQ2: In which category does each paper belongs based on their use case?

2 Background

The Berkeley Packet Filter was eBPF's forerunner, having been constructed for the first time in 1992 at Lawrence Berkeley National Laboratory by Steven McCanne and Van Jacobson [8]. It was created to aid in the monitoring and analysis of network traffic under Linux. It enables the raw packets of the link layer to be accepted and delivered by providing a raw access to the data link layers. This is accomplished by generating device nodes that are not attached to a physical device, which may be thought of as a ”virtual device” that is typically connected to one of the system's network interfaces. Two operations are performed on the device: reading from it (reading the incoming packets) and writing to it (introducing the packets in the interface).

To begin, there are a few fundamental ideas that must be understood in order to comprehend eBPF. On the one hand, the kernel space is reserved for the exclusive purpose of executing the kernel, its extensions, device modules, and drivers. The Linux kernel space enables complete hardware access. On the other hand, all user mode applications run in the user space, which can be swapped out if needed. User-space processes usually run in their own virtual memory space and, in most of the cases, they do not have access to the each other's memory. We illustrated the user space and the kernel space in Figure 1.

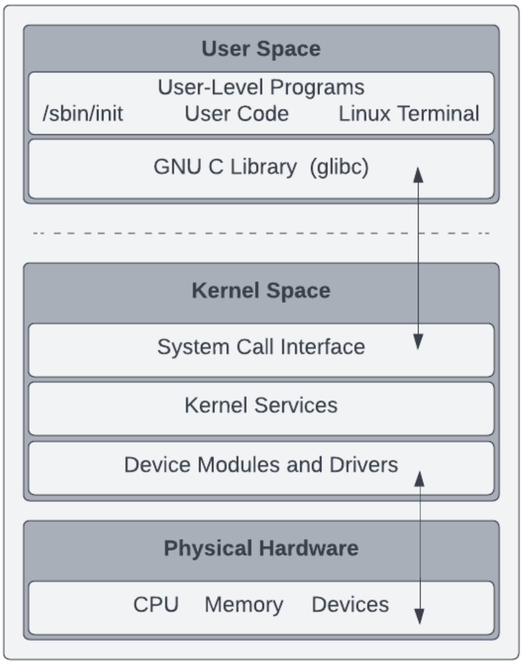


Fig. 1: User and kernel space.

- *Iulia-Cristina Tomoescu is a student at University of Groningen.*
  E-mail: i.tomoescu@student.rug.nl
  Student number: S4913531
- *Sina Rezagholipour is a student at University of Groningen.*
  E-mail: s.rezagholipour@student.rug.nl
  Student number: S4589483

We now have the loadable kernel module (LKM), which dynamically inserts or deletes code from the Linux kernel. It allows the kernel to work with the hardware when the kernel does not need to understand how the hardware functions. LKMs have many security vulnerabilities and a potential risk of crashing the kernel. Additionally, developing and maintaining kernel modules is a significant amount of work. eBPF programs do not replace LKMs, but they allow us to make use of the protected hardware resources whilst not being a hazard for the kernel.

2.1 Components

eBPF programs need and event-driven environment, therefore they are executed when a process reaches certain locations in the kernel, known as kernel hooks. Preset hooks are available for a variety of events, including entrance and exit functions, system calls and network events. If there is no preset hook that meets our specific need, we can also create a kernel probe (kprobe) or a user probe (uprobe) that allows the incorporation of eBPF programs at nearly any location in applications.

A helper function might be called after the eBPF program is triggered. It is worth mentioning that these functions have to be defined by the kernel and they cannot have more than five arguments. This list is still expanding and it can be easily found in the user manual of Linux. These functions are able to carry out a broad range of tasks, such as printing debugging messages, performing operations on data, communicating with eBPF maps, redirecting packets and generating random numbers. There is also possible to run another eBPF program that will take over the current execution conditions. This process is known as a tail and function call.

eBPF works with maps (key-value pairs) in order to manage data storage and sharing between the application and the kernel/user space. By using the helper functions mentioned before, programs can transmit and receive data in maps using a wide variety of different data structures, including hash table, stack trace, array, stack trace, ring buffer, LPM (Longest Prefix match) and LRU (Least Recently Used).

2.2 Architecture

Now that we are familiar with eBPF's components, in the following we will describe how eBPF internally works. In Figure 2 the architecture of eBPF is displayed.

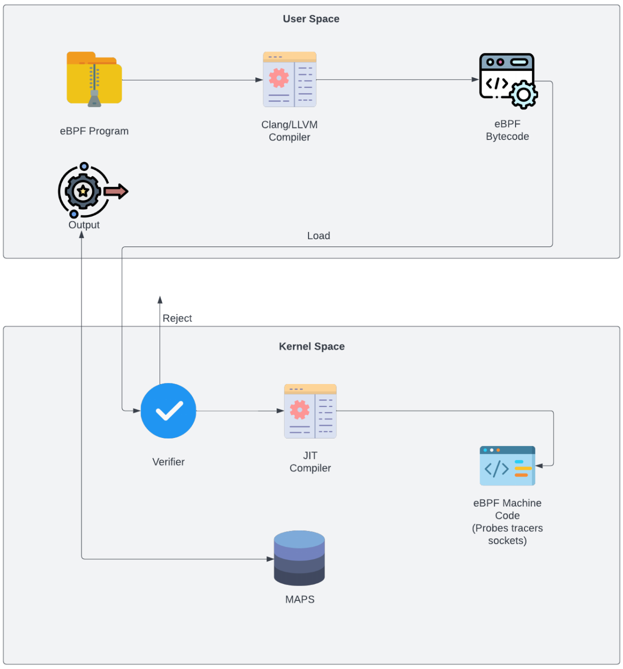


Fig. 2: Architecture of eBPF.

Because the kernel only accepts eBPF programs if they are loaded as bytecode. Therefore, we require a method of compiling higher level languages and in order to develop this needed compiler, eBPF leverages the LLVM back-end architecture, that can be used to construct a front-end for any programming language. Clang is the front-end programming language used for eBPF applications since they are written in C.

However, the compiled bytecode must first pass a set of tests before it can be put onto the hook point. An in-kernel Verifier can prevent programs from looping, not having the proper permissions, or crashing the kernel by simulating the program in a virtual machine-like architecture. The program must pass all the tests.

Following the verification step, the eBPF bytecode is converted to native machine code by the Just-In-Time (JIT) compiler. eBPF's design features 64-bit encoding and a total of eleven registers. This makes eBPF very similar to x86_64, ARM, and arm64 hardware architectures.

2.3 Use cases

This section offers an overview of the four main categories of eBPF's use cases mentioned in Introduction.

Security: Extending the fundamental capabilities of viewing and understanding all system calls, as well as giving packet and socket-level views of all networking processes, allows the creation of groundbreaking approaches to system security. Historically, completely autonomous systems handled various components of system call filtering, process context tracking, and network-level filtering. But eBPF makes possible the simultaneous management and visualization of all aspects and this gives us the possibility to design security solutions that function with a higher degree of context and control.

Networking: Because eBPF is both efficient and programmable, it is an excellent choice for all packet processing needs in networking systems. Therefore, insertion of protocol parsers and modifications in the packets forwarding are possible because of the programmability, whilst performance comparable to that of in-kernel code is given by the effectiveness of the JIT compiler.

Tracing and Profiling: With eBPF programs we are not only able to discover the kernel/user probe points, but also additional (custom) points. Hence eBPF provides significant insight into both application and system runtime activity and allows us to easily find system performance difficulties.

Observability and Monitoring: eBPF significantly increases the depth of visibility whilst also notably decreasing the general processing time required by the system software. This is achieved by collecting only the needed visibility data and by generating data structures at the source of the event, such as histograms, rather than being dependent on sample export.

2.4 Advantages

In this section we will address the most significant benefits of eBPF:

- Speed and performance: eBPF enables the kernel-space to delegate the packet processing to the user-space. The eBPF program is called after the compilation of the bytecode, rather than a new interpretation of the bytecode for each method.

- Unintrusive: eBPF can be used as a debugger and when this is the case, we do not have to stop the application in order to examine its activity or status.

- Security: The kernel source code is unaltered and the verifier guarantees that programs will not execute indefinitely.

- Visibility: Additionally, eBPF provides a centralized, robust, and easily accessible platform for process tracing. It not only leads to an increase in security, but in visibility as well.

- Convenience: Building and maintaining kernel modules is rather difficult and it requires low-level programming languages. Hence, writing code for the hooks or helper functions is easier.

- Power: eBPF is able to execute functions that are often reserved for high-level programming languages. It can trace everything in a system.

2.5 Disadvantages

Even though eBPF is a powerful tool, it has certain limitations such as:

- Portability: eBPF is only accessible on Linux kernels.
- Infrastructure requirements: The kernel version must be quite recent, newer than 4.13 which was released in 2017.
- Functionality: Limiting the resources and components of the operating system that a program can access improves security, but at the cost of functionality.

3 Literature

This section summarizes the four papers that make the subject of our study and mention a few applications of eBPF that we found intriguing.

3.1 Paper 1: Fingerprint-Based Automated Rule Generation for DDoS Mitigation

DDoS or Distributed Denial of Service attacks and threats have become more prevalent in society in recent times. It is worth noting that in these attacks, targets are specifically flooded and bombarded with excessive internet traffic. At the same time, the maximum bandwidth's intensity has also increased. For contributing to the efforts of mitigating these attacks, DDoSDB was launched by Jair Santanna from Twente University. This platform is accountable for storing fingerprints that tend to contain an evaluation of different aspects and features of DDoS attacks and threats. In addition, users are enabled by the platform to download the traffic of the attack that can be utilized for replay attacks. The use of these fingerprints enables the generation of mitigation rules for different DDoS attacks.

There is no doubt that in the past few years, a number of technologies such as Web Application Filters, Intrusion Detection Systems, Flowspec, and even Border Gateway Protocol have emerged. It is, however, important to note that these technologies specifically tend to target routers and often function or work on top of different operating systems. It leaves low-level functionality that is unused for the mitigation of DDoS attacks. Therefore, for filling this gap, the author have carried out this study.

It is indicated by Koelewijn that the current study focuses on the utilization of eBPF's extended version. Actually, it is a technology that generally enables the filtering of packets in even the lowest operating system layer, which is the kernel. In the kernel, filtering packets are capable of having different performance advantages. The author has, therefore, designed a method or technique for the automated generation of eBPF for the mitigation of DDoS on the basis of attack fingerprints that are acquired from DDoSDB.

It is determined by the author that the existing studies tend to focus only on the eBPF performance or capability in packet filtering rather than focusing on accuracy. When it comes to DDoS mitigation, eBPF is highly promising. There is little to no information in attack fingerprints about the value probability and it tends to simplify the conversion of different fingerprints to different rules and to one individual threshold that can be configured. When there is a use of probability theory, it results in the reduction of fingerprints which helps in obtaining optimal configuration.

In addition, it has been indicated by the author that the proposed method influences or increases the accuracy to a significant extent, approximately more than 95 percent. In some other cases, the accuracy for the proposed method or technique has actually been above 90 percent as well. It indicates that the proposed method or technique for comprehensive automated eBPF generation on the basis of fingerprints acquired from DDoSDB tends to satisfy all the requirements that have been identified and determined in the previous studies and researches. Future studies need to further explain and describe the eBPF possibilities for the mitigation and prevention of DDoS attacks. Furthermore, there is a need for further investigating the use of eBPF for different attack protocols [5].

3.2 Paper 2: Combining System Visibility and Security

Typically, IDSes or Intrusion Detection Systems on the basis of networks such as Bro, Suricata, and Snort tend to passively monitor the network traffic that is acquired from a variety of Terminal Access Points or mirror ports. In a similar manner to antiviruses, it should be noted that most of the IDSes are based on signatures. It means that they generally determine problems and exhibit different types of alerts through the extraction of patterns from the traffic that is captured by them. Then, it is compared against different patterns of predetermined attacks and threats within the database.

For performing these activities and tasks, different network packets are required to be reassembled and defragmented in streams, and then, their content has to be compared against the recognized signatures. Actually, Intrusion Prevention Systems (IPSes) are fundamentally based on IDSes with the capability of bridging the traffic across two different interfaces of the network. In recent years, the rise in the speed of the network together with the encryption of data has led to a number of challenges to not only IPSes but also IDSes. These challenges are associated with not only performance but also with visibility because the signatures are unable to be identified with the encryption of the traffic. Some further challenges have been created by the use of virtualization at the level of operating systems.

The lack of visibility tends to limit the utilization of different security tools that are based on packets. Therefore, in this study, Deri, Sabella, Mainardi, Degano, Zunino have leveraged eBPF for precisely combining system introspection with an effective security policer at the level of the system which allows the development of different valid security policies that are customized for specific users, containers, and processes. It serves as a major advancement for different network security applications that are capable of being benefitted from system introspection for enriching the information that is specifically extracted from different network packets while paving the way for the application of network and systemaware security policies that tend to combine the security and visibility at a fraction of the existing solutions' computational costs.

In this research, the designed solution can actually be logically segregated into two major components. First, a library that is accountable for the development of eBPF events related to the network. Second, an application, which is responsible for the description of the implementation and architecture of ntopng and libebpfflow. It has been identified by the authors that the proposed scheme or framework has indicated that the integration of system and network information is actually a step ahead with respect to different approaches that are purely based on packets. It is shown by the authors that the capability of going beyond ports, protocols, and IP addresses, actually enables the development of a new generation of different security protocols and tools that consider the metadata at the system level including containers, processes, and users. While the framework is effective, a major limitation is concerned with the inability of developing loops in the scheme of eBPF. In the future, there is a need to resolve this issue [3].

3.3 Paper 3: Leveraging eBPF for programmable network functions with IPv6 segment routing

In general, Xhonneux, Duchene and Bonaventure [10] determine that with the advent of SDN or Software Defined Networks, SFC or Service Function Changing and NFC or Network Function Virtualization operators generally expect and suppose networks to support different flexible services beyond just forwarding different packets. It is important to note that the network programmability framework that is being created with the IETF through the leveraging of Segment Routing of IPv6 allows the realization of different in-network functions and operations.

It is further determined that segment routine was initially developed as a simplification of MPLS or Multiprotocol Label Switching for different ISP networks and it has eventually evolved into a specifically more generic solution. It is worth noting that segment routing is quite an effective and modern version of the fundamental source routing. It allows routers to actually forward different packets along a succession of different shorter paths and each of them is identified or determined by a specific segment. Meanwhile, when it comes to SRv6, it generally leverages the flexibility of the packet format of IPv6 and even the large addressing space of IPv6. It should be noted that with this format, each and every format is encoded as an address of IPv6 that is advertised through the protocol of intradomain routing.

In this research, the authors have focused on demonstrating this in-network protocols' vision can be realized in an effective manner. Through the leveraging of eBPF support in the kernel of Linux, a flexible framework has been implemented by the authors that enable network encoders to precisely encode their own different network functions and operations in the form of an eBPF code. This code is actually executed automatically while processing different packets. It is indicated and determined by the authors that the overhead of using and calling these types of eBPF functions is acceptable. With the use of Extended Berkley Packet Filter, it becomes possible for the operators to implement different network functions. In addition to it, the Linux kernel is utilized by the authors for the description of their architectures and Linux 4.18 has been used for the release of the extension.

In addition to it, three different use cases are considered by the authors for the demonstration of the flexibility of their approach. These use cases include network discovery, hybrid networks, and delay measurements. It is also indicated by the authors that upon running eBPF network operations on the routers of Linux, the performance penalty does not really experience or incur a large overhead. Instead of it, the results are substantial and even acceptable to a significant extent. In fact, it has been indicated by the authors that the functions of eBPF generally have a minimal overhead as compared to the one that is developed by their different static variants. Their key benefit is concerned with the fact that they are quite generic. New ways have been opened by the proposed framework for researchers and network operators to implement a variety of in-network functions.

3.4 Paper 4: eBPF-based content and computation-aware communication for real-time edge computing

Generally, interconnected sensors, based on IoT or the Internet of Things tend to transmit and collect data for precise analysis and evaluation to different remote servers. The delivery of authentic data, however, might incur loss or even delay because of the limited resources of the network. There is no doubt that network congestion might influence the capability of the system to enable and support different real-time services including virtual reality, smart transportation, traffic monitoring, or even video surveillance. It is worth noting that for such applications, different paradigms of edge computing and cloudlets generally address the problem of latency by actually placing different computation-capable devices and products in a single-hop wireless topology. In various network scenarios, however, the coexistence of these intensive streams with several other services over limited networks requires effective and new solutions to a significant extent.

It is worth noting that recent frameworks on the basis of SDN or Software Defined Networks have exhibited the capability of improving the management of network resources with the use of NFV or Network Function Virtualization and dynamic flow control. At the level of communication, SDR or Software Defined Radios have been utilized for adapting the different parameters and aspects of wireless transmission. The main challenge and issue of properly using the available bandwidth for performing different real-time applications, however, is concerned with managing the traffic stands that are produced by these services and applications.

The bandwidth consumption in order to stream a video at time T, using N video sensors $\{S_1,...,S_N\}$, to M computational processes $\{C_1,...,C_M\}$ at maximum quality Q^{full}:

$$B_{total} = M \sum_{i=0}^{N} S_i(Q_i^{full}) \tag{1}$$

Therefore, the QoC or Quality of Computing framework has been proposed for relaxing and addressing the constraints of interference on different IoT streams while facilitating their proper use. Therefore, in this research, Baidya, Chen, Levorato [1] have proposed a communication control framework that is computation-aware for different real-time IoT applications and functions that generally result in high-volume traffic data that is processed and managed at the network edge. On the basis of QoC requirements, it should be noted that the framework generally offers real-time forwarding and replication of packets controlled by the user within the in-kernel VM or Virtual Machines with the use of Extended Berkeley Packet Filter. Actually, the implementation relies on the use of NFV and SDN for the achievement of dynamic and highly programmable replication of packets. It is indicated by the authors that the use of eBPF effectively improves the performance of the framework through the introduction of 10 registers of 64-bits while using the JIT or just-in-time compiler. At the same time, the programs of eBPF are capable of being invoked from a number of network stack layers including drivers, qdisc, and socket.

It should be noted that the proposed framework is specifically instantiated for addressing a scenario based on a case study where different video streams from a number of cameras are actually transmitted and transferred to the edge processor for precise and real-time analysis or evaluation. It is indicated by the numerical results that the proposed framework is better than other possible alternatives and frameworks in terms of system resource-saving, network bandwidth, and programmability. It is further determined that the proposed framework has the capability of adapting to different data streams that support remote processes of computation.

3.5 Related work

While doing research on this topic we discovered some noteworthy applications:

- ViperProbe is a framework based on eBPF. It is used for both collecting microservices that enables dynamic sampling, as well as collecting deep, diversified, and accurate system metrics. [6]

- BPFbox is a precise process confinement application that makes enforcing the principle of the last privilege possible at the user and kernel space functions. [4]

- eBPF allows the monitoring and tracing of software processes, therefore leading to discovering malware or abnormalities at the kernel-level. [2]

- Express Data Path (XDP) is a kernel network layer that processes packets closer to the network interface card in order to speed up packet processing [9].

4 Discussion

In this section we will classify the papers discussed in the last section by their use cases.

4.1 Paper 1: Security

The goal of the paper "Fingerprint-based automated rule generation for DDOS mitigation using the Berkeley Packet Filter" [5] is to develop a method to automatically create an eBPF to mitigate DDoS attacks based on DDoSDB fingerprints and evaluate its accuracy. In most distributed denial-of-service attack strategies, several distributed agents attack the target system simultaneously so that the resources of the target system are completely consumed: if the domain name of the Command and Control (C&C) server or download site is known, the zombified PC accesses the C&C server. This is a method to control responses to DNS queries from the zombie PC, so that when it tries to do so, it bypasses the DNS dump server and intercepts the attack command.

A protocol is a set of rules and instructions that allow devices on a network to communicate with each other. Network devices cannot transmit data without using the protocol. The operating system then uses the Just-In-Time (JIT) compiler and checking mechanism to ensure security and operability as if it had been originally compiled. This has resulted in a number of eBPF-based designs that can be used for a wide range of usage scenarios, including next-generation networks, observability and security features. Enabling high-performance networking and load balancing in modern data centers and cloud environments, extracting low-cost, fine-grained security observability data, and helping application developers track applications and troubleshoot performance issues.

The possibilities are endless, and the innovation that eBPF brings has only just begun. The author points out that existing research tends to focus only on the performance and filtering capability of eBPF packets, and not on accuracy. eBPFs hold great promise in the fight against DDoS.

Attack traces contain little information about the probability of the values, which makes it easy to convert different traces into different individual rules and thresholds that can be set. When probability theory is used, this leads to a reduction of fingerprints, resulting in an optimal configuration.

4.2 Paper 2: Observability and Monitoring

In ”Combining system visibility and security using eBPF” [3] we dive into how eBPF makes a system both transparent and secure. The eBPF events were reduced only under high load and the average latency was acceptable for our network and system monitoring use cases. This has led to a series of eBPF-based designs that cover a wide range of use cases, including next-generation networks, observability, and security functions. Network administrators use network monitoring systems to quickly detect problems such as device and link failures and traffic bottlenecks that impede data flow. These systems can alert administrators to problems via e-mail or text messages and generate network analysis reports.

Network monitoring solutions monitor network equipment and traffic and alert users to performance anomalies. By analyzing network traffic and monitoring overall performance metrics, companies can gain a complete picture of network performance. Network monitoring tools are moving beyond focusing on network performance and adding network security capabilities. Most network monitoring tools are designed not only to monitor network performance, but also to display performance metrics and generate insightful reports.

Sometimes the greatest degradation in network performance can be attributed to staff using too many valuable network resources. Users may also use the network for other purposes, such as streaming video or VoIP calls. At some point, network devices will need to be reconfigured to improve network performance, upgrade old devices or fix faulty hardware. The successful implementation of a network monitoring solution begins with identifying the specific situations in which the solution will be used. This allows users to clearly define the functionality required for the solution. It also helps the vendor’s network monitoring and implementation team understand the expected results of the implementation.

4.3 Paper 3: Networking

In ”Leveraging ebpf for programmable network functions with IPv6 segment routing” [10] three different use cases have been developed by the authors, namely network discovery, hybrid networks, and delay measurements. They also point out that there is no performance degradation or significant overhead in the network through the eBPF Linux router. On the contrary, the results are significant and even nearly acceptable.

The authors note that eBPF functions tend to have the lowest overhead compared to functions developed from its various static variants. The most important feature of the eBPF is that it is relatively versatile. The proposed framework opens new possibilities for researchers and network operators to implement various functions in their networks. In the near future, hybrid networks will be the infrastructure model that defines most multi-site enterprises. Today, Cloud/Analytics-as-a-Service is used on a wide variety of devices, including PCs, smartphones, tablets, and IoT devices, and its use cases are becoming more diverse. This diversity is increasing overall traffic, causing network congestion and negatively impacting business.

Advanced use cases explore data from primary data sources (outside ONAP) to improve data quality - data quality - information complexity. Objective is to provide building blocks to enable data discovery from primary data sources (outside ONAP) - Network sources - Overlay between core and data centers SDC can be used to discover specific network sources using self-service input models.

4.4 Paper 4: Tracing and Profiling

In the paper ”eBPF-based Content and Computation-aware” [1] we find how we can use content and process arranged correspondence for constant edge calculation, in light of eBPF. At the correspondence level, Software Defined Radio (SDR) is utilized to apply various boundaries and parts of radio transmission.

The network could be used to get the tracing facility in the Android tool as it is helpful to save the device activity and store it in a trace file. At the time of running device, the files are saved in Perfetto format. It use systematic format that could be accessed. The profiling is helpful to present the summary of statistics information and its required to perform the data that is equally important to trace in the organization. The both operations are commonly used by the organizations to get the high performance computing. The system performs different functions related to store the data.

Notwithstanding, a significant test and issue in appropriately using the accessible transfer speed for different constant applications is dealing with the traffic produced by these administrations and applications. The fundamental commitment of this examination is the plan, execution, and testing of an open source programmable structure for overseeing constant edge processing correspondences utilizing an installed eBPF bit. Network function virtualization infrastructure is an important starting point for the development of distributed cloud computing. Today, businesses often have independent virtualization environments in multiple locations. The ability to manage resources and sites in a coordinated manner is a step forward. This allows us to establish policies and restrictions on virtual network functions deployment.

5 Conclusion

To put everything into perspective, our paper explained the use of eBPF in different situations and classified the papers referenced. By extending the basic capabilities to see and interpret all system calls and provide packet-level and socket-level visibility into all network operations, an innovative approach to system security can be developed. eBPF is a Linux kernel feature that allows virtual machines to run within the kernel. This virtual machine can be used to securely load programs into the kernel and customize their behavior.

Traditionally, it has been difficult to make changes to the kernel; we would have to call APIs to get data, but we could not affect the contents of the kernel or execute code. Instead, we had to submit our modifications to the Linux community and wait for them to be approved. eBPF allows us to load a program into the kernel and ask the kernel to execute the program when some event occurs, for example, when we see a certain package.

Not only did eBPF solve these problems, it is also available indirectly through projects such as bpftrace and Cilium [7]. These projects provide an abstraction of eBPF, so there is no need to write programs directly. We can specify an intent-based definition and the eBPF will implement it. Instead of virtual IP, eBPF can use programs in the kernel to load balances at the source. Since there is no need for Destination Network Address Translation (DNAT) in the packet processing path, all Network Address Translation (NAT) overhead on service connections can be eliminated.

References

[1] S. Baidya, Y. Chen, and M. Levorato. ebpf-based content and computation-aware communication for real-time edge computing. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 865–870, 2018.

[2] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, and M. Zuppelli. Kernel-level tracing for detecting stegomalware and covert channels in linux environments. *Computer Networks*, 191:108010, 2021.

[3] L. Deri, S. Sabella, and S. Mainardi. Combining system visibility and security using ebpf. In *ITASEC*, 2019.

[4] W. Findlay, A. Somayaji, and D. Barrera. Bpfbox: Simple precise process confinement with ebpf. CCSW'20, page 91–103, New York, NY, USA, 2020. Association for Computing Machinery.

[5] D. Koelewijn. Fingerprint-based automated rule generation for ddos mitigation using the berkeley packet filter, February 2019.

[6] J. Levin and T. A. Benson. Viperprobe: Rethinking microservice observability with ebpf. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pages 1–8, 2020.

[7] Makowski and P. Grosso. Evaluation of virtualization and traffic filtering methods for container networks. *Future Generation Computer Systems*, 93, 09 2018.

[8] S. McCanne and V. Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, page 2, USA, 1993. USENIX Association.

[9] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. 53(1), feb 2020.

[10] M. Xhonneux, F. Duchene, and O. Bonaventure. Leveraging ebpf for programmable network functions with ipv6 segment routing. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 67–72, New York, NY, USA, 2018. Association for Computing Machinery.

Is it not yet Time to Swish?
Comparing the ReLU and Swish Activation Functions

Gerrit Luimstra and Willard Verschoore

Abstract—The choice of an activation function in a neural network has a significant effect on the training dynamics and the task performance. For this reason, many activation functions have been introduced with the goal of finding an activation function that works well across learning tasks. Currently, the most versatile and performant activation function is the widely-used Rectified Linear Unit (ReLU). Recently, however, a new activation function called Swish has been introduced that has shown improved performance over a variety of tasks.
In this paper we compare the ReLU and Swish-β activation functions by means of a literature study as well as our own experiments with a shallow neural network on three small datasets. We find that the literature is inconclusive and the choice of activation functions heavily depends on factors such as the data, the model, and hyperparameters. Our own experiments on the three datasets are in line with the inconclusivity of the literature. We show that there is no discernible difference of performance and convergence between the ReLU and Swish activation functions for a shallow neural network architecture on the datasets in question.

Index Terms—Swish, Swish-β, ReLU, activation function, machine learning, neural networks.

1 Introduction

The choice of an activation function in a neural network is said to have a significant effect on the training dynamics and the task performance [19]. For this reason, many activation functions have been introduced with the goal of finding an activation function that works well across learning tasks. According to the literature, one of the most versatile and performant activation functions is the widely used Rectified Linear Unit (ReLU). Recently, however, a new activation function called Swish has been introduced that has shown improved performance over a variety of tasks [20].

However, by studying other experiments in the literature we have found that the right choice of an activation function is rather task dependent. For example, in [6] the authors compare activation functions across a set of diverse natural language processing (NLP) tasks including sentence classification, document classification and sequence tagging. The findings indicate that there is no clear winner that works well across all tasks. In fact, the largest difference between the worst and best performing activation function is usually only one or two percentage points of the accuracy, suggesting that if a sensible activation function is used, the performance differences are minute.

The experiments from the literature conducted so far only feature large scale datasets, which means that the performance for small datasets is not yet known. Small datasets also allow for the use of shallower networks with fewer hyperparameters. Compared to the complicated deep networks required for large datasets, this greatly simplifies the experiments and enables more detailed analysis.

Aside from empirical evaluations, activation functions are often also compared based on their mathematical properties. For many of these properties there are convincing intuitive arguments for their importance. However, in all but a few cases there is a significant lack of any mathematically rigorous justifications. Aside from this, some of the commonly valued properties are contradictory and do not consistently lead to improved results.

The insignificant differences in experimental results and inconsistencies in the common mathematical properties allow us to suggest that the best choice of activation functions is highly situational. Some mathematical properties can be desirable and some activation function can outperform others, but which properties and activation functions depends on factors such as the network architecture, dataset, and choice of hyperparameters.

In Section 2 we start with a short discussion of activation functions, focusing on ReLU and Swish. We then discuss some of the mathematical properties of activation functions that are considered to be important. We conclude Section 2 with an overview of instances from the literature where ReLU and Swish are compared experimentally. In Section 3 we describe our own experiments and datasets used to determine if the Swish activation function is preferred over ReLU. These experiments are focused on a shallow neural network and small datasets. In Section 4 the results of these experiments are presented and discussed. Finally, in Section 5 we present our conclusions.

2 Background

Activation functions are a crucial component of a neural network because they allow the network to learn non-linear mappings instead of being restricted to linear ones. Concretely, an activation function of a node defines the output of that node given an input or set of inputs.

Historically, the choice of activation function was inspired by the activation pattern of biological neurons. This led to the widespread use of the *sigmoid* and *tanh* activation functions (see Figure 1). Recently, these functions have fallen out use due to their poor achieved accuracy and convergence behaviour.

In modern applications the most widely used activation function is likely the Rectified Linear Unit (ReLU) [9, 16]. ReLU is defined by the simple mapping $x \mapsto \max(0,x)$. Recent years have seen a large number of new activation functions introduced, many of them based in some way on ReLU.

One recent activation function that has seen some success is Swish [20]. Swish was found using an automated search and is defined by $x \mapsto x \cdot \sigma(\beta x)$, where σ is the *sigmoid* function

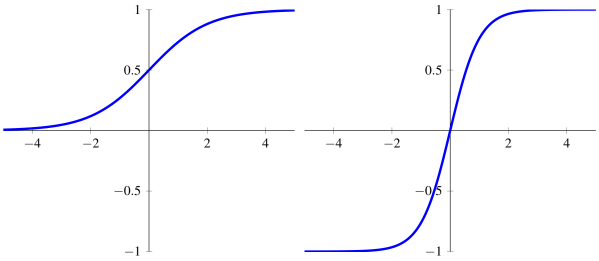
$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

The default setting of $\beta = 1$ recovers the preexisting SiLU activation function [7]. When $\beta = 0$, Swish becomes the linear function $f(x) = x/2$, and as $\beta \to \infty$ Swish approaches ReLU. For this reason, it can be said that Swish smoothly interpolates between a linear function and ReLU. Figure 2 shows a graph of both ReLU and Swish.

2.1 Mathematical Properties

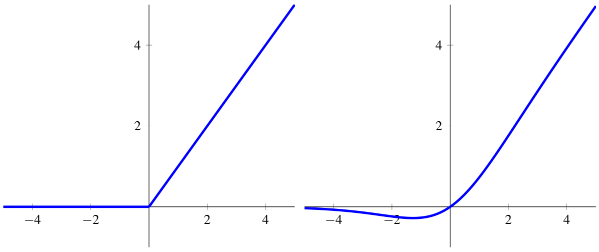
Recent years have seen an explosion in the number of activation functions proposed. The inventors tend to base their new activation function on a preexisting activation function that they modify to have certain desirable mathematical properties. There are quite a few such

- *Gerrit Luimstra is with University of Groningen, E-mail: g.s.luimstra@student.rug.nl.*
- *Willard Verschoore is with University of Groningen, E-mail: w.a.verschoore.de.la.houssaije@student.rug.nl.*



(a) Sigmoid. **(b)** Tanh.

Fig. 1: The *sigmoid* and *tanh* activation functions.



(a) ReLU. **(b)** Swish ($\beta = 1$).

Fig. 2: The ReLU and Swish activation functions.

properties that are commonly believed to be beneficial, yet these beliefs are rarely backed up by mathematical proof. In the following section we cover some of the most important of these mathematical properties. We will discuss the reasoning behind them and see that some properties may not be as important as previously thought.

2.1.1 Non-linearity

Arguably the most important property is non-linearity. This property is required to allow the neural network to learn non-linear mappings of the input. If non-linearity is not present in the activation functions of a neural network, the whole neural network can be collapsed into a single matrix, i.e., a linear mapping. Since datasets are rarely generated from a purely linear process, this is an important property.

Beyond simple non-linearity, some non-linear activation functions such as *sigmoid*, *tanh*, and ReLU have been proven to allow for the construction of universal function approximators [3, 11, 21]. This means that a network with these activation functions can, given the right network architecture, approximate any function to arbitrary precision.

2.1.2 Differentiability

Another important property of activation functions is that they must be differentiable. Neural networks are generally trained using gradient based techniques, which means that it must be possible to find the gradient of an activation function for each input value. Interestingly, ReLU breaks this requirement by being non-differentiable at the origin. Theoretically this may seem problematic, but in practice this issue does not noticeably affect performance since the chance of an input being exactly equal to zero is sufficiently low.

A stronger version of differentiability is smoothness. A function is smooth if it is continuous and any number of repeated differentiations also result in a continuous function. Swish is an example of a smooth activation function. In their paper [20], the inventors of Swish argue that this smoothness leads to a smooth loss-landscape and thus easier optimization using gradient descent.

2.1.3 Unboundedness

One of the main reasons that researchers have moved away from the *sigmoid* and *tanh* functions is because they are bounded. Both functions approach a lower and upper limit where they level out. This saturation of the activation function means that the gradient vanishes for low and high inputs. A small gradient causes slow convergence and is especially problematic in deep networks where backpropagation exacerbates the problem.

The reason *sigmoid* and *tanh* are bounded is because they are based on the activations of biological neurons and these are known to saturate. Interestingly, in an early paper discussing ReLU [9], Glorot *et al.* justify the unboundedness of ReLU not only on the basis of the performance gained by avoiding the vanishing gradients problem, but also because it is more biologically accurate. The reason they give for this is that “The neuroscience literature indicates that cortical neurons are rarely in their maximum saturation regime”. This means that, due to its unbounded nature, ReLU could be considered a more accurate biological analogue than saturating functions such as *sigmoid* and *tanh*.

The astute reader will have noticed that while ReLU is indeed unbounded from above, it is still bounded from below. Many other popular activation functions such as Swish and Softplus are also unbounded from above, but bounded from below. More specifically, the output of these activation functions approaches zero as the input gets lower. The reason for this is that disabling the output of an activation on some range of input values is a form of regularization that promotes sparsity. This means that activation functions like ReLU naturally give rise sparse networks, which is both mathematically desirable and a known property of biological neural networks.

While being bounded from below indeed has the advantage of sparsity, that does not mean that the problem of vanishing gradients is entirely avoided. For this reason, activation functions such as leaky ReLU (LReLU), which has a small positive derivative left of the origin, were introduced as an alternative to ReLU. While the paper introducing LReLU found no significant improvement over ReLU [14], other papers since have found that some variants consistently outperform ReLU [2, 23]. In a survey by Eger *et al.*, the authors find that saturating functions perform more stably across a variety of hyperparameter settings, but unbounded functions have better best case performance [6].

2.1.4 Monotonicity

A seemingly reasonable requirement for an activation function is that it must be monotonically increasing. This means simply that if we have two input values where the first is smaller than the second, then the output of the activation function for the first input will be smaller than or equal to the output for the second input. This matches the intuition obtained from biological neurons and by far the majority of activation functions satisfy this property.

However, in Figure 2b we clearly see that Swish has a “dip”. The creators of Swish believe that this non-monotonicity is part of the reason for Swish’s success. They claim that it “increases expressivity and improves gradient flow” [20].

Swish increases expressivity in the sense that it does not crush all negative inputs like ReLU does. The “dip” allows for the preservation of small negative inputs. Ramachandran *et al.* show in their experiments that such inputs occur frequently, so the behaviour of Swish in this region is likely to be an important factor in its success [20]. In order for an activation function to still have the regularization properties discussed in Section 2.1.3, the activation should approach zero for large negative inputs. Combining this property with the preservation of small negative inputs naturally leads to a non-monotonic function.

2.1.5 Zero-centring

The final property we will discuss is that of zero-centring. An activation function is zero-centred if its average activation is approximately zero. When this is not the case, activation functions impose a *bias shift* away from zero. If the units in a layer are correlated, this shift increases for each hidden layer. By centring an activation function at zero this bias is avoided, which leads to faster learning. This is described in detail in the paper introducing the zero-centred Exponential Linear Unit (ELU) [2].

The authors of [13], in which the Scaled Exponential Unit (SELU) is introduced, describe a stronger variant of this property which they call "self-normalization". An activation function is self-normalizing if it has zero mean and unit variance. When an activation function has this property it leads to automatically normalized outputs, obviating the need for regularization procedures such as batch normalization. Notably, the unit variance property requires activation functions to saturate in some regions in order to dampen the variance. The authors of the SELU paper argue that, despite the saturation, vanishing gradients are not a large problem because self-normalizing functions are more robust to perturbations caused by stochastic training procedures.

2.2 Comparing Activation Functions

In their papers introducing Swish [19, 20], Ramachandran *et al.* compare Swish against other activation functions in a number of image classification and machine translation tasks. The most notable discovery is that Swish outperforms ReLU in all experiments. The difference in accuracy can be as high as 2.2%, but tends to be anywhere from 0.5% to 1%. They note that Swish outperformed ReLU by a large margin in very deep networks. This is especially interesting since Swish has a non-unit derivative around the origin, which would conventionally be considered problematic in deep networks due to vanishing gradients. It is important to note that in the experiments the authors used models and hyperparameters designed for ReLU; Swish was simply substituted in its place. The authors speculate that if Swish is widely adopted and models are constructed with it in mind, then the improvements over ReLU may be even greater.

While these results seem to clearly indicate that Swish is the superior activation function, they should be interpreted with care. This is demonstrated by Eger *et al.* who obtain more nuanced results in their comparison of 21 different activation functions across a variety of NLP tasks [6]. Their experiments show that Swish and ReLU perform very similarly with the difference in accuracy often being less than 0.5%. They note that both Swish and ReLU have good best case performance, but may need more hyperparameter tweaking than other, more stable activation functions.

Other comparisons find similarly ambiguous results. For example, Xu *et al.* find no significant difference between popular activation functions in their experiments [24]. Dubey *et al.* show that ReLU outperforms Swish for most models on the same image classification datasets as used by Ramachandran *et al.* [5]. The differences in accuracy were often more than one standard deviation and training for Swish was also significantly slower. Interestingly, for a machine translation experiment the same paper shows Swish outperforming ReLU. A study on activation functions in generalized learning vector quantization found that Swish achieved slightly better accuracy and significantly faster convergence when compared to ReLU [22].

These examples clearly demonstrate that the literature is inconclusive when it comes to the choice of activation function. One issue is that the differences in accuracy that are found tend to be no more than 1%. In their papers introducing Swish [19, 20], Ramachandran *et al.* argue that these differences are significant since one year of architectural improvements going from Inception V3 to Inception-ResNet-v2 led to a similar increase in accuracy. However, as discussed above, other experiments showed similar differences in accuracy in ReLU's favour.

In our opinion, the main issue with comparing activation functions is that performance is highly dependent on a wide variety of factors. The choice of model and hyperparameters significantly affect performance and the best choice differs per activation function. In the case of Swish, there is the additional parameter β which can also be optimized. All of this means that, while activation functions may have a noticeable effect on a network's performance, they should be chosen as part of a broader model selection procedure where they are compared against others for each separate task. There is sadly no universally superior activation function, but both ReLU and Swish are good options in many situations.

3 Methodology

In order to avoid complications of hyperparameter selection we observed in previous experiments, we chose to use a simple neural networks architecture in our experiments. The choice of a simple architecture naturally leads to the use of small datasets. Hence, to empirically compare the performance of both the ReLU and the Swish activation function, we train a simple neural network on three small multivariate regression datasets from the UCI Machine Learning Repository [4]. This should allow us to isolate the effect of the choice of activation function as much as possible.

All experiments have been conducted in the Python programming language in combination with the open source machine learning framework PyTorch.

3.1 Datasets

In this section we outline the three small datasets used in our experiments.

3.1.1 YearPredictionMSD

The first dataset is the YearPredictionMSD dataset [1] in which the goal is to predict the release year of a song based on its audio features. The songs are mostly western and commercial tracks released in the years 1922 to 2011. The dataset has a total of 515,345 samples with 90 attributes. The attributes contain 12 timbre averages and 78 timbre covariances, where timbre is the perceived sound quality of a musical note, sound or tone. For a more detailed explanation of the features we refer to [1].

3.1.2 Abalone

Abalones are a group of marine snails. The age of abalone can be found by cutting the abalone shell through the cone, staining it, and counting the number of rings through a microscope. In general, this is a laborious and monotonous task. Instead, a machine learning based method can be designed that uses easily measureable properties of the shell to predict the age and thereby preventing the cutting of the shell. With the Abalone dataset [17] the goal is to predict the age of abalone shells based on a set of measurements including the diameter, height, weight and the number of rings on the shell. The dataset has 4,177 samples and 8 dimensions.

3.1.3 Bike Sharing Dataset

The goal of the Bike Sharing dataset [8] is to predict the number of bikes that will be rented daily based on the environmental and seasonal circumstances of that day. After the removal of some superfluous columns, the modified dataset contains 731 samples with 11 dimensions. The features (dimensions) in the dataset consists of date/time information, such as the season, year, month and whether the day is a weekend or working day and environmental information such as the weather situation, temperature, windspeed and humidity. Using these features, the total number of bikes rented is to be predicted. The scale of the output variable is in the thousands.

3.2 Data Preprocessing

To aid with the convergence speed of the neural network, all the input data is scaled to have a mean of 0 and a standard deviation of 1. To do this, we perform for each feature j the transformation:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j},$$

where x_{ij} denotes the jth feature of the ith sample and μ_j and σ_j are the mean and standard deviations of feature j over the training data. The inputs x_{ij} are then replaced with the $z-$scores z_{ij}.

3.3 Network Architecture

Since the experiments focus on the use of small datasets, the neural network architecture need not be complex. Hence, we opted for an elementary feed-forward architecture with a single hidden layer with K

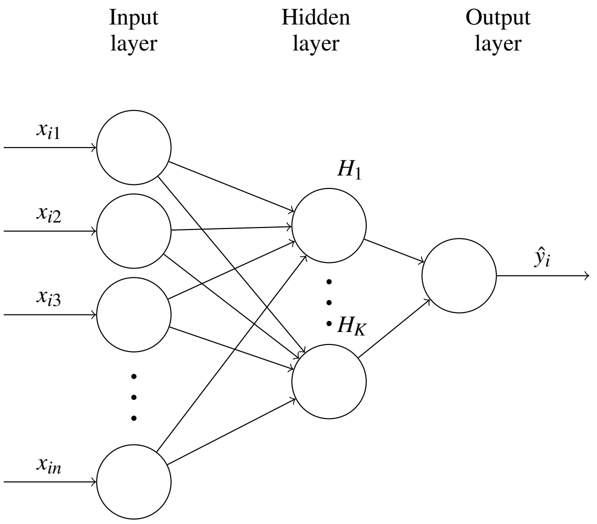


Fig. 3: The neural network architecture used in the conducted experiments. It consists of a single hidden layer with K units with as the activation function one of the considered activation functions as outlined in the experiments. The single output layer has the identity function as the activation function. All the weights and biases are adaptable.

units (with bias) and a single output with the identity activation function as displayed in Figure 3.

For the activation function on the hidden layer, the ReLU function or the Swish-β function will be used with $\beta = 0$ for a linear function, $\beta = 1$ for the standard Swish function, and $\beta = 0.5$ as an interesting value which can be interpreted as an interpolation between a linear activation function and Swish. Depending on the performance of the different activation functions, we can empirically conclude which activation is best. All the weights and biases in the network are adaptable, including the hidden to output weights.

3.4 Training and Evaluation

The neural network weights and biases are initialized using He initialization [10], since this works well for rectifier type activation functions. Furthermore, the trained weights are obtained using the adaptive moment estimation (ADAM) optimizer [12]. In contrast to vanilla gradient descent, ADAM maintains a per-parameter learning rate that improves performance on problems with sparse gradients. On the YearPredictionMSD dataset an initial learning rate of 0.1 is used, for the Abalone dataset the initial learning rate is set to 0.002, and for the Bike Sharing dataset the learning rate is set to 0.1. For each dataset, the model is trained for 2000 epochs. These settings were experimentally found to be suitable.

The cost function that is optimized is the mean-squared-error loss:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where y_i and $\hat{y}_i$ are the actual and predicted outputs respectively. In this work we will however report the square root of this loss function, called the root-mean-square error (RMSE) since this is on the scale of the predictor itself and allows for a more intuitive interpretation.

To obtain the performance of the different parameter settings, 5-fold cross validation is used on the whole dataset, where the performance of an iteration is set to the final loss of the model on the hold-out set.

4 Results & Discussion

In this section we present and discuss the results of our experiments.

4.1 Performance Reports for Different K and Activation Functions

The RMSE scores of different K and activation functions on our selected datasets are displayed in Figure 4.

In Figure 4a, the performance on the YearPredictionMSD dataset is displayed. For the case of $K = 1$, the data points for $\beta = 0.5$ and $\beta = 1$ are excluded, as the RMSE for these data points were 357 and 684 respectively. From the plot it is clear that there are no clear differences between the performance of the activation functions when using more than one hidden unit, however in some cases ($K = 9$) the linear activation function is slightly worse. Interestingly, a model with just 2 hidden units ($K = 2$) obtains a decent RMSE already.

The performance on the Abalone dataset is displayed in Figure 4b. From the plot it is clear that for a good RMSE at least 6 hidden units are required. For $K \geq 6$, the difference in activation function performance is negligible. For the cases of $K < 6$, the network is too constrained and is not able to obtain a proper solution. Interestingly, in this region the Swish-like activation functions perform significantly worse than the linear activation function. The reason for this is not clear to us, but it could be explained by the derivative being present everywhere for the activation function. In Figure 4c, the performance plots of the Bike Rental dataset are displayed. We see behaviour similar to that in Figure 4b, where for $K \geq 4$ the model is again too constrained. For $K < 4$ the linear activation function is yet again the best performing. It should be noted that even the best performance on the Bike Rental dataset is not spectacular however. The obtained RMSE of about a 800 indicates that the model can predict the amount of bikes rented with on average an error of about 800 bikes. Hence, these results should not be over-interpreted.

In general there does not appear to be a clear winner when it comes to the activation functions considered. The differences between all the activation functions are small for a suitable number of hidden units.

4.2 Convergence Behaviour of ReLU vs Swish

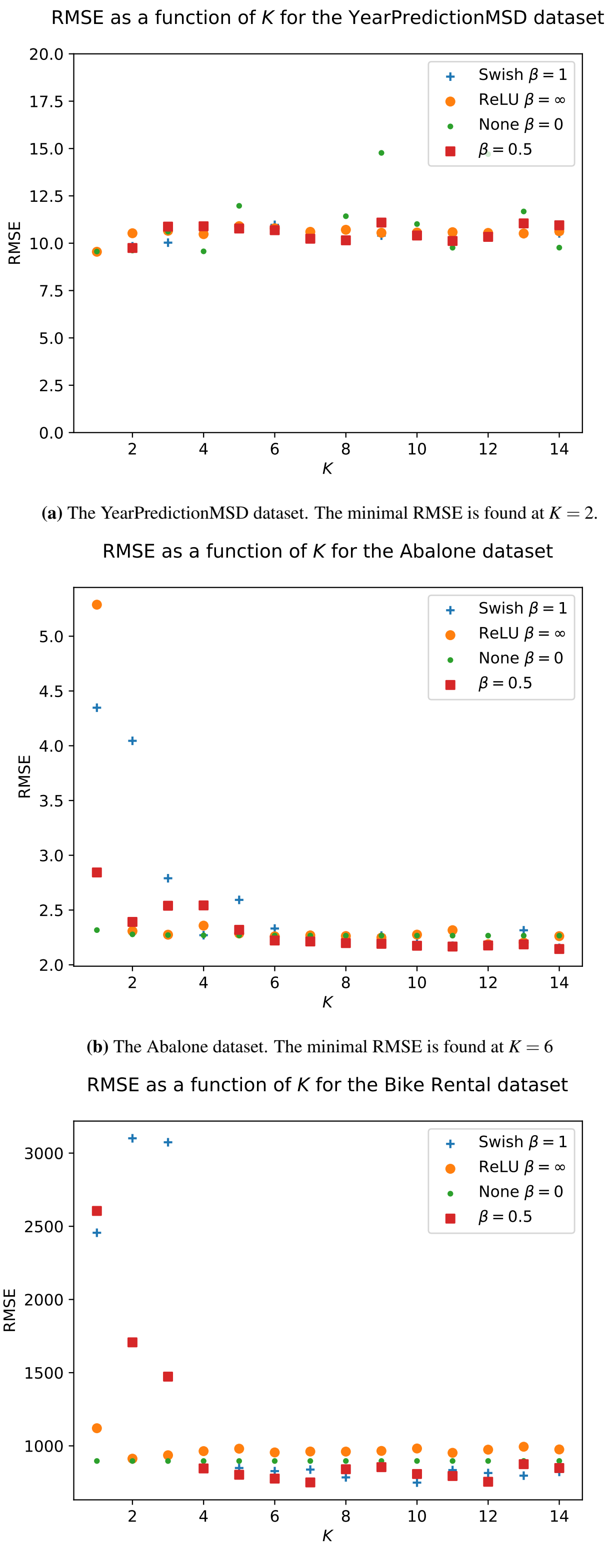
As described before, the authors of [20] believe that the use of the Swish activation function leads to a smooth loss-landscape and thus easier optimization using gradient descent. If this is true, the convergence of the training RMSE should be faster for the Swish activation function on the three datasets considered. To determine the convergence behaviour on each dataset, the optimal K is fixed and the training loss curve as a function of epochs is obtained by training on the whole dataset. From the previous experiments, the optimal number of hidden units K can be determined and is found to be 2, 6 and 4 for the datasets respectively by picking the smallest K such that the performance is close to minimal for all activation functions.

The plots of the training loss curves are displayed in Figure 5. From the plots it can be seen that the convergence behaviour for the different activation functions is very similar over the three datasets. In general, while the identity activation function appears to work slightly better on the first two datasets, there is no clear winner nor a clear loser when it comes to convergence speed.

4.3 Discussion of the Experimental Setup

Because of the simplicity of our network architecture we can consider our experiments to be quite thorough. There are not many other hyperparameter settings we could have used without extending the architecture, which would contradict our aim to keep down complexity in order to isolate the effects of the choice of activation function. It would have been possible to use more choices of β for the Swish activation function, but the results obtained show that it is unlikely this would have made a significant difference. Furthermore, the choice of the optimizer can have a non-trivial impact on the performance of the models. We chose to use the ADAM optimizer over the simpler vanilla gradient descent due to its popularity and its ability to use local gradients for different parameters which speed up the learning process.

While we have explained the reasons for performing experiments on small datasets, it can also be argued that ReLU and Swish are mainly intended for use in deep neural networks, and the comparisons in this



RMSE as a function of K for the YearPredictionMSD dataset

RMSE as a function of K for the Abalone dataset

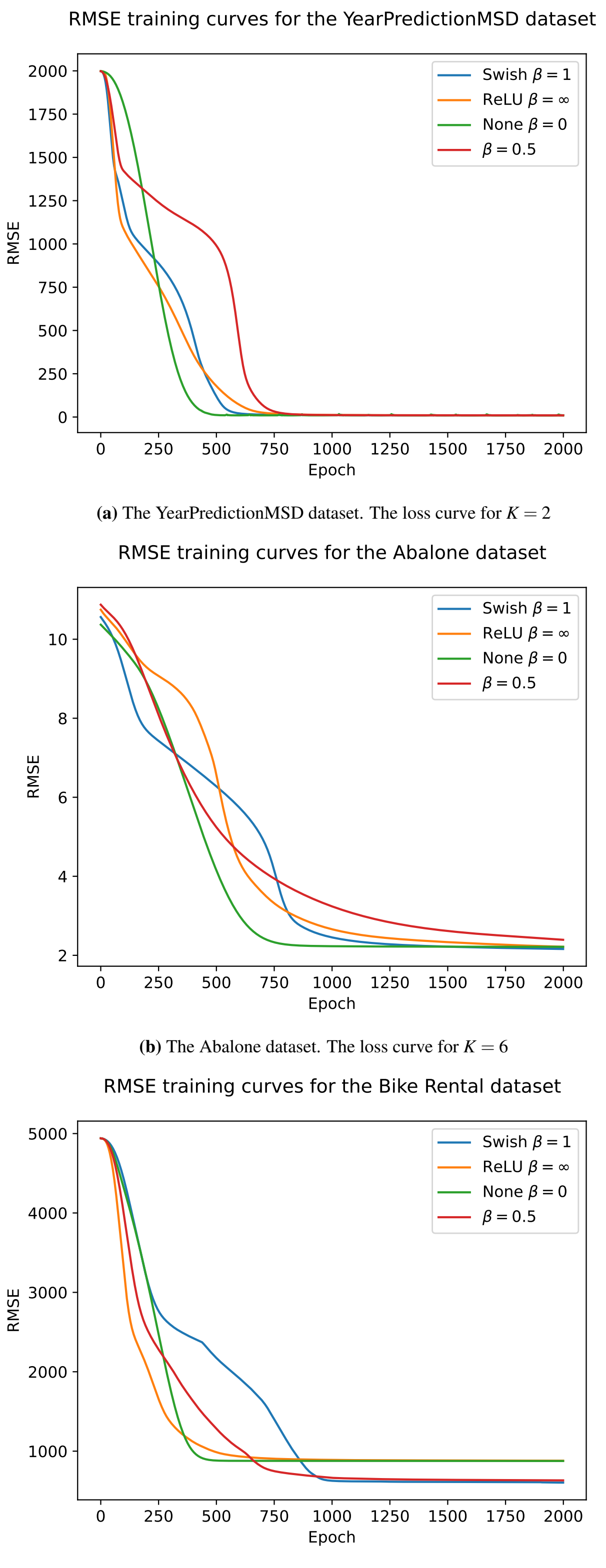
RMSE as a function of K for the Bike Rental dataset

(a) The YearPredictionMSD dataset. The minimal RMSE is found at $K = 2$.

(b) The Abalone dataset. The minimal RMSE is found at $K = 6$

(c) The Bike Rental dataset. The minimal RMSE is found at $K = 4$

Fig. 4: The RMSE performance plots for different K and β. The y-axis denotes the RMSE and the x-axis displays the number of hidden units K of our network architecture. Colors are used to differentiate between the activation functions considered.



RMSE training curves for the YearPredictionMSD dataset

RMSE training curves for the Abalone dataset

RMSE training curves for the Bike Rental dataset

(a) The YearPredictionMSD dataset. The loss curve for $K = 2$

(b) The Abalone dataset. The loss curve for $K = 6$

(c) The Bike Rental dataset. The loss curve for $K = 4$

Fig. 5: The RMSE performance as a function of the number of epochs. The y-axis denotes the RMSE and the x-axis displays the number of sweeps over the dataset. Colors are used to differentiate between the activation functions considered.

paper are thus somewhat irrelevant. Both activation functions mitigate the issue of vanishing gradients, which occurs with many layers, but this is largely unnecessary in the single hidden layer architecture of Figure 3. Because of this, our experiments cannot rule out the possibility that there is a significant difference between ReLU and Swish for more complicated networks where their unique properties may have more of an effect. However, we discussed in Section 2.2 how experiments with deeper networks found no consistent differences either.

5 Conclusion

A survey of the literature has shown that there is no obvious optimal choice of activation function. Between ReLU and Swish, neither has a clear advantage over the other. Even when restricted to a particular task or dataset, both functions can come out on top depending on factors such as the model architecture and the chosen hyperparameters. When a difference in achieved performance is observed, it is rarely discernible and arguments for the significance of such a difference are not always convincing.

In our own experiments on the three small datasets, we measured the performance and convergence behaviour of the ReLU and the Swish activation function (and variants thereof). From the results of these experiments, it is found that there is no clear winner or loser when it comes to the choice of activation function.

Conclusively, since there is not yet a clear mathematical framework present to study the complex learning behaviour of activation functions on general datasets and architectures one should turn to approximate and empirical results with concrete datasets and architectures. Our experiments suggest that on the considered datasets and for our specific architecture, there is no discernible difference between the ReLU and Swish activation functions in terms of performance and convergence speed.

6 Future Work

In future work, a few items may be investigated. There exists a body of research surrounding trainable activation functions in which the activation functions themselves have parameters that the optimizer can adapt and optimize. One such family of activation function is the P-Swish activation function defined in [15]. Using such an adaptable activation function removes the issue of having to manually set the amount of monoticity used in the activation function (through β), since it can be learned in a task agnostic way. By investigating the optimized parameters, a better conclusion can be obtained about the performance improvements of Swish versus ReLU.

Finally, the inconclusive nature of our experiments and those of others show that empirical evaluations are able to provide only limited insight into the differences between activation functions. For a more complete understanding, mathematical analyses should be conducted. Deep neural networks are hard to investigate in this way, but for shallow networks like those used in our experiments it may be possible. This is demonstrated by Oostwal *et al.* who compare ReLU and sigmoidal activation using statistical physics techniques [18]. Similar comparisons involving Swish may provide valuable insights.

References

[1] T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[2] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2016.

[3] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[4] D. Dua and C. Graff. UCI machine learning repository, 2017.

[5] S. Dubey, S. Singh, and B. Chaudhuri. A comprehensive survey and performance analysis of activation functions in deep learning. *arXiv preprint 2109.14545*, 2022.

[6] S. Eger, P. Youssef, and I. Gurevych. Is it time to swish? comparing deep learning activation functions across NLP tasks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4415–4424. Association for Computational Linguistics, 2018.

[7] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *arXiv preprint arXiv:1702.03118*, 2017.

[8] H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15, 2013.

[9] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[11] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[13] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[14] A. Maas, A. Hannun, and A. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[15] M. A. Mercioni and S. Holban. P-swish: Activation function with learnable parameters based on swish activation function in deep learning. In *2020 International Symposium on Electronics and Telecommunications (ISETC)*, pages 1–4, 2020.

[16] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[17] W. Nash, T. Sellers, S. Talbot, A. Cawthorn, and W. Ford. The population biology of abalone in tasmania. i. blacklip abalone from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report No*, 48, 01 1994.

[18] E. Oostwal, M. Straat, and M. Biehl. Hidden unit specialization in layered neural networks: Relu vs. sigmoidal activation. *Physica A: Statistical Mechanics and its Applications*, 564:125517, 2021.

[19] P. Ramachandran, B. Zoph, and Q. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941v2*, 2017.

[20] P. Ramachandran, B. Zoph, and Q. Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941v1*, 2017.

[21] S. Sonoda and N. Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017.

[22] T. Villmann, J. Ravichandran, A. Villmann, D. Nebel, and M. Kaden. Investigation of activation functions for generalized learning vector quantization. In *International Workshop on Self-Organizing Maps*, pages 179–188. Springer, 2019.

[23] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[24] L. Xu and C. Chen. Comparison and combination of activation functions in broad learning system. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3537–3542, 2020.

Capsules vs. Convolutional Neural Networks

Merlijn Frikken and Sander Zeeman

Abstract— Convolutional Neural Networks have been a staple in image classification for a long time. However, CNNs have a few well-known weak points. Specifically, these networks are fully position invariant. In extreme cases, this may lead to serious issues. Capsule Networks were proposed as an alternative. These networks store parameters such as location, orientation and size, while still being location invariant, in the sense that the location of an entity will not affect to probability of it being detected in the image. A literature study was performed on both methods and the benefits and shortcomings of either approach were analysed. We found that capsule networks achieve greater performance than conventional CNNs, while also allowing for large future improvements. For complicated data sets, the capsule networks have been shown to fail.

Index Terms—CNN, Capsule, Neural Network, Analysis, MNIST, LUNA, CIFAR.

1 Introduction

Convolutional Neural Networks (CNNs) are a type of Artificial Neural Network (ANN). LeCun et al. were one of the first to popularize CNNs in [8]. CNNs were proposed in order to ensure shift and distortion invariance in ANNs. These properties are extremely important, as the absolute position of a feature should not matter for a neural network. Relative position, the position of a feature with respect to other features, is very important however.

In regular feed-forward networks, the topology of an input signal is not taken into account. This means that the correlations between features are entirely ignored. Naturally, this causes issues when processing signals, as the neighborhood of a signal can contain a large amount of information towards the nature of this signal. CNNs were introduced to remedy this issue.

However, various issues were found within CNNs as well. This will be described in detail, later in this paper. However, the main issue that was found, was the exact fact that CNNs are position invariant. As long as all features that describe some object are present in a signal, the exact placement of these features becomes far less important. This may lead to problems, such as the classification that can be seen in Figure 1. In this figure, the CNN has been trained to perform face recognition. Both faces are classified as a face, due to the fact that CNNs are fully position invariant, as was mentioned before. All required features are present in the image, though not in the correct positions, relative to one another.

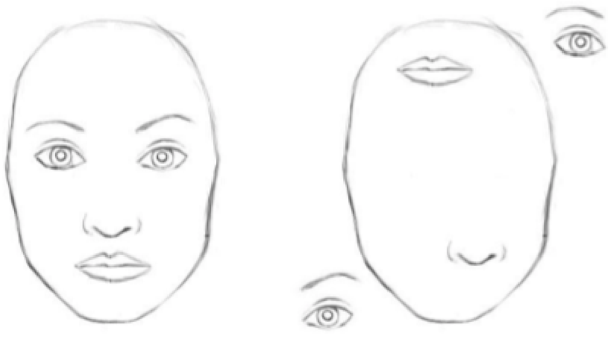


Fig. 1: Both images are classified as a face by a CNN [4]

Recently, an alternative was proposed by Hinton et al. in [13]. Hinton states that Capsule Networks attempt to replicate the human visual cortex. Capsule Networks attempt to extract entities from a signal. Unlike CNNs, the features that were extracted by a Capsule Network describe various attributes of an entity. Capsule Networks addressed the issue that was previously mentioned. Additionally, this alternative improved upon multiple other factors. These improvements will be described extensively in Section 4.

In this paper, we will compare state-of-the-art CNNs with the recently introduced Capsule Networks. The strengths, limitations and characteristics of both approaches will be analysed and compared based on a literature study. We conclude with a suggestion regarding the future use of Capsule Networks.

We will attempt to answer the following research question:

How do the newly introduced Capsule Networks compare to conventional Convolutional Neural Networks?

From an initial literature study, we believe Capsule Networks have the potential to become a powerful alternative to conventional CNNs. However, we also believe that Capsule Networks are unlikely to fully replace conventional CNNs in the near future.

In Section 2, the basic architecture of a CNN will be extensively described. Each layer will be defined mathematically, then its importance will be outlined. Afterwards, in Section 3, the same will be done for a Capsule Network. Furthermore, in Section 4, an outline will be given of the strengths and weaknesses of either approach. Additionally, various attributes, such as accuracy, computational load and training data requirements will be mentioned. This analysis will then be used to compare the two approaches in Section 5. Then, we will conclude this paper with Section 6, where we will once again outline the main findings and formulate an answer to our research question. Finally, we will mention some future work that could be done on the topic in Section 7.

2 Convolutional Neural Network

Neural networks are used to solve a large range of problems. Examples of common applications are: classification, regression, image segmentation and many more. Before the introduction of CNNs, large feed-forward networks were applied. While these networks were capable of producing adequate results, these networks would often repeat groups of weights at various positions within the signal. Feed-forward networks are not position invariant by nature after all. As a result, Convolutional Neural Networks were introduced.

CNNs were defined with the specific goal of ensuring position invariance within a feed-forward network. A typical CNN architecture is built from three types of layers: Convolutional layers, Pooling layers and Fully Connected layers. Here, the former two layers are used for feature extraction, while the latter layer is used for classification. A typical architecture is displayed in Figure 2. Many more layers have

- *Merlijn Frikken is with the University of Groningen, E-mail: M.Frikken@student.rug.nl.*
- *Sander Zeeman is with the University of Groningen, E-mail: s.zeeman@student.rug.nl.*

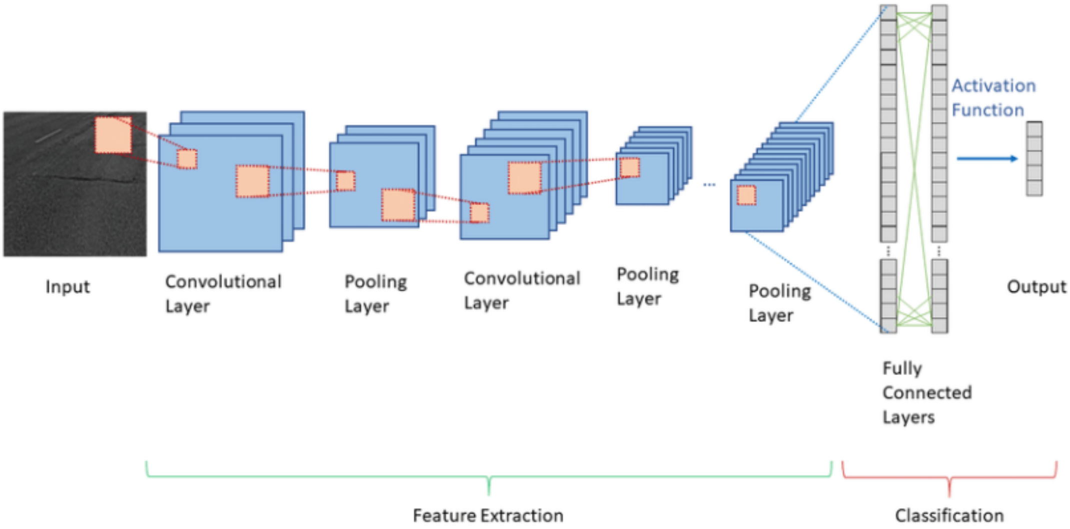


Fig. 2: A basic CNN architecture for a classification task [2]

been defined in the state of the art, however, these are beyond the purpose of this paper. Therefore, we will limit ourselves to the aforementioned three layers.

2.1 Convolutional Layer

In the convolutional layer, filters will act as a sliding window over the input image. A filter, in this context, is another name for a weight matrix. Generally,

$$\sum_{w \in W} w = 1, \text{ where } W \text{ is the filter} \tag{1}$$

holds, but this is not a strict requirement.

At every position, a convolution will be computed between the filter and the section of the image it currently overlaps. The convolution of some filter W and an input image I is defined as

$$(W * I)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} W(s,t) I(x-s, y-t). \tag{2}$$

Feature maps are the intermediate representations within a CNN. Generally, an activation function is applied to the result of the convolution, to produce the feature map. Because of this, the feature map is often referred to as an activation map as well. After all, this intermediate representation denotes the activation of various areas within the input. Many choices for the activation function can be found in the literature, but the most common is the ReLU activation function [12], which is defined as

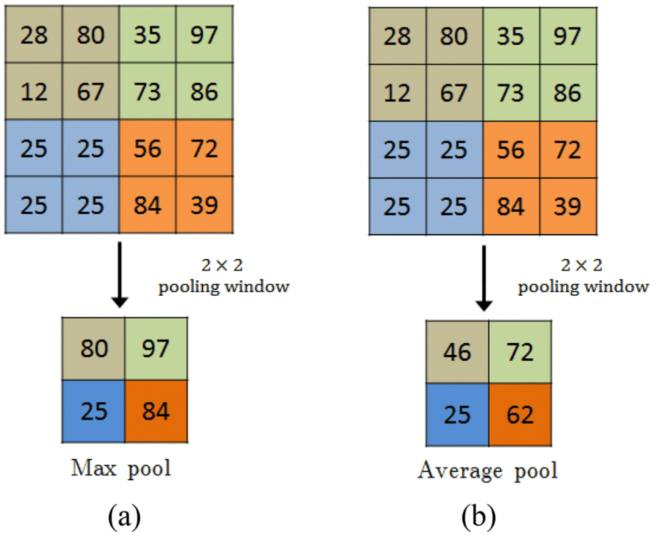
$$f(x) = \max\{0, x\}. \tag{3}$$

Note that the convolution of two matrices is similar to the correlation between the two matrices, with one of the matrices being mirrored horizontally.

Additional parameters can be set to alter the size of the output. Firstly, multiple filters can be used on a single image, which means we have to set the number of filters: K. Next, we can set the stride: S, which defines the number of pixels we slide over after every step. Additionally, we can add a padding: P to our input. This will add P rows/columns of zeros at the start and end of the input. Finally, a filter will have size FxF. In this definition, the filters are assumed to be square matrices, however, this is once again not a strict requirement. When we want the output feature maps to have the same dimensions as our input, the settings are generally $S = 1, P = \frac{F-1}{2}$.

2.2 Pooling Layer

In the pooling layer, the goal is to make the feature maps smaller and more manageable. Pooling is also referred to as down-sampling in the literature. Two main types of pooling are used in CNNs, once again, many more exist, yet this is beyond the scope of this paper [14]. Pooling works by dividing the feature map into smaller areas of size SxS. For each of these areas, some statistic will be computed.



28	80	35	97
12	67	73	86
25	25	56	72
25	25	84	39

2 × 2 pooling window

80	97
25	84

Max pool

(a)

28	80	35	97
12	67	73	86
25	25	56	72
25	25	84	39

2 × 2 pooling window

46	72
25	62

Average pool

(b)

Fig. 3: An example of max/average pooling [1]

This statistic will be the new value that is entered into the output feature map. An example of max/average pooling is included in Figure 3, in order to visualise the process. First, we discuss average pooling. As the name suggests, this method of pooling computes the average of an area. This can often be used to retain information about the general area, while allowing the network to forget specific details. Alternatively, max pooling can be applied. Max pooling, once again, acts according to its name. This method takes the maximum value within some area and assigns this value to this spot in the feature map. This method will retain any sudden peaks in a feature map, which could help us find critical areas within the image.

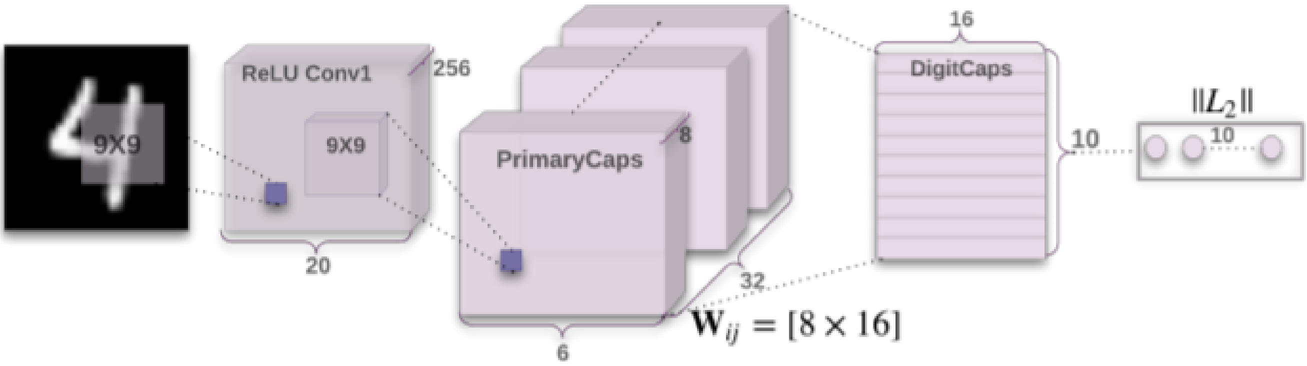


Fig. 4: A basic Capsule Network architecture [13]

2.3 Fully Connected Layer

Finally, we discuss the fully connected layer. This layer contains a number of neurons. Here, every pixel of every feature map is connected to every neuron, hence the name "Fully connected". These layers are added to the end of a CNN in order to introduce non-linearity; convolution and average/max pooling are linear operations. The problems that call for a machine learning solution are often non-linear in nature. Therefore, it is very beneficial to allow for non-linearity within a model. In the end, a linear solution could propose a viable approximation of the non-linear solution. However, a non-linear solution is magnitudes more likely to approach a correct solution.

Finally, these neurons will perform their activation functions and return an output. This output generally consists of the probabilities that the input contains each of the defined classes.

In conclusion, the convolutional layers are responsible for extracting features from various locations in the input image. The pooling layers are responsible for decreasing the size of our feature maps, while retaining the important features that were extracted. Finally, the fully connected layers are responsible for combining the previously extracted features into a solution.

3 Capsule Networks

As mentioned before, one of the main issues with CNNs is that the information regarding the position of various segments is lost by the end of the network. Instead, Hinton et al. [13] propose an alternative. Capsule Networks consist of a collection of capsules. Capsules are a novel data structure, which will contain various neurons. These neurons will store various parameters of the entity that it represents; the instantiation parameters. Meanwhile, retaining the ability to determine the probability that an object appears in an image. This method will define various capsules, which are later linked together in order to represent objects.

First, we define the concept of a capsule, similarly to Hinton et al. [13]. Simply put, a capsule is a collection of neurons, which will represent some entity within a signal. Neurons take as input a list of scalar values and outputs a scalar value. A capsule is simply a vector of neurons. Therefore, a capsule will take as input a vector of vectors, i.e. a matrix. Then, the output will become a vector containing the instantiation parameters, as previously defined.

In Section 2, we described the output of a CNN as "the probabilities that the input contains each of the defined classes". In Capsule Networks, the output of neurons will consist of a vector of instantiation parameters, rather than a probability. For example, the first value in the vector may encode the entity's size, while the second and third encode the location. Therefore, another method of determining the probability of the existence of an entity is required. As the various values within the vector encode properties of the entity, we can simply use the length of this vector as the probability that the entity it represents exists in the image. After all, whenever many parameters of an entity are strongly noticeable in an image, it is very likely the entity is included in the image.

Whenever some feature changes its position or size in an image, the probability that this feature exists in the image should not change. The instantiation parameters will change, as this is where these attributes are stored. However, its contribution towards the length of the vector should not change. After all, the aim of Capsule Networks was to obtain absolute position invariance, while considering relative positions of objects.

A typical architecture for a Capsule Network can be seen in Figure 4. We can see various layers in this architecture. First, we see a convolution layer, as previously described in Section 2.1. Afterwards, Primary Capsules are applied.

3.1 Primary Capsules

As written in [4], primary capsules act as a transition between the scalar values that are produced by the prior convolution to an eight-dimensional vector output. Thus, these capsules will perform an affine transformation of an input vector $\boldsymbol{u}_i$ with some weight vector $\boldsymbol{W}$. This transformation can be described as

$$\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\boldsymbol{u}_i. \tag{4}$$

First, we group eight convolutional units from the previous layer into one capsule. Then, we simply perform a convolution, as described in Section 2.1. However, as we are using vectors, rather than scalar values, we can no longer use the ReLU function. The ReLU function is only defined for scalars after all. Instead, we will apply the novel activation function by Hinton et al. [13]; the squashing function.

3.1.1 The squashing function

We mentioned previously that the length of an output vector will represent the probability that the entity it represents exists within an image. However, currently there is no boundary on the length of an output. To remedy this, Hinton et al. introduced the squash function, which is defined as

$$f(\boldsymbol{s}) = \frac{||\boldsymbol{s}||^2}{1+||\boldsymbol{s}||^2}\frac{\boldsymbol{s}}{||\boldsymbol{s}||}, \text{ where } \boldsymbol{s} \text{ represents an output vector.} \tag{5}$$

In this equation, $\frac{\boldsymbol{s}}{||\boldsymbol{s}||}$ is the definition of the normalisation of $\boldsymbol{s}$. $||\boldsymbol{s}||$ is the L^n norm of vector $\boldsymbol{s}$, where n is the length of $\boldsymbol{s}$. The first term, let us call it the squash factor, transforms the result into a non-linear function. Whenever the length of $\boldsymbol{s}$ is very large, the squash factor will approach 1. Conversely, whenever the length is very small, the squash factor will approach 0. A plot of the squash factor as a function of the length of $\boldsymbol{s}$ can be seen in Figure 5.

As we can see, as the length continues to grow, the squash function will continue to grow towards 1, which would denote a 100% probability of the entity that this capsule represents appearing in the input image.

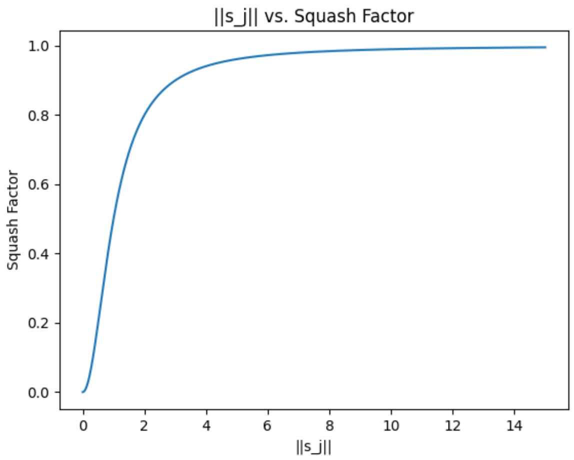


Fig. 5: The squash factor

3.2 Digit Capsules

The vectors that were derived by the Primary Capsules are then passed into Digit Capsules. The eight-dimensional outputs of the Primary Capsules are fully connected with the Digit Capsules layer. This layer will have some weight matrix $\boldsymbol{b}$ of size $8x16$ associated with it. Following basic linear algebra, the matrix multiplication of an eight-dimensional vector and weight matrix $\boldsymbol{b}$, will result in a sixteen-dimensional output vector. This output vector will act as the input vector to the Digit Capsules.

In a conventional CNN, a pooling method would be applied in-between the two convolutional layers. Rather than pooling, Hinton et al. have developed a novel routing technique. Routing algorithms will be used to determine the weights in weight matrix $\boldsymbol{b}$, such that related concepts will be connected within each capsule. The initialization of $\boldsymbol{b}$ can be done in various ways, with similar results. However, the most common initialization is named tabula rasa, which means the entire matrix is set to 0 at initialization. This technique will apply dynamic routing, which means the routing happens at runtime, rather than it being a hyperparameter. The goal is to route the capsules' outputs, such that the inputs of the capsules in the new layer agree with one-another. Dombetzki described this in the following simple terms in [4]: "... this can be compared to routing a detected nose to the face-capsule and not the car capsule. A detected nose, eye and mouth agree together in the face capsule, while the nose would not agree with a wheel and door in the car capsule."

Coincidence filtering [6] tells us that, in a high-dimensional space, whenever a group of data points has similar values, these data points are very likely to be related. We know the parameter space can grow to large dimensions very quickly. Because of this, it is extremely unlikely that a cluster of vectors are not related to each other.

Algorithm 1 The Routing algorithm used by Hinton et al. [13]

```
procedure ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
    for all capsule $i$ in layer $l$ and capsule $j$ in layer$(l+1)$: $b_{ij} \leftarrow 0$.
    for $r$ iterations do
        for all capsule $i$ in layer $l$: $\boldsymbol{c}_i \leftarrow$ softmax $(\boldsymbol{b}_i)$
        for all capsule $j$ in layer $(l+1)$: $\boldsymbol{s}_j \leftarrow \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i}$
        for all capsule $j$ in layer $(l+1)$: $\boldsymbol{v}_j \leftarrow$ squash $(\boldsymbol{s}_j)$
        for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\boldsymbol{u}}_{j|i} \cdot \boldsymbol{v}_j$
    end for
    return $\boldsymbol{v}_j$
end procedure
```

Thus, Hinton et al.'s routing algorithm will be a simple clustering algorithm on the input capsules. The chosen algorithm can be seen in Algorithm 1. This is a general iterative clustering algorithm. Notice that a hyperparameter r is mentioned, which defines the number of iterations of the clustering algorithm. After an empirical analysis, Hinton et al. recommend to set $r \approx 4$. Afterwards, the weight matrix $\boldsymbol{b}$, which links the previous capsules to the current capsules, will be defined.

3.3 Predicting the Class

All that remains is to compute the probabilities of each class appearing in the input image. Luckily, as mentioned before, the length of a capsule vector determines the probability that some entity appears in an image. Additionally, in the previous section, we described how various capsules were routed to the matching capsules in the Digit Capsules layer. Finally, we described how a capsule is a representation of an object, while individual neurons represent smaller segments of that object.

From this, we conclude that all vectors in a capsule will relate to the same object. Therefore, the lengths of the vectors in a capsule, are the probabilities that each of these segments appeared in the image. The capsule with the highest average vector length can be considered the most likely solution to the task.

Thus, we can conclude that the final fully-connected layer is no longer needed. A variation of this layer already appeared during the routing after all.

4 Analysis

Now that the concepts of a CNN and a Capsule Network have been explained, we can focus on comparing the two structures. In our comparison we take note of the strong points and the weak points for both structures, and look at any differences in performance, execution time and training efficiency.

4.1 Analysing CNNs

As has been mentioned in Section 1 and can be seen in Figure 1, a major weak point of using CNNs is that they are position invariant: as long as all features of an object are present in an image, the placement of those features in the image is irrelevant, which can lead to incorrect results in extreme cases. Another weak point is that since the viewpoint is the largest source of variance in an image, a single transformation on the image changes a lot of pixels making the CNN exponentially increase in size. Another thing to consider is that pooling involves the same neurons for the same activation. A translated image will then involve a different set of neurons for handling the activations. This is not intuitive as the human brain would recognise that the same object is visible before and after its translation and would activate the same neurons.

Another downside of CNNs is the fact that intermediate values in the feature extraction are extremely hard, if not impossible, to interpret. The final result is easily interpretable, it is the probability that some class is detected in an image. However, the intermediate states mean nothing to the researcher. This makes it difficult to track the progress of an architecture. In general, the performance of CNNs is satisfactory if only one kind of object is involved. For multiple objects their information needs to be kept separate and the CNN struggles with detection and segmentation [4].

4.2 Analysing capsules

As mentioned in Section 3, the main strong point of using capsules is viewport invariance: the probability of a certain feature being in an image is constant when the feature undergoes a translation. Since viewpoints are represented as linear transformations the capsule networks also have a better generalisation for new viewpoints [4]. Furthermore, as the neuron groups are connected between layers instead of individual neurons, a connection requires fewer parameters. A drawback is that the capsules look at everything in an image, so clutter in the image needs to be handled properly by e.g. modelling the clutter. Capsules

(a) MNIST (retrieved from [11] (b) LUNA16 (retrieved from [3] (c) CIFAR10 (retrieved from [10]

Fig. 6: Examples of the images in the data sets we will be discussing.

also assume that each location in the image contains at most one instance of an object. This can cause problems if an image is crowded with multiple instances of the same object.

Additionally, in contrast to the aforementioned uninterpretable intermediate values of a CNN, Capsule Networks work with vectors that carry important parameters regarding the entity it represents. Therefore, Capsule networks allow for a certain understanding of the classification process. While this property may not directly affect the performance of the classification, it does help the researcher in understanding where an architecture fails and where it is accurate. This could be considered equally important.

4.3 Theoretical comparison

When theoretically comparing the two structures it is evident that capsules fix a lot of the shortcomings that CNN's have: capsules are viewport invariant while CNN's are not; capsules require fewer parameters than CNN's due to the connections between groups of neurons. Furthermore, as a network of capsules converges in fewer iterations than a CNN, less amount of training data is needed. This can be important for use cases where only a small amount of data is available.

4.4 Comparing Performance

In order to properly compare the performance of conventional CNNs and the novel Capsule Networks, we have searched the literature for implementations of both regular CNNs and Capsule Networks on three data sets, which increase in complexity. Specifically, the three data sets we will compare are:

- MNIST (CNN: LeNet-5 [9], Capsule Network: CapsNet [13])
- LUNA16 (CNN: U-Net [7], Capsule Network: SegCaps [7])
- CIFAR10 (CNN: ViT-H [5], Capsule Network: CapsNet [13])

In Figure 6, an example image of each of the data set can be viewed. The MNIST dataset consists of images of handwritten numerical digits, the LUNA16 dataset consists of images of lung nodules and the CIFAR10 dataset consists of coloured images in 10 different classes.

In the papers referenced above both conventional CNNs as well as Capsule Networks were built and applied to the three well-known datasets mentioned above. The results can be seen in Table 1

	CNN (error in %)	Capsule Network (error in %)
MNIST	0.95	0.25
LUNA16	1.55	1.52
CIFAR10	0.50	10.60

Table 1: A comparison of various architectures on three well-known data sets.

These results will be discussed in the following section, where conclusions will be drawn from the analysis.

5 Discussion

In Section 4, we discussed the various positives and negatives of both conventional CNNs and Capsule Networks. We will first discuss any potential issues that should be kept in mind when interpreting the results we collected. Then, we will take this analysis and use it to answer our research question. Afterwards, we will look back at our hypothesis and modify it if necessary. Finally, we will state our final thoughts on the topic.

Overall, we found that Capsule Networks offer some major improvements over conventional CNNs. These improvements include viewport invariance, fewer hyperparameters and trainable parameters and a shorter training phase. Viewport invariance allows for far more accurate and complex architectures to be built, which could lead to them being used by Computer Vision research groups, who often develop very specific software to detect objects while prior knowledge is abundant. Furthermore, less parameters (both hyperparameters and training parameters) greatly decreases the required data and training time, which will assist researchers in finding optimal hyperparameter settings in acceptable time frames. Additionally, we noticed an increase in performance for simpler data sets by Capsule Networks. This can be seen in Table 1. Finally, Capsule Networks apply interpretable intermediate states. Through these states, it becomes simpler to understand the process that a Capsule Network goes through to obtain a final result. This may even add to Explainable AI research.

However, we must also consider the negatives we observed regarding Capsule Networks. Specifically, whenever data sets grow complex, current Capsule Networks are quickly overwhelmed in comparison to conventional CNNs. Additionally, CNNs offer many more variations, which are geared towards specific problem. This makes it very likely that these specialised networks will have a better performance than Capsule Networks.

Various reasons may exist for these shortcomings. The most probable reason: CNNs have been around for far longer than Capsule Networks. It is possible that, as Capsule Networks continue to be explored, techniques will be discovered that assist Capsule Networks in overcoming these limitations.

One reason for possible discrepancies in our data may be that, for some of the data, large time periods exist in-between the CNN and Capsule Network approaches. Improved architectures may have been discovered in the present, however, the chosen architectures seem to be the leading architectures for the specific data sets. Additionally, we must take care when comparing the results in Figure 1. As mentioned in Section 4.4, the results were obtained by various researchers, through multiple research papers ([5], [7], [9], [13]). This allows the results to be mostly unbiased, however, it introduces the issue that these results may not accurately represent the current optimal architectures for either network. Nevertheless, significant differences in error rates were discovered. This helps us answer the research question that was posed at the beginning of the paper.

Let us first reiterate this question:

How do the newly introduced Capsule Networks compare to conventional Convolutional Neural Networks?

We hypothesised the following: ”From an initial literature study, we believe Capsule Networks have the potential to become a powerful alternative to conventional CNNs. However, we also believe that Capsule Networks are unlikely to fully replace conventional CNNs in the near future.”

After an extensive study, we have found that Capsule Networks already have proven themselves to be an alternative to CNNs, for simpler data sets. However, we, as authors, still hold the opinion that, given more time, Capsule Networks will grow to match the performance of conventional CNNs, even for more complex problems. Additionally, Capsule Networks may become the preferred method, due to their explainable nature, as mentioned in the analysis.

6 Conclusion

In conclusion, we have analysed conventional CNNs and Capsule Networks. We then compared them in various criteria. We found that Capsule Networks offer great benefits over conventional CNNs. However, at this time, researchers have not been able to apply Capsule Networks to complex problems and achieve state of the art performance. They have, however, managed to improve the state of the art on data sets such as MNIST and LUNA.

Based on these results, we believe that, currently, the field should not yet adopt Capsule Networks as its standard method for position invariant neural networks. We do, however, believe that more research should be performed on Capsule Networks, as they clearly have the potential to be as effective as, if not more effective than, conventional Convolutional Neural Networks.

7 Future Work

In the future, more research could be done in the following areas:

- In this paper, we did not take notice of alternative routing algorithms. Comparing the effectiveness of various routing algorithms could lead to interesting insight.
- The Capsule Network we mainly considered in this report is a very simple, yet effective, example. More complex Capsule Networks may be analysed in detail. These networks may even have the potential to solve more complex problems.
- If time allowed for this, we could have performed our own experiments, rather than study the results of others. This would have allowed us to compare training times, classification speeds. Additionally, it would have allowed us to ensure that there were no external circumstances affecting the obtained results.

References

[1] P. Ahamed, S. Kundu, T. Khan, V. Bhateja, R. Sarkar, and A. Mollah. Handwritten arabic numerals recognition using convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing*, 11, 11 2020.

[2] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, A. Mraz, T. Kashiyama, and Y. Sekimoto. Transfer learning-based road damage detection for multiple countries. *CoRR*, abs/2008.13101, 2020.

[3] F. V. Beers. Capsule networks with intersection over union loss for binary image segmentation. In *Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods*. SCITEPRESS - Science and Technology Publications, 2021.

[4] L. A. Dombetzki. An overview over capsule networks. 2018.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[6] Z. Gao, Y. Li, Y. Yang, N. Dong, X. Yang, and C. Grebogi. A coincidence-filtering-based approach for cnns in eeg-based recognition. *IEEE Transactions on Industrial Informatics*, 16(11):7159–7167, 2020.

[7] R. LaLonde and U. Bagci. Capsules for object segmentation, 2018.

[8] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*. The MIT Press, 1995.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

[10] Y. Min and M. Bennett. Causal explanations of image misclassifications. *CoRR*, abs/2006.15739, 2020.

[11] F. Pan, Z.-X. Zhang, B.-D. Liu, and J.-J. Xie. Class-specific sparse principal component analysis for visual classification. *IEEE Access*, PP:1–1, 06 2020.

[12] D. Rumelhart, G. Hinton, and J. Mcclelland. A general framework for parallel distributed processing. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, 01 1986.

[13] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.

[14] H. Yingge, I. Ali, and K.-Y. Lee. Deep neural networks on chip - a survey. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, Feb. 2020.

Misinformation in social networks: Spread and Susceptibility

Andrei Stoica
Cassandra Ann Fernandes

Abstract—
With the sudden growth of digital media, there is a surge in accessible information and with the rise of social media and networks, they act as an agent in misinformation as well. This has become an increasingly worrying phenomenon, due to the risk of diffusing public attention to irrelevant topics, thus diminishing the focus on important matters. Virtual means of communication enhance the transmission of raw, unfiltered information to large communities. Large scale reactions happen within very short time spans, hence creating chaos. Public security can be threatened and, more importantly, the public opinion can be misled. So, how does one differentiate true and useful information from misinformation?

Current research suggest that misinformation thrives off the main social networks, such as Twitter, where like-minded individuals interact and create circles. One common solution used in practice in order to address misinformation spread through social media platforms and these circles is deploying interventions which falsify a piece of information by cross-checking with a correct source.

The relationship between the initiator of the correction and the owner of the target tweet accounts for the spread of rumours. Users who promote misinformation acknowledge it with a higher rate when it comes from friends, i.e. users who follow each other, compared to the situation where the correction is published by a stranger, i.e. a user with no formal connection on the network. The reaction to corrections happening within ongoing conversations or ”out-of-the-blue” has been measured in terms of response time or frequency of subsequent replies in defined time windows. Moreover, reactions to challenging and less challenging tweets, respectively, have been tracked.

We analyzed the segregation of the network into groups of naive and hesitant and we reviewed the existing research on how one group influences the other. They offer insights into the existence and behaviour of homophilies within communities where misinformation is spread and we explore potential answers to the following question: can we identify tight-knit circles of users within these communities? Additionally, we investigate which parameters, such as particular account attributes and network topology, influence the creation of a homophily and how it is connected to the detection of communities and network segregation.

Index Terms—Misinformation, Fact-checking, Network segregation

1 Introduction

Social and digital media have been gaining popularity for a variety of purposes, to interact with their social circles and to keep up with the news and upcoming world affairs. People may use digital media to express their opinions and ideas about an assortment of topics and ongoing issues. But without any sort of fact-checking, this can create an environment where an accelerated spread of misinformation occurs. Without any sort of verification of facts, this can lead to disastrous effects on the public.

Misinformation spread over social media causes the public focus to be directed to peripheral topics. It is a mechanism frequently implemented to manipulate opinions, as well. For instance, during the 2016 presidential election period in the United States of America, a Russian manipulation campaign occurred on Reddit, a social network [Roose, 2018]. This claim was made by a moderator who spotted suspicious activity on Reddit's largest pro-Trump forum. Users submitted links to a web address managed by Russian nationals and funded by Russia's Federal News Agency. At first sight, links appeared to be from trusted websites, registered and hosted in America. However, after accessing them, readers got redirected to a Russian page. Reddit decided to perform an investigation and, consequently, banned several web addresses. The incident can be considered an attempt of propaganda, using misinformation, leading to polarization and community segregation in online network discussions on the aforementioned social platform.

In tense situations, another layer of physical distress is added to people who absorb incorrect information. The authors in [Yasmim Mendes Rocha, 2021] provided a systematic review which analyses psychological and physical reactions to the pandemic generated by the COVID-19 virus. Due to effects such as economic recession, home confinement, disruption in education or the shift to online learning, research in the field needed to be intensified in order for scientists to report the potential findings to the population. Subjects in the aforementioned study suffered from mild disorders (fatigue, fear or panic), as well as serious health issues, such as anxiety, stress, depression or insomnia. Multiple other publications were included in the analysis and the results showed there is a general acceptance of the information (either true or misleading) which circulates through the media channels. One additional observation mentioned by the same authors, which emphasises the impact of misinformation, is the following: the rumour that the consumption of pure alcohol would diminish the effects of the virus and would eliminate the virus from the body completely led to 800 people dying and another 5876 being hospitalized for methanol poisoning.

While misinformation is an ever-present and ongoing phenomenon, one solution to combat it is deploying a fact-checking intervention. The authors [Hannak et al., 2014] present the following context: one user who signals a potentially false statement posted by another user on a social media platform verifies the information with the help of a third-party specialised institution, such as government websites, Wikipedia or other traditional or non-traditional source and posts a reply, indicating the source of the information. The terms introduced by the authors are *”snoper”* and *”snopee”*, which refer to the initiator of the fact-check and the user whose initial post is questioned, respectively.

Past publications concluded that individuals are resistant to change their beliefs when they face an opposite opinion [Ullrich K. H. Ecker, 2010]. However, factors such as the social relationship between the *snoper* and the *snopee*, the structural position or the temporal relationships are analysed and they do have an impact in the reactions to fact-checking interventions.

Zubiaga et. al performed an analysis on rumour threads posted on Twitter, annotating each tweet in terms of three dimensions, namely support and response type, certainty and evidentiality [Zubiaga et al., 2015]. The study observes the behaviour of users in

Andrei Stoica - a.stoica@student.rug.nl
Cassandra Ann Fernandes - c.fernandes.2@student.rug.nl

two periods of the rumour cycle, i.e. prior to the resolution of its veracity status and after it has been verified or debunked. The authors concluded that true rumours tend to be accepted as correct faster than false rumours get resolved. Moreover, well-known organisations, such as news publishers, tend to support rumours, regardless of whether they have been debunked or confirmed, tweet with certainty and offer evidence within their tweets.

The structure of the present paper is the following: Section 2 offers an overview of the fundamental concepts and states the research questions to be extended throughout the following sections. This is followed by the analysis of the existing literature in Section 3. Section 4 is dedicated to our ideas and approach and finally we conclude the paper with our findings and discussion in Section 5.

2 Background

While there is no doubt misinformation has a negative influence on individuals in certain instances, we aim to retrieve and compare quantified metrics with respect to the behaviours of individuals within networks, how subjective preferences influence the reaction to a false piece of information or how the time aspect of a corrective reply influences the follow-up reaction in the following sections. In this section we aim to introduce the existing literature in the main topics that will be covered in our research.

2.1 Echo chambers

We investigate the idea of an echo chamber and how this concept relates to homophily. Due to the growth of the internet and the access to social media, the volume and variety of information have also increased. This can give rise to varying opinions or ideologies depending on what information people were exposed to [Barberá et al., 2015].

In an echo chamber with like-minded people, when a member makes a claim, it usually gets mimicked by other members or repeated without questioning or critical analysis. In other terms, they mimic each other's uncritical statements on the perspectives of a particular source that is unverified without any debates [SourceWatch, 2022]. Essentially, an echo chamber is a condition in which an individual only hears or sees information or opinions that mirror and reinforce their ideas.

According to [Nguyen, 2020], echo chambers can use another epistemic safeguard mechanism which the authors mention to be the ”disagreement-reinforcement mechanism”. This mechanism can be described as where members in an echo chamber are persuaded to hold fast to a set of beliefs or ideas, with the presence and voicing of conflicting opinions reinforcing the original set of beliefs and the discrediting of the conflicting opinion - making it difficult to ”dislodge” or change, since it is self-reinforcing, confined, and designed to ignore any contradictory information.

2.2 Homophily

The idea of homophily comes from the Ancient Greek words of (homós) meaning 'same, common', and (philía) 'friendship, love'. [Wikipedia contributors, 2022]. In this context, homophily in social settings can be defined as the tendency of people to connect and engage with similar individuals [Explained, 2022]. The existence of homophily is common if not prevalent among those individuals using social networks to connect with others, which could manifest in some way or another but nonetheless exists on these networks. The homophily concept drives various sorts of network links, including marriage, friendship, employment, counsel, cooperation, knowledge transmission and other various types of ties.

As a result, people's private social networks are homogeneous in nature and have many socio-demographic, behavioral and interpersonal features [Mewa, 2020]. The authors stress the implications that severe homophily can have. It restricts the individual's world view from what kind of information they acquire, the perspectives they develop and the types of interactions they have with other individuals. When homophily exists in the context of race and origins of people, followed by age, religion, education, occupation and gender, it creates the strongest distinctions within groups of people.

2.3 Network segregation

A closely related phenomenon to homophily is *network segregation*. The authors [Henry et al., 2011] state that there is a widespread tendency for network linkages to concentrate between actors who have similar key characteristics. They form tightly-knit communities of homogeneous actors and reinforce division between disparate groups. Cooperation and conflict on social networks have shaped group dynamics in the context of multiple issues, such as climate change or economic development. Hence, this is a particularly important aspect to take into account when considering the spread of misinformation. The authors [Tambuscio et al., 2018] indicate that it is more likely for misleading information to thrive in the social groups separated from the rest of the network, partly due to the purpose of the algorithms which mediate the exposure to misinformation. It aims to filter and recommend stories with a high potential for engagement. Consequently, echo chambers are created, which favor confirmation bias and repetition.

2.4 Research Questions

With the existing concepts, we aim to explore the following research questions:

RQ1 How can we study network dynamics, given the misinformation corrective actions, in the shape of Twitter replies?

RQ2 Does homophily influence the creation of echo-chambers and how?

RQ3 Which are the peak times when misinformation is spread?

3 Misinformation Contexts

This section focuses on providing a detailed analysis of past publications regarding the factors which determine the spread of misinformation within networks, group behaviour following the spread of misinformation and the reaction time of other users in an attempt to debunk or confirm statements.

3.1 Network Dynamics and Corrective Actions

The authors [Hannak et al., 2014] emphasize the social aspect of fact-checking, i.e. the real-world conversations between users, as opposed to the asocial phenomenon of professional media institutions deploying such interventions in order to attract wider audiences.

The authors proposed a methodology where the data acquired, i.e. the fact-checking interventions, are replies to statements posted by other users, which contain references to at least one of the following dedicated websites: Snopes.com, PolitiFact.com and FactCheck.org. The collected data is represented by a set of 3969 tweets posted between January 2012 and August 2013 on the Twitter ”gardenhose” public stream. The authors mention that, although their study focuses on tweets with certain restrictive characteristics, the fact-checking phenomenon can be seen in other shapes too. For instance, tweets containing no links or links to other websites could prove to be corrections as well.

Existing literature, [R. Kelly Garrett, 2013], acknowledges that revealing the correct information, i.e. successfully debunking non-valid information, does not have a significant influence on the attitudes and beliefs of the individuals, specifically if it is not aligned with their beliefs. Cohesion of a group is often more important to individuals who are part of it than the actual truth value of the information. [DiFonzo and Bordia, 2007] state that constituents of a group can speculate novel or uncertain situations, by having unanimous views on a certain topic.

The authors [Hannak et al., 2014] defined the term *snope* as an attempt performed by an individual to get another individual to be aware of the true facts on a given topic or within a conversation. *Snopes* generally yield better results, i.e. the *snopee* acknowledges the veracity of his statement after the correction, if the *snoper* is the leader of a group.

The study aimed at discovering details about the relationship between the *snoper* and the *snopee*, in the shape of structural positions and temporal relationships. *Snopes* between users who have a mutual connection on the network are only a fraction of 30% out of all *snopes*, meaning most of the interactions happen between either total strangers or at least one of the parties is not aware of the other. The structural position is the difference between absolute popularities. Experiments demonstrated that within friend-to-friend *snopes*, both participants have similarly-sized audiences that are usually relatively small. In contrast, the social positions of a *snopee*, who is not aware of the *snoper*, but at the same time is followed by him, are clearly different. In this case, the user whose initial tweet was corrected has a larger audience. A comparison between the numbers of followers of *snopers* (red) and *snopees* (green), corresponding to each type of relationship between users involved in such *snopes*, is illustrated in Figure 1.

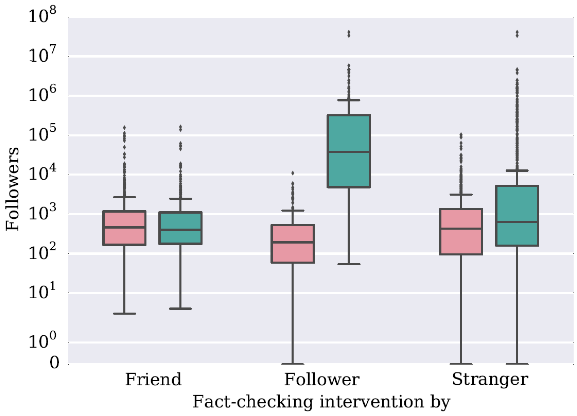


Fig. 1. Comparison between the number of followers of *snopers* and *snopees* [Hannak et al., 2014]

Another finding is that the *snopers* usually have smaller direct audiences, but these are more tightly connected and have larger indirect audiences than *snopees'* indirect audiences. An assumption made by the authors is that, within well-connected communities, activists are prone to call out celebrities, having the support of their communities. Regarding temporal relationships, empirical results showed no pattern between the *snoper's* and the *snopee's* ages.

Additionally, the paper of [Hannak et al., 2014] discussed whether any behavioural consequences exist after launching fact-checking interventions. Two types of snopes have been observed, namely ”within ongoing conversation” or ”out-of-the-blue”, the latter being the most frequent. Friends are roughly three times more likely to respond to ”out-of-the-blue” *snopes* from friends than they are from strangers. This can be explained by the nature of the interventions, which, between friends, are less provoking/challenging than between strangers. Also, friends are more likely to respond to challenges from friends. Strangers, irrespective of their intended *snope*, are definitely more likely to be ignored.

3.2 Echo chambers and Homophily

According to the authors [Jamieson and Cappella, 2010], the echo chamber is a confined space in online media that has the potential to amplify messages exchanged within it and insulates them from rebuttal. The authors [Amy Ross Arguedas and Nielsen, 2022] introduce the term as well, mentioning that social scientists use it to define a situation that exists due to a certain supply, demand and distribution.

It is mentioned that these echo chambers, in principle, could regard any topic, and amplify any type of message regarding the respective topic. This leads to any kind of message to be exchanged, either ambiguous, backed by evidence or simply something completely false. The authors also stress the fact that social scientists have studied echo chambers that are politically one sided, where people get their information exclusively from sources that are aligned with the respective side.

Another incident the authors state is that Twitter data is often used for analysis, since it is easier to analyse, though these echo chambers are not restricted to one platform. They find a pattern of ”cross cutting exposure” [Dahlgren, 2019] where people who are frequent visitors of online news on one side of the spectrum also tend to frequent news of the other side of the same spectrum. The paper notes that in the study of [Bos et al., 2016] there is evidence in The Netherlands that there is partisan ”selective exposure” to this news, but this was undermined by the public service broadcasting services. This suggests that even though most people use diversified media platforms, they often focus on large sources with diversifying viewpoints and very few rely entirely on partisanal sources. They mention that there are studies which support the idea that exposure to like-minded biased or partisanal media that are under certain experimental conditions can reinforce the views of other like-minded partisan individuals [Levendusky, 2013]. This indicates the existence of a homophily, where individuals reinforce each other's beliefs in an echo chamber.

We investigate this statement with the study of authors [Boutyline and Willer, 2017] where people with multiple political opinions can display varying degrees of homophily or the inclination to associate with those that have similar political ideologies. They elaborate that political groups or networks with stronger political homophily have better connections with their members, rather than those people with views which do not align with theirs. This suggests that higher levels of certain politically aligned homophily have a reduced chance to encounter those with varied political beliefs. Those that show stronger homophily strengthen and reinforce their beliefs as they have higher rates of encounters and interactions. They mention the study from prior work [Centola, 2010] that this political homophily generates extensive clusters of ”within group ties” that reinforce behaviors and patterns. This paper emphasizes the statement that those individuals with stronger inclinations look for assurance through social interaction, which can mimic echo chambers while affording them the reassurance they require. In this paper, the authors analyse Twitter network statistics within the range of 13% of US adult citizens using the platform [Smith and Brenner, 2012]. Their data looks at the user's ideologies and their ties between members of congress and policy non-profit organizations. Based on this data, the authors propose the following hypotheses:

- Groups with politically right ideologies exhibit larger homophily than the left politically aligned groups.
- Those groups that are politically extremes have greater political homophily than those that are neither extreme, nor neutral.

To investigate these hypotheses, the authors' method looks at different Twitter networks of politically involved Americans. Broadly speaking, these imply looking at 159 congress members and their ties with 33 policy non-profit organizations and used these as a benchmark to analyse their followers' orientations through the computation of some homophily measures and the use of multivariate regression with ”cluster adjusted” standard errors.

The parameters that they use to measure in their research is their political orientation and homophily. They measure the political orientation in terms of their ideology score that can measure the orientations of the senators and representatives as proxies of their Twitter followers. Their homophily measure works by modeling the ties that are interpersonal and linking those that could be casual acquaintances or friendships. They distinguish ties as those that are interpersonal as user-to-user connections versus their audience where the ties are from a hub of an organization or popular figure to users.

One of the drawbacks to the dataset, as mentioned by the authors, is that it is a rather diverse and large study case, than a snapshot of the data, with the tradeoff being that since the Twitter dataset contains public data, it was easier to calculate the number of likely homophilous partners per user to have a baseline of homophily rates.

The publicly available dataset is a snapshot of the Twitter network from June 2009, as the authors mention that this archival data precedes the ”who to follow” feature on the platform. It supported users to follow the same accounts as their peers, which encouraged greater homophily.

Their results show that the rates of homophily are higher amongst the more conservative and politically extreme persons which may have major implications for the emerging mechanisms of their particular political networks.

3.3 Peak Timings

As the use of social networks can be used to obtain information about current events and the news, it can cause a lot of people being misled and even endanger lives. In this section, we also investigate how timing affects the spread of misinformation and rumors as well as how it affects corrective actions.

The authors [Zubiaga et al., 2015] propose a methodology that takes a look at the dataset comprising of 330 rumour threads (4,842 tweets) linked to 9 newsworthy events. By separating two levels of standing in a rumour life cycle, i.e. before and after its veracity status is resolved, they create an understanding of how people disseminate, support, or refute rumours that are subsequently confirmed true or untrue. Their dataset contains a further contains rumor stories that can be one of three categories:

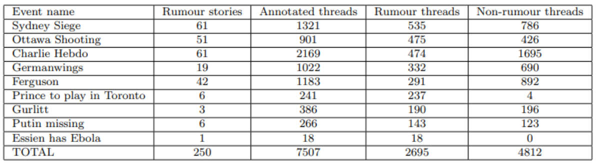
- true
- false
- or unverified

Each rumour story comprises a number of rumour threads, and a timeline in which the threads are arranged by time. When determining whether a story is genuine or untrue, one tweet was chosen as the resolution tweet from the chronology of the tale.

These conversations in the dataset were annotated with the following aims in mind:

- Categorize separate rumourous tweets (and related conversation threads) as being part of the same tale,
- Annotate the source tweet to be either a rumor or not

The number of source tweets marked as rumours and non-rumours for the events where each narrative corresponds to a collection of rumourous tweets are mentioned in the Figure 2.



Event name	Rumour stories	Annotated threads	Rumour threads	Non-rumour threads
Sydney Siege	61	1321	535	786
Ottawa Shooting	51	901	475	426
Charlie Hebdo	61	2169	474	1695
Germanwings	19	1022	332	690
Ferguson	42	1183	291	892
Prince to play in Toronto	6	241	237	4
Gurlitt	3	386	190	196
Putin missing	6	266	143	123
Essien has Ebola	1	18	18	0
TOTAL	250	7507	2695	4812

Fig. 2. Table showing the dataset used by [Zubiaga et al., 2015]

As Twitter implemented the feature where users can reply to one another, the replies to the 2695 rumourous source tweets were also recorded and analysed as part of the dataset.

With this dataset, the authors examine the timelines from the rumors gathered and labelled for the nine events where the rumors were colored according to their veracity, i.e. an unverified status (orange) from where the rumors have their starting point after which can either be determined to be true (green) or false (red) and some uncategorised which are left to be colored as orange throughout the timeline established in Figure 3.

From this study, they state that there is a delay from when a rumor is posted for the first time and it being determined as either true or false. The delay is greater in the case of debunking false rumors than when confirming a true statement. This difference is significant in the case of a false rumor, with the median delay being over 14 hours. This is comparatively longer, as the median delay of a true rumor takes around 2 hours, as we see illustrated graphically in Figure 4.

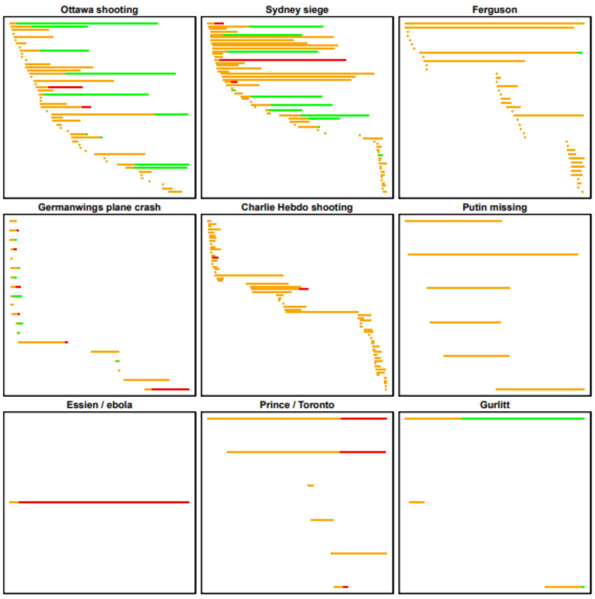


Fig. 3. Timelines of the events as conversational threads according to their veracity status [Zubiaga et al., 2015]

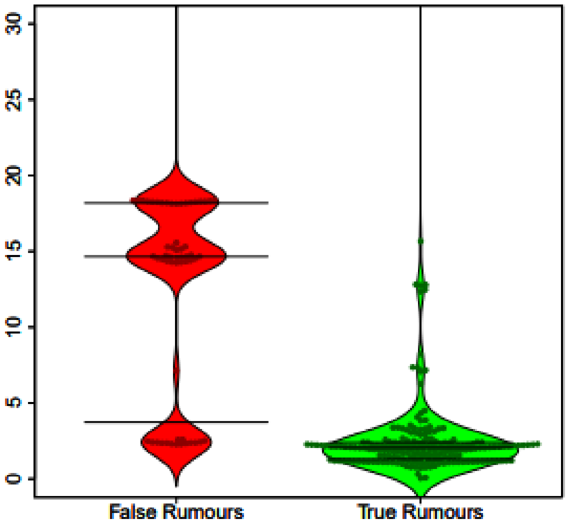


Fig. 4. Illustration of delays between false and true rumors. [Zubiaga et al., 2015]

In order to draw a more in depth view of this analysis, the authors [Zubiaga et al., 2015] look at the ”diffusion” of the rumors which are in the form of retweets. These retweets also prove to be useful in analyzing how the rumors were spread. They are classified as:

- accurate: either refute false rumors or accept true rumors.
- inaccurate: retweets of the original tweets that are false which either refute the true rumors or accept the false rumors.
- unverified: retweets that are not verified.

They mainly showcase their research about how often each sort of rumour gets retweeted, as well as the time-effect on the rumour diffusion patterns, where some rumours are retweeted more frequently at first and then decrease in number as time goes on, by using the average distribution of retweets for various rumorous events or stories.

They look at the ratio of retweets that appear over 15 minute epochs. They distinguish between tweets that come before and after resolving tweets, (pre- and post-) tweets, that are part of a real or false rumour, and tweets that support or reject a rumour.

They show that tweets sent before resolving tweets receive a lot of retweets in the first few minutes, but this pattern goes away in less than 20 minutes. This is effect is demonstrated in the Figure 5.

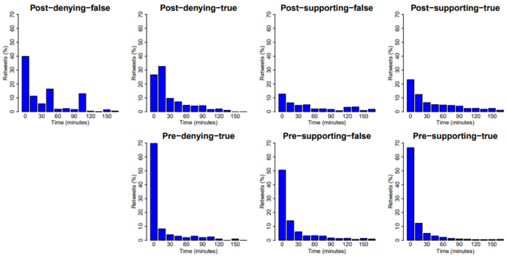


Fig. 5. Figure from [Zubiaga et al., 2015] showing Retweet timelines where the number of retweets at the beginning. This implies that there is a lot of interested generated in the first 15 minutes; implying more spread. [Zubiaga et al., 2015]

The retweets are more uniformly distributed over time after the occurrence of resolving tweets, indicating that post-tweets are retweeted over a longer period of time. The conclusion they infer from this is that the pre-supporting tweets support rumors that are unverified. Additionally, the number of retweets supporting both false and true rumors also drop drastically once the rumor has been determined to be of either category, implying that the interest in the rumor drops after it has been determined.

Ultimately, this proves that this research uncovers an intriguing trend in user behavior: users tend to endorse unverified rumours (explicitly or implicitly), maybe due to the arousal that these early, unconfirmed reports generate and their potential social influence.

4 Open Questions and Our Approach

Throughout this paper, we analysed three important aspects with respect to misinformation and its spreading behaviour. We identified several publications which propose reliable solutions to tackle it, as well as analyse the phenomenon through the lens of different factors, such as structural positions within the network, group behaviour and peak reaction times. However, the effects of misinformation can be quantified through other metrics, as well. Some still open questions that we identified are:

- Does the network topology within communities change over time, given it is exposed to both misinformation and corrective actions?
- Is there any difference between the evolution of follow/unfollow relationships within communities where a certain influential node promotes misinformation and communities where the homologous node supports true facts?

In this section, we offer a thorough explanation of our approach to the aforementioned questions.

As previously stated, there is typically a reaction to misleading tweets within a certain time span. We consider the reference point in time as t_0 and the moment after n time units, noted as t_n, with n large enough to potentially spot changes. The data to be analysed is a network formed of a ”central” influential node, i.e. a node/user who has a large number of followers within a certain community, who regularly promotes misinformation, and the subnetwork/cluster of followers. We compare the topology of this network in both time instances. We aim to find out whether the activities of the cluster (tweets regarding the same events, identified via hashtags) differ over time. Did they intensify in numbers or is the topic not relevant anymore? Does the ”central” node have as many followers at time t_n or has this number decreased from time t_0? Did the corrective actions influence the opinions of some nodes within the subnetwork, such that they even unfollowed the ”central” node in the community?

Provided we obtain numerical results as answers to the questions above, we could draw some conclusions. Specifically, if the number of nodes that unfollow the ”central” node is large, we observe a change in the general opinion. This leads to the assumption that misinformation is not believed anymore. This could indicate a pattern of network susceptibility and, on top of that, the results could be further used to identify how long misinformation survives within a network.

As far as the second open question is concerned, we ought to compare the evolution of the number of followers over time for both a ”central” node which supports false information and another ”central” node, corresponding to a different community, which posts true statements. This way, we could quantify the effects of misinformation on the general public. If both values are constant in time, then misinformation is as powerful and as influential as true facts, which would be a worrying sign. If the former value reduces over time, then corrective actions have a positive impact on the general viewpoint regarding a certain topic, i.e. people generally change their opinions.

5 Conclusion

Social media interactions have become a point of interest since the emergence of digital platforms in the last decades. They enable quick and efficient communication, regardless of the geographical location of users. This is why they have attracted attention from numerous users. Information of any kind is spread through this channel, so the inevitable scenario of misleading facts being shared needed to be addressed. A solution which has been deployed in practice is the fact-checking intervention. However, this may have limited impact on future courses of actions and people's actual beliefs, depending on certain factors which we aimed to comprehensively present, explain and compare throughout our paper. We looked into individual relationships between users involved in such interventions, as well as group behaviours. We further analysed the relationship between echo chambers and homophilies. These echo chambers exist mainly due to the fact that like-minded people amplify and intensify beliefs of people with similar key attributes, although this number can be small. In a political environment, however, we investigated how members of tight-knit circles associate themselves with people showing similar political views. By measuring their ideologies and the homophily rates of the networks, it has been proven that individuals with more conservative or extreme ideologies have higher homophily in those respective networks. When considering the timing of when misinformation is spread, we can conclude from the current research that the beginning of when a newsworthy event has been released has more weight, i.e. it is considered peak time, when rumors are dispersed within networks of people. A rumor that is deemed true takes significantly less time to be resolved, when compared to a false rumor.

References

[Amy Ross Arguedas and Nielsen, 2022] Amy Ross Arguedas, Craig T. Robertson, R. F. and Nielsen, R. K. (2022). Echo chambers, filter bubbles, and polarisation: a literature review. Reuters Institute, University of Oxford, The Royal Society.

[Barberá et al., 2015] Barberá, P., Jost, J. T., Nagler, J., Tucker, J. A., and Bonneau, R. (2015). Tweeting from left to right: Is online political communication more than an echo chamber? *Psychological Science*, 26(10):1531–1542. PMID: 26297377.

[Bos et al., 2016] Bos, L., Kruikemeier, S., and de Vreese, C. (2016). Nation binding: How public service broadcasting mitigates political selective exposure. *PLOS ONE*, 11(5):1–11.

[Boutyline and Willer, 2017] Boutyline, A. and Willer, R. (2017). The social structure of political echo chambers: Variation in ideological homophily in online networks. *Political Psychology*, 38(3):551–569.

[Centola, 2010] Centola, D. (2010). The spread of behavior in an online social network experiment. *Science*, 329(5996):1194–1197.

[Dahlgren, 2019] Dahlgren, P. M. (2019). Selective exposure to public service news over thirty years: The role of ideological leaning, party support, and political interest. *The International Journal of Press/Politics*, 24(3):293–314.

[DiFonzo and Bordia, 2007] DiFonzo, N. and Bordia, P. (2007). Rumor psychology: Social and organizational approaches.

[Explained, 2022] Explained, E. (2022). Homophily explained.

[Hannak et al., 2014] Hannak, A., Margolin, D., Keegan, B., and Weber, I. (2014). Get back! you don't know me like that: The social mediation of fact checking interventions in twitter conversations. *Proceedings of the International AAAI Conference on Web and Social Media*, 8(1):187–196.

[Henry et al., 2011] Henry, A. D., Prałat, P., and Zhang, C.-Q. (2011). Emergence of segregation in evolving social networks. *Proceedings of the National Academy of Sciences*, 108(21):8605–8610.

[Jamieson and Cappella, 2010] Jamieson, K. H. and Cappella, J. N. (2010). Echo chambers, filter bubbles, and polarisation:.

[Levendusky, 2013] Levendusky, M. S. (2013). Why do partisan media polarize viewers? *American Journal of Political Science*, 57(3):611–623.

[Mewa, 2020] Mewa, T. (2020). 'birds of a feather: Homophily in social networks' by miller mcpherson, lynn smith-lovin, james m cook (2001). In *Identifying Gender and Sexuality of Data Subjects*. https://cis.pubpub.org/pub/birds-of-a-feather-2001.

[Nguyen, 2020] Nguyen, C. T. (2020). Echo chambers and epistemic bubbles. *Episteme*, 17(2):141–161.

[R. Kelly Garrett, 2013] R. Kelly Garrett, Erik C. Nisbet, E. K. L. (2013). Undermining the corrective effects of media-based political fact checking? the role of contextual cues and naïve theory.

[Roose, 2018] Roose, K. (2018). We asked for examples of election misinformation. you delivered.

[Smith and Brenner, 2012] Smith, A. and Brenner, J. (2012). Twitter use 2012. Pew Research center.

[SourceWatch, 2022] SourceWatch (2022). Echo chambers.

[Tambuscio et al., 2018] Tambuscio, M., Oliveira, D. F. M., Ciampaglia, G. L., and Ruffo, G. (2018). Network segregation in a model of misinformation and fact-checking. *Journal of Computational Social Science*, 1(2):261–275.

[Ullrich K. H. Ecker, 2010] Ullrich K. H. Ecker, Stephan Lewandowsky, D. T. W. T. (2010). Explicit warnings reduce but do not eliminate the continued influence of misinformation.

[Wikipedia contributors, 2022] Wikipedia contributors (2022). Homophily — Wikipedia, the free encyclopedia. [Online; accessed 21-March-2022].

[Yasmim Mendes Rocha, 2021] Yasmim Mendes Rocha, Gabriel Acácio de Moura, G. A. D. C. H. d. O. F. D. L. L. D. d. F. N. (2021). The impact of fake news on social media and its influence on health during the covid-19 pandemic: a systematic review.

[Zubiaga et al., 2015] Zubiaga, A., Hoi, G. W. S., Liakata, M., Procter, R., and Tolmie, P. (2015). Analysing how people orient to and spread rumours in social media by looking at conversational threads. *CoRR*, abs/1511.07487.

Range-Only SLAM Algorithm: A Comparison

Maarten van Ittersum, Arjan Tilstra

Abstract— The SLAM (Simultaneous Localization And Mapping) problem is used in robotics to create a sense of vision for the agent. The SLAM problem is known to be paradoxical, as localization requires to have an accurate understanding of the environment, such as a map, while mapping requires sensor information about the current location. Many different approaches have been documented in solving the SLAM problem, using statistical approximation to generate a map and a path. These approaches are in the form of algorithms and each have their own requirements and performance.

Our paper gives a comparison between 5 SLAM algorithms, WiFi-SLAM, GraphSLAM plus an extended version, FastSLAM and SAM. These SLAM algorithms are compared on performance (time complexity and accuracy) and requirements (input data). We found that SAM and FastSLAM have a preferable time complexity, while SAM has the best accuracy. Furthermore, only the extended GraphSLAM requires LIDAR data, making the usage of this algorithm more expensive.

Index Terms—SLAM, WiFi, mapping, localization, robot navigation.

1 Introduction

Finding your way in buildings can be difficult for robots, as visual perception using light can be difficult and often very expensive, unlike for humans. However, as the only visible waves are light for humans, robots can make use of a much larger frequency bandwidth. Unfortunately, being indoors means that GPS signals are not available or unreliable. With WiFi being the norm nowadays indoors, robots can make use of the signals sent by routers to localize and map the space. In robotics, this is called Simultaneous Localization And Mapping (SLAM). SLAM combines both the mapping of an area with keeping track of the agent that is exploring the area. This is an important area of robotics, as the problem is a paradox. In [19], the problem is described as a paradox: *"to move precisely, a mobile robot must have an accurate environment map; however, to build an accurate map, the mobile robot's sensing locations must be known precisely."* Thus, in this problem mapping is defined as interpreting sensor information to estimate the locations of landmarks. Localization means finding the location on the map by analysing the sensor information.

Over the last years, solutions by approximation have been found for the SLAM problem. The different approaches use different statistical or probability analysis methods to solve the problem, to give a solution in realistic timing. Approaches differ not only on their mathematical methods, but also on the sensor data required to work on. In this paper, we look at Range-Only (RO) variants of the approach. These approaches make use of sensor data that only contains the range of the agent to a beacon. These approaches are easily accessible, as WiFi beacons and receivers can be used, which are cheap and very commonly found nowadays. Additionally, we also look at one approach that is extended by also using the sensor data of a LIDAR sensor. A LIDAR sensor does not require external beams to detect and sends out its own light beams. This provides much more information about the surroundings, while also adding up to the total cost of the project, as the sensor is heavy and expensive. Next to RO and LIDAR data, many more SLAM approaches have been found, such as echo or acoustic SLAM [15, 5, 6], visual SLAM [14, 13] and radar SLAM [10]. Our paper only focuses on comparing RO-based SLAM approaches.

To compare the different approaches properly, we look at the different attributes of an approach. As the approaches are in the form of an algorithm, we look at what steps are required to perform in these algorithms. Boolos & Jeffrey [2] describe an algorithm as explicit instructions for an input set, returning with an output. In our comparison, we will be looking at the input, output and the instructions. For the input, the types of sensors required are compared. The instructions are compared by computational complexity. Specifically, the worst-case time complexity is considered, which is the worst amount of time an algorithm takes over a certain size of input. Finally, the accuracy of the localization in SLAM is compared. Kümmerle *et al.* [18] describe a method that calculates the error of the output, based on the ground truth. We compare the calculated error, as the smallest error amount means a more accurate result.

In our paper, we look at five different approaches towards the SLAM problem. These approaches all make use of RO sensors, but differ in their statistical approach of mapping and localization. We look at WiFi-SLAM by Ferris *et al.* [7], GraphSLAM by Huang *et al.* [11] and an extended version of GraphSLAM by Kudo *et al.* [17], FastSLAM by Montemerlo *et al.* [20], and finally SAM by Dellaert & Kaess [4].

In the next sections, we go more in-depth into the five SLAM approaches. In section 2, we describe the structure of the input data, as well as the steps for each approach. In section 3, we describe the steps that we take in the comparison. In section 4, we perform the previously explained steps and in section 5 we discuss the results of the comparison. Finally, in section 6, we wrap up with a conclusion and give a recommendation for the future.

2 The SLAM Algorithms

There has been quite a lot of research done into different approaches of the SLAM problem. These approaches can be complex in their inner workings, as statistical and probability analysis are used often to find an approximation. Our first step in the comparison is to look into the algorithms and describe what instructions and data structures are used. Furthermore, we look at how exactly the input data is structured. In this section, we first describe the sensor data in detail, following up with the description of each algorithm.

2.1 The sensor data

In the studies that we compare, the authors used a dataset that was recorded when walking around in a building. During the tour, a device would pick up signals from Range-Only (RO) sensors, such as WiFi beacons, and the signal's strength. With this, we have three types of data. We have the device's state, x_n, which is a time frame. We have our landmarks, l_n, being the sensors. We link the state and the landmark together with measurements of the strength of the signal, z_n. The structure is shown in figure 1. To avoid the data association problem and keep the landmarks separated, the RO-sensors emit a unique identifier.

- *Arjan Tilstra is with the University of Groningen, E-mail: a.tilstra.2@student.rug.nl.*
- *Maarten van Ittersum is with the University of Groningen, E-mail: m.van.ittersum@student.rug.nl.*

Manuscript received 31 March 2008; accepted 1 August 2008; posted online 19 October 2008; mailed on 13 October 2008.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org .

Certain SLAM approaches require the input data to be already available in full or batch when running the algorithm. This means that the algorithm cannot be used when real-time updates are required.

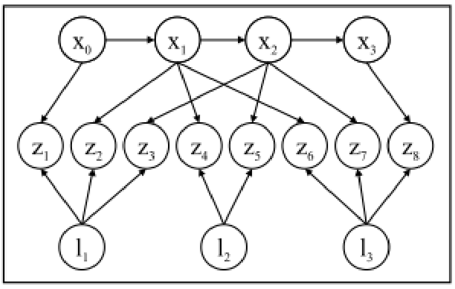


Fig. 1: The chain of data. x_n is the state of the device at moment n. l_n is landmark n. z_n is a measurement of a landmark at a state. Figure by Delleart *et al.*

2.2 WiFi-SLAM

One method is described by Ferris *et al.* [7]. The authors call their method WiFi-SLAM and it aims to apply the usage of WiFi signals in order to work around one of the weaknesses of the more common SLAM approaches [24]: the reliance on the calibration data for the sensors used. This data is inherently flawed when using WiFi signals, as the signal strength varies in unpredictable ways in different circumstances. Complications can include other moving objects, people passing by or even subtle changes in the atmosphere. In order to prevent this, the authors make use of Gaussian process latent variable models (GP-LVM). This technique is used to map high-dimensional data to a low-dimensional latent space. The high-dimensional data in the problem's context is the input data: the signal strengths of all the WiFi access points in range of the mapping device. The low-dimensional space used here consists of two dimensions, which are effectively interpreted as the xy-coordinates of the mapping device.

The technique furthermore uses some constraints or assumptions: locations that are near each other should perceive similar signal strengths, similar perceived signal strengths indicate the two signals where measured near each other and finally, location data is read sequentially and sequential locations should be near each other.

The first constraint is informed by a property of GP-LVM (as well as other dimensionality reduction techniques): similar input data has similar output data and vice versa, therefore in this problem, similar locations must have similar signal strengths.

The second constraint can lead to problems in environments where WiFi access points are sparse, as similar signal strengths may be observed in different locations around the same access point. However, the authors note that the approach is typically valid in office environments, where usually multiple access points are in range of the device.

The third constraint is dictated by the author's goal of modeling the movement of a person or device moving through a building, therefore it does not make sense for sequential data points to be far from each other in physical space.

2.3 GraphSLAM

Another approach is offered by Huang *et al.* [11]. This work aims to improve upon the previous example by highlighting and correcting two of its shortcomings: the reliance on specific, predefined shapes in the mapped area (narrow and straight hallways, primarily) and the assumption that WiFi fingerprints are unique in different locations. Doing away with this second assumption would allow this approach to work better in environments where access points are sparse. This work makes use of the fact that SLAM using WiFi signal strength can be formulated as a GraphSLAM problem. It also improves the GraphSLAM algorithm to be more time-efficient: while the original algorithm has a time complexity of $O(N^3)$, the adapted version has a time complexity of $O(N^2)$, where N is the dimensionality of the state space, or the number of poses being estimated.

One of the main appeals of any GraphSLAM-based approach is that the algorithm itself reduces to a standard non-linear least squares problem, which means that it has access to many thoroughly studied optimization techniques. The author's made use of the Gauss-Newton algorithm to solve this part of the problem.

2.4 Extended GraphSLAM

A third approach is proposed by [17]. This method focuses on extending the GraphSLAM algorithm, making use of both WiFi signals, LIDAR scans and odometry data from the mapping device. This approach has the advantage of being able to more easily solve the loop closure problem, in other words, detecting when a place that has already been visited is revisited. Once the algorithm concludes that the mapping device has arrived in a location that it has visited before, the generated map is updated retroactively to include the loops found so far. A major drawback is, of course, the large amount of data that this approach needs in order to work. The other methods did not require any other data than the WiFi signals.

2.5 FastSLAM

A fourth approach that we will be analysing is proposed in a paper by Montemerlo *et al.* [20]. The FastSLAM algorithm is based on SLAM using a particle filter, the extended Kalman filter (EKF) [19].

The extended Kalman filter uses two steps in the recursive state estimation, the prediction and correction step. The state estimation uses a state vector created from the current position of the device, as well as the measured locations of the landmarks.

$$s_t = (x_t, y_t, \theta_t, m_{1,x}, m_{1,y}, \cdots, m_{N,x}, m_{N,y})^T \tag{1}$$

The SLAM approach loops over each state in the data. In the loop, the first step is to use EKF predict to predict the next state. After, the next state is read and new beacons are initialized if these are not yet initialized. Finally, the EKF correction is done. EKF assumes that the data set has a Gaussian distribution [23].

FastSLAM solves the complexity issue of using the extended Kalman filter (EKF), which is updating the covariance matrix of $O(K^2)$ elements if a single landmark is observed. The approach uses an observation made in [21] that determining the landmark locations can be decoupled in K problems. The algorithm uses a balanced binary tree to access each landmark per particle used in the resampling step, reducing the complexity to $O(M \log K)$.

2.6 SAM

The final approach is the SAM algorithm proposed in [4]. This approach uses smoothing estimates, unlike the previously mentioned FastSLAM and EKF SLAM, which are filtering approaches. SAM uses the factor graph representation of the collected data. A factor graph is a is a bipartite graph that expresses how a global function of many variables factors into a product of local functions [16]. We can express the measurements made into a factor graph. After generating a factor graph of the measurements, the algorithm creates estimates for each node in the graph. A matrix of Jacobian matrices and a RHS vector are calculated, from which the least-squares can be found. As node elimination introduces fill-ins, of which too many can cause slow factorization, the order of elimination is approximated using Column Approximate Minimum Degree Ordering (COLAMD).

This version of the square root SAM uses batching or requires all data to be available at once. In newer work by the authors, incremental SAM (iSAM) can work as soon as data is received [12].

3 Comparison method

With better insights into the different SLAM approaches, we describe a comparison method in this section. We focus on three methods, comparing computational complexity, input and output.

3.1 Computational time complexity

The first measure we are going to compare the algorithms with is their time complexity. Time complexity is an expression of the time a computer will take to run an algorithm. It is expressed as a factor of the input data given to the algorithm. In many of the SLAM algorithms, the time complexity is $O(n^2)$, meaning the time the computer takes, scales with the size of the input data quadratically.

A lower time complexity is a big advantage for the compared algorithms, for obvious reasons. Robots making use of these approaches are often not very powerful in terms of computing speed, so efficient algorithms are more viable to use in most situations.

3.2 Input sensed data

Several different types of input data can be used by SLAM Algorithms. The mapping device needs a way of gathering this input data, so each additional type needed, or more complex type needed, means more sensors need to be attached. In addition, such sensors can be very expensive or difficult to attach to a robot due to their size. For instance, WiFi signal strength sensors can be very small and are often already built in to most simply computers. Opposed to this, LIDAR scanners can be much more expensive, making it less easy to implement a SLAM algorithm that makes uses of this technology. In general, an algorithm that gets good results with fewer or simpler types of data is preferable to one that needs more or more complicated input data. Of course, if perfection is required, more input data could be the only way to achieve that.

3.3 Localization accuracy

SLAM has two outputs, a map and the path that leads through the map. In this comparison, we look at how the calculated path differs from the ground truth. In an experimental setup, the ground truth can be known by creating a map from real-life data, such as a blueprint. The path in the measurement can be mapped out by exactly calculating the position of each measurement in relation to the true map. After running SLAM, the true map and the output map can be fitted towards each other, such that the output path and the ground truth path can be put in the same space. Plotting this shows both routes, such that a visual representation of the accuracy can be made.

Kümmerle *et al.* [18] give the following equation to find the error of the output:

$$\varepsilon(\delta) = \frac{1}{N}\sum_{i,j} \mathrm{trans}(\delta_{i,j} \ominus \delta^*_{i,j})^2 + \mathrm{rot}(\delta_{i,j} \ominus \delta^*_{i,j})^2 \tag{2}$$

In this equation, δ is the distance and rotation relative to the previous measurement, and δ^* is the relative distance to the previous state in the ground truth. The $\ominus$ operator is the inverse of the standard motion composition operator ($\oplus$). In this equation, the error of the rotation and translation are separated, such that these can be compared individually, while also not adding up the error multiple times.

In our comparison, we use the results from the authors of the papers. The authors do not use the same datasets. Instead, different datasets with different surfaces are used. To create a better understanding of how these measurements are done, we divide the surface by the mean error.

$$\text{accuracy} = \frac{\text{surface}}{\text{error}} \tag{3}$$

4 Comparison results

In this section we will be comparing the different approaches to the SLAM problem. We will be comparing the approaches on time complexity, method, and accuracy. To do this, we look at the data from the studies that describe the results.

4.1 Time complexity

The different approaches analyzed in this paper have varying degrees of time complexity. Obviously, a lower time complexity will allow the device using it to do so faster or with less computing power, making the approach generally more viable, assuming the error rate remains similar.

The WiFi-SLAM method described by Ferris *et al.* [7] has no stated time complexity in the source work and no pseudo code of the algorithms is present to derive this from. Nevertheless, we know that the method makes use of the computation of a variance-covariance matrix, which has a theoretical best-case time complexity of $O(n^{2.373})$ [1]. Therefore, we can conclude that the total algorithm will have this as its theoretical best as well, likely being worse than this in practice.

The method proposed by Huang *et al.* [11] is based on the GraphSLAM algorithm, which enabled the authors to improve the time complexity from $O(n^3)$ to $O(n^2)$.

The approach offered by Kudo *et al.* is also based in GraphSLAM and boasts a similar time complexity to Huang *et al.*'s method, being $O(n^2)$.

The FastSLAM algorithm contributed by Montemerlo *et al.* [20], as mentioned above, has a time complexity of $O(M \log K)$, where M is the number of particle filters the algorithm uses and K is the number of landmarks the path comes across.

Finally, the SAM algorithm proposed by Dellaert *et al.* has no explicit mention of a time complexity. The method makes use of a lot of matrix multiplication, but many of the matrices are sparse. Therefore, the worst-case time complexity is less relevant than in most cases. From experimental results, the SAM algorithm can be compared with the FastSLAM in figure 2.

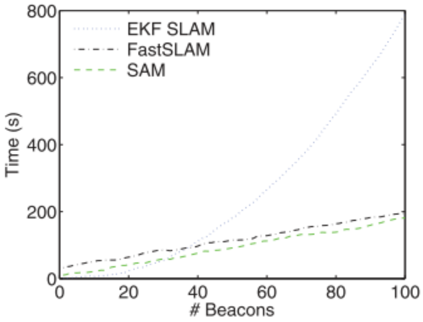


Fig. 2: The time complexity of EKF SLAM, FastSLAM and SAM measured, ran against a certain amount of beacons. Figure by Herranz *et al.*

4.2 Input data

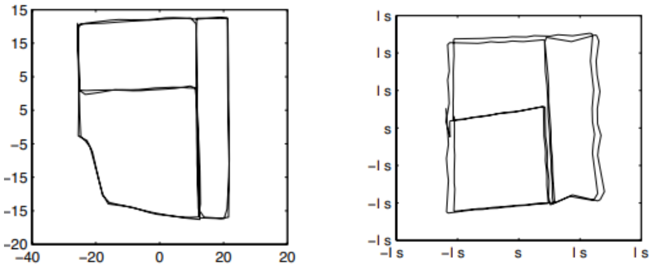
All of the analyzed methods make use of RO-sensor signal strength, such as the signals from WiFi beacons, but some need additional data as well, which makes these algorithms less straightforward to use.

The WiFi-SLAM, FastSLAM, SAM and GraphSLAM algorithm all make only use of landmarks, captured with RO-sensor data. These sensors have to be kept separated by using a unique identifier, as otherwise the problem of data association will arise, making the algorithm more complex. The structure of this data is shown in figure 1.

The exceptions to only using RO-signals is Kudo *et al.*'s extended GraphSLAM. In their experiments, they augment the SLAM algorithm by using LIDAR scan-based 2D SLAM with a WiFi augmentation. This means that the dataset used in the algorithm requires much more data than the other algorithms.

4.3 Localization accuracy

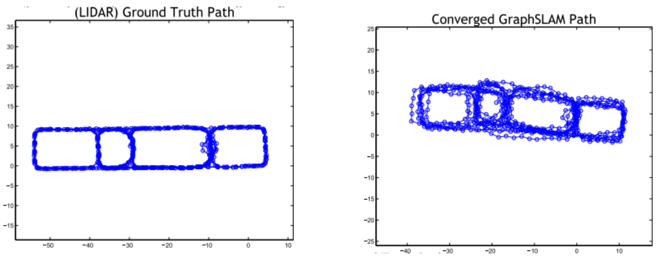
For localization and mapping algorithms, accuracy of the resulting data is important. Having less accurate paths can cause robots to make mistakes in movement, with a risk of becoming stuck. To compare the accuracy, we look at experiment results. These results compare the estimated path to the ground truth path, using Euclidean or Mahalanobis distance between the estimated locations and truth. When data of the ground truth path is not available, visual comparison can be done using maps or blueprints.



(a) The ground truth

(b) The estimated path

Fig. 3: The mapped out ground truth in 3a and the estimated path in 3b, using the WiFi-SLAM algorithm by Ferris *et al.*



(a) The ground truth

(b) The estimated path

Fig. 4: The mapped out ground truth in 4a and the estimated path in 4b, using the GraphSLAM algorithm by Huang *et al.*

In [7], an experiment was run in a university building, with a surface of around 30×60 meters. The ground truth and estimated path plots are shown in Figure 3. In these plots, a clear difference can be seen in bottom left angle, as this data was lost. Furthermore, the localization accuracy is measured using an approach shown in [3]. The authors calculated a mean error of 3.97 meters.

In [11], the ground truth and estimated path are both plotted again (Figure 4). This experiment was ran on a university floor with a surface of 60×10 meters. When comparing both plots, we can see a similar structure of the graph. The estimated path is much wider and seems to be translated and rotated slightly. Huang *et at.* did measure the localization accuracy with the same method as Ferris *et at.*, resulting in a mean error of 2.23 meters.

The extended GraphSLAM algorithm in [17] ran two experiments, one for mapping a large area and finding a loop and one for the localization of a person. The mapped plot is shown in Figure 5.

For both the FastSLAM and SAM algorithm, we will be using the results from [9]. In this study, the algorithms were ran against the two UAH datasets [8]. UAH1 are measurements of a second floor with some small corridors and a larger main corridor, with a total surface of 60×60 meters. UAH2 is a third floor with four large corridors forming a square, with a total surface of 120×120 meters. In Figure 6, the estimated paths are mapped in red together with the ground truth in blue. The FastSLAM algorithm reported to have a mean error of 3.028 meters on UAH1 and 4.501 meters on UAH2. The SAM algorithm algorithm reported both 2.672 meters and 1.505 meters as the mean error, on UAH1 and UAH2 respectively.

In table 1, the accuracy statistics of the algorithms are compared. We normalize the accuracy by dividing the total surface by the mean error, as having a larger error on a smaller surface means that the algorithm performs worse. The best performing algorithm has the highest score, which is the SAM algorithm.

Algorithm	**Surface**	**Mean error**	**Normalised**
WiFi-SLAM	1800m^2	3.97m	453.400504
GraphSLAM	600m^2	**2.23m**	269.058296
FastSLAM	3600m^2	3.028m	1188.90357
SAM	3600m^2	2.672m	**1298.7013**

Table 1: An overview of each algorithm with their mean error. For the normalised value, we used $\frac{\text{surface}}{\text{mean}}$. The larger the value, the better the algorithm performs.

5 Discussion

In this paper, we compared the different types of SLAM algorithms with each other. First, we described the main differences of each algorithm. After that, we made the comparison of the algorithm on time complexity, input data and accuracy. In this section, we will discuss the differences that we found in each algorithm, as well as discuss the approach that we used.

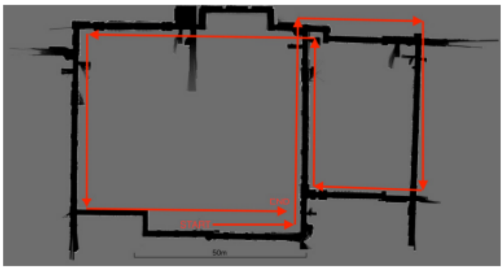
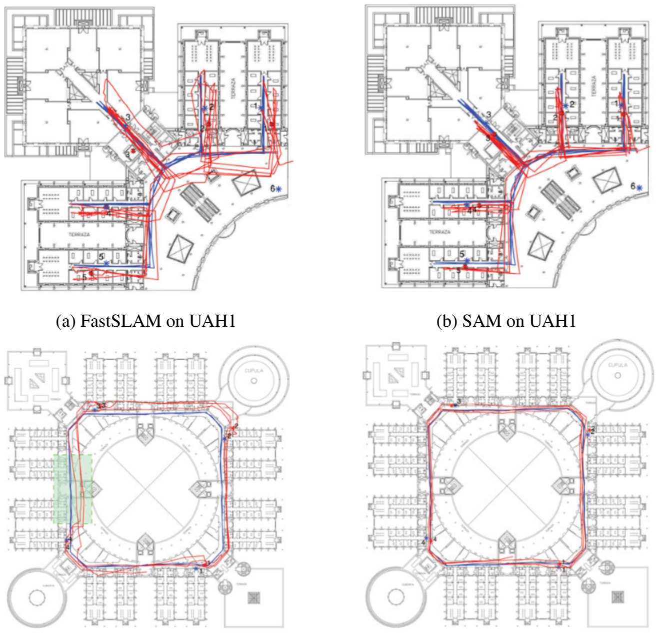


Fig. 5: The path found by Kudo *et al.* overlaid on its ground truth. Located at the second floor of a university in Taiwan.



(a) FastSLAM on UAH1

(b) SAM on UAH1

(c) FastSLAM on UAH2

(d) SAM on UAH2

Fig. 6: Both the FastSLAM and SAM algorithm estimations of the UAH1 and UAH2 dataset plotted (red) together with the ground truth (blue) on the blueprint of the true area. Results by Herranz *et al.*

5.1 Results

For the time complexity of the algorithms, we found that most algorithms are in the $O(n^2)$ range, with the exception of the FastSLAM and SAM algorithm. The FastSLAM algorithm is known to be in $O(M \log K)$. In the experiment results, the SAM algorithm shows to be in the same range. This makes these algorithms have the best time complexity, being the most time efficient while running on low-end hardware.

The results of the input data comparison is very basic. As four of the algorithms all make use of only the RO-sensor data, these make for the most obvious choices in terms of ease to set up. However, in case LIDAR scan data is available as well, or extra accuracy is required, the extended GraphSLAM algorithm could be made useful, as mapping with this algorithm seems more accurate, while having a lower time complexity compared to other solutions using the LIDAR scan data.

With comparing the accuracy of the algorithms, the section has the most comparable data. As shown in table 1, we see that the algorithm that performs the best here is the SAM algorithm. This is confirmed in figures 6b and 6d, as the estimated path seems to be overlapping the ground truth path very accurately, without much of translation or rotation. The GraphSLAM algorithm provides a smaller mean error value however, but has a much smaller surface to make errors on. In the end, it is questionable on how comparable the results are, as different layouts are used in the different kinds of study. To have more accurate results, the algorithms should compete on the same dataset. Unfortunately, we were not able to come in contact with the authors of the UAH dataset [8].

5.2 Approach

In this paper, we analysed five different algorithms. The approach we took was to find and describe the main steps in the algorithm. Out of these main steps, we found the time complexity and type of input data. We looked at the experimental results in the studies to find the accuracy of the algorithm. All studies provided a mean error in the localization aspect of the algorithm. We compared these values after normalizing.

Our main focus with this approach was to structure results and to be able to compare them. This approach is still quite concise, as we are not able to compare different aspects of the algorithms. In [9], the results are much more extensive, e.g. providing translation and rotation error and showing a computational complexity analysis. However, the other studies do not provide extensive results like these, making it hard to find concrete data that can be compared. Our method primarily focused on analysing data that was already produced and theoretical data. The comparison of time complexity and the input data do not require any practical information, while the localization accuracy does.

5.3 Modern approaches

Our study majorly focused on the comparison of relatively older approaches towards the SLAM problem. As shown in a literature review by Panigrahi & Bisoy in 2021 [22], newer papers that describe a SLAM approach have come out. The approaches use similar approaches as our discussed approaches.

6 Conclusion

The aim of this paper was to compare multiple SLAM algorithms, providing a clear overview on each statistic of the algorithm. In a sense, our paper extends the paper by Herranz *et al.* [9], comparing the FastSLAM and SAM algorithm to three more algorithms.

For the comparison, we found that the FastSLAM and SAM algorithm both have the most optimal time complexity, being in $O(M \log K)$. As for the input data types, we found that four of the compared algorithms only use RO-sensor data, while one also uses LIDAR scan data. To decide on what the best option is, we have to look at the context of the problem, for example, whether LIDAR scan data is available or feasible to implement for the problem at hand. Finally, we have the accuracy. In this comparison, the SAM algorithm seems to be the best option, providing a low normalised mean error value over the rest. As mentioned in the previous section, the results are questionably comparable, as they have not competed against the same dataset.

Our future recommendations for this comparison is to do a practical analysis of the different approaches. The different approaches can be ran against the same dataset, thus generating much more comparable results. Furthermore, while our focus was primarily on WiFi-based approaches, modern robotics and hardware allow for much better measurements using different signals. For example, if an experiment involves modern day smartphones, LE-Bluetooth or sonar signals can be used additionally. Also mentioned in the discussion, our main focused was on older solutions towards the SLAM problem. Newer lesser-known solutions can be compared towards these older solutions.

References

[1] J. Alman and V. V. Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 522–539. Association for Computing Machinery, oct 2021.

[2] G. Boolos and R. C. Jeffrey. *Computability and logic*. Cambridge University Press, Cambridge, 1980.

[3] M. Bowling, D. Wilkinson, A. Ghodsi, and A. Milstein. Subjective localization with action respecting embedding. *Springer Tracts in Advanced Robotics*, 28:1–13, 2007.

[4] F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[5] C. Evers, A. H. Moore, and P. A. Naylor. Acoustic simultaneous localization and mapping (A-SLAM) of a moving microphone array and its surrounding speakers. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2016-May:6–10, may 2016.

[6] C. Evers and P. A. Naylor. Acoustic SLAM. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 26(9):1484–1498, sep 2018.

[7] B. Ferris, D. Fox, and N. Lawrence. Wifi-slam using gaussian process latent variable models. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2480–2485, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[8] F. Herranz, A. Llamazares, E. Molinos, M. Ocaña, and M. A. Sotelo. Alcalá WiFi database. *University of Alcalá*, 2016.

[9] F. Herranz, A. Llamazares, E. Molinos, M. Ocaña, and M. A. Sotelo. WiFi SLAM algorithms: An experimental comparison. *Robotica*, 34(4):837–858, 2016.

[10] Z. Hong, Y. Petillot, A. Wallace, and S. Wang. Radar SLAM: A Robust SLAM System for All Weather Conditions. *arXiv preprint arXiv:2104.05347*, 2021.

[11] J. Huang, D. Millman, M. Quigley, D. Stavens, S. Thrun, and A. Aggarwal. Efficient, generalized indoor wifi graphslam. In *2011 IEEE International Conference on Robotics and Automation*, pages 1038–1043, 2011.

[12] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[13] A. Kanwal, Z. Anjum, and W. Muhammad. Visual Simultaneous Localization and Mapping (vSLAM) of Driverless Car in GPS-Denied Areas. *Engineering Proceedings*, 12(1), 2021.

[14] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich. The vSLAM algorithm for robust localization and mapping. *Proceedings - IEEE International Conference on Robotics and Automation*, 2005:24–29, 2005.

[15] M. Kreković, I. Dokmanić, and M. Vetterli. EchoSLAM: Simultaneous localization and mapping with acoustic echoes. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11–15, 2016.

[16] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[17] T. Kudo and J. Miura. Utilizing wifi signals for improving slam and person localization. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 487–493, 2017.

[18] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots 2009 27:4*, 27(4):387–407, sep 2009.

[19] J. Leonard and H. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. *Intelligent Robots and Systems IROS*, (91):1442–1447, 2002.

[20] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem Michael. *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598, 2002.

[21] K. P. Murphy. Bayesian map learning in dynamic environments. *Advances in Neural Information Processing Systems*, pages 1015–1021, 2000.

[22] P. K. Panigrahi and S. K. Bisoy. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University - Computer and Information Sciences*, mar 2021.

[23] M. I. Ribeiro. Kalman and Extended Kalman Filters: Concept, Derivation and Properties. *Institute for Systems and Robotics Lisboa Portugal*, (February):42, 2004.

[24] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.

A Framework for the Comparison of Cross Validation and Bootstrap Techniques on Classification Problems

Lorenzo Rota and Aditya Ganesh

Abstract— In the domain of classification problems a key challenge is determining, and minimising, the prediction error rate [1] of a model. This problem becomes more acute when the number of samples available for training is low. Two related sets of resampling techniques namely Cross Validation and Bootstrapping aim arrive at an accurate estimation of the prediction error. The former performs this through repeated experimentation on a training set by creating a number of test subsets within it, and the latter through sampling with replacement of the samples within the data-set.

However, there is still significant ambiguity about the most appropriate technique to be applied to a given problem type. While Efron and Tibshirani [3] found that their .632+ Bootstrap outperformed Cross Validation, Borra et al [1] and Kim [8] independently found that the results were not as distinctive. Further, Kim noted that .632+ tends to have a downward bias while Cross Validation has an upward bias, with neither technique ostensibly presenting a clear advantage over the other.

The goal of this paper is to assess the two sets of techniques through a structured evaluation process, treating the estimation of generalisation error as a problem of minimising the deviation between predicted and actual error. Through this framework, incorporating a series of tests on a selection of commonly used classifiers and synthetic data-sets with controllable irreducible error, we propose a guideline as to the optimal selection of validation technique for a given problem type.

Index Terms—Cross Validation, Bootstrapping, Machine Learning, Generalisation Error

1 Introduction

In the realm of statistical learning theory, it is understood that a suitable *decision function* can be selected from the *hypothesis space* when its associated *risk function* is minimized. Since a risk function requires knowledge of the probability distribution of a set of *data values* associated with a decision function, which is typically not known at hand, it is necessary to estimate the risk function in an unbiased manner, such as through a *sample mean* [6]. In machine learning literature, this is more concretely understood as selecting a candidate prediction model based on an estimation of its *prediction error*. The bottle-neck here, is that the estimator for the prediction error can only provide an approximation of the true prediction error that is as good as the available data that is used for the model selection. Another problem in estimating the prediction error is that the data set, on which the model is trained, is not necessarily the best model for predicting novel data. This fact is justified by the bias-variance trade-off of a model, which suggests that an optimal prediction error requires that the model is not completely unbiased. A distinction should therefore be made between the testing and the training error of a model, where the true prediction error should be estimated by the former.

Over time, various estimators have been designed for accurately estimating the prediction error of a model, such as the Cross-validation (CV) and bootstrap estimators, as well as modifications of the two [5, 2]. The prediction error of both classes of estimators can be described through their bias-variance decomposition, where the CV estimator is designed to be an unbiased estimator with the shortcoming that the estimation error of its prediction error is vastly attributed to high variance [8]. The bootstrap estimator could be viewed as an extension of the CV estimator with the aim of minimizing the error by reducing its variance [3]. The authors specifically proposed using the .632 bootstrap estimator as well as the improved .632+ estimator, which factors in bias-correction. In the studies done by Borra et al. and Kim. [1, 8], the performance of the prediction error among both classes of estimators were evaluated for different types of predictive models through different simulations, albeit with different goals in mind; the focus of Kim [8] was to compare the prediction error performance of adaptive boosting and pruned decision tree classifiers, whereas Borra et al. [1] considered a suite of regressors, namely, regression trees, projection pursuit regression and feed-forward neural networks in Monte Carlo (MC) simulations. Similarly, Ounpraseuth et al. [10] studied performance estimates of Quadratic Discriminant Analysis (QDA) classifiers on MC simulations as well as real data sets. In the three papers, the results seem to be mixed, however, Kim [8] suggests that CV estimators tend to perform better over large data sets, whereas the bootstrap estimators perform better when the data sets are relatively small.

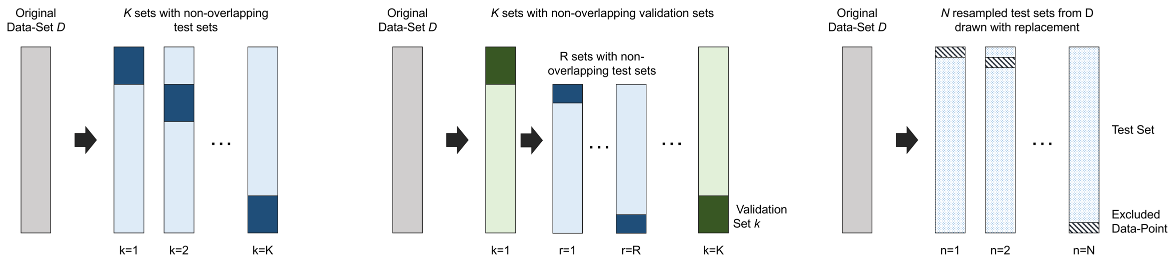
To better understand the utility of the CV and bootstrap estimators, it is important to note that the performance of specific prediction error estimator widely varies according to the type of the prediction model [2]. The main research objective of this paper is to devise and test a comparative framework that can be used for determining an optimal prediction error estimator. To apply this framework, we specifically consider binary classifiers due to their simplistic yet descriptive power. Moreover, we separate the comparisons for two classes of binary classifiers, namely the adaptive and non-adaptive classifiers. For the non-adaptive classifiers, we consider K-Nearest Neighbors (K-NN) and linear Support Vector Machine (SVM), and for the adaptive ones we consider a SVM with Radial Basis Function kernel, and Generalized Linear Vector Quantisation (LVQ).

The paper proceeds as follows: in section 2 (Literature Review) we provide the necessary background needed for understanding the problem at hand, in section 3 (Methods) we specify the comparative framework and how it will be used to compare the two classes of estimators, in section 4 (Results) we compare the results obtained from experiments in which different types of synthetic data sets are generated and utilised within the comparative framework. Finally, we carry out a discussion in section 5 (Discussion), and finish with a conclusion and section on future work.

2 Literature Review

This section outlines the validation techniques under consideration, namely Cross Validation and Bootstrap, along with the theoretical basis required to formally arrive at the objective of the paper.

- *Lorenzo Rota is a Master's student at the University of Groningen, E-mail: l.d.f.rota@student.rug.nl.*
- *Aditya Ganesh is a Master's student at the University of Groningen, E-mail: a.ganesh@student.rug.nl.*



(a) Cross Validation

(b) Nested Cross Validation

(c) Bootstrapping

Fig. 1: Resampling Techniques

2.1 Generalized Prediction Error and its Estimates

The generalisation error is a measure of a Machine Learning model's ability to correctly predict a value of an example that the model has not encountered during its training phase. Due to the fact that the principal utility of such a model lies in its predictive capability, methods to arrive at an accurate assessment of this error garner significant interest.

Typically, the generalisation error is understood in terms of a model's Bias and Variance [7]. Here Bias refers to error arising from imperfect approximation by the model to the data provided to fit the given reference values, while Variance refers to the degree of spread in the model's predictive ability should it have been trained on a different set of data.

These two metrics form what is commonly termed the Bias-Variance Tradeoff, typically demonstrated through the Mean Squared Error, i.e. the expected value of the squared distance between the predicted and actual outcome. It can be shown that the Mean Squared Error may be decomposed into three terms, Bias, Variance, and the Irreducible Error [7], i.e.,

$$E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + (Bias(\hat{f}(x_0))^2 + \sigma^2,$$

where

$$Bias(\hat{f}(x_0)) = y_0 - E(\hat{f}(x_0))$$
$$Var(\hat{f}(x_0)) = (\hat{f}(x_0) - E(\hat{f}(x_0)))^2$$

Typically, the Mean Squared Error has a single minimum value for some $(bias, var)$ that represents an ideal fit. The estimation of the generalisation error of a classification model may be treated as a minimisation problem of the Mean Squared Error of the predicted generalisation error obtained compared with the ground truth. This is possible only if the irreducible error σ^2 is known.

Due to the fact that this error is generally not possible to obtain in practice due to the absence of information regarding the population, the validation techniques under consideration have been constructed in order to arrive at an estimate of this measure.

2.2 Cross Validation

The Cross-Validation (CV) techniques for estimating the generalization error of a model appear in the literature well before it was named. In the paper by Stone [13], this generalized estimator is investigated as a means of assessing the performance of statistical predictions, and forms the foundation for the modern CV estimator and all of its variants. The variants that are considered for the comparative framework are K-Fold Cross Validation (K-CV), Leave One Out Cross Validation (L1O-CV) and Nested Cross Validation (NCV).

2.2.1 K-Fold Cross Validation

The idea behind K-CV is that a data set D is partitioned into K partitions or 'folds', where $K-1$ partitions in union form the training set, and the remaining partition forms the test set. The prediction error is then determined based on the test set, and the procedure is repeated for a total of K times where in each successive step, a new test set is selected. The approach is outlined in figure 1a.

2.2.2 Leave One Out Cross Validation

L1O-CV is a special case of K-CV, where $K := N$ when N is the size of the data set D. As the name suggests, the validation set consists of just a single sample, which is left out of the training set, and then used as the only novel sample to estimate the performance of the prediction error.

2.2.3 Nested Cross Validation

The NCV technique, shown in figure 1b, is an extension of K-CV where each training set is split further into R partitions, where the first $R-1$ folds form the new training set and the last fold forms the validation set. This means that there is an inner-loop with R iterations for each of the K outer-loop iterations. In this approach, we first perform model selection within each inner-loop, and then estimate the prediction error on the unseen testing set, as opposed to performing model selection on all samples from the data set and then trying to estimate the error by re-substituting the test set. This would avoid information leakage as it ensures that the model selection did not take place on any samples from the test set in each iteration of the CV procedure.

2.3 Bootstrap Techniques

Bootstrapping techniques, similar to Cross Validation, are resampling techniques. However, where Cross Validation divides a data-set into a number of folds, and performs repeated validation using them, Bootstrapping techniques seek to derive inferences regarding a given data-set through repeated sampling from it with replacement. Applying Bootstrapping as a validation technique involves the creation of validation sets from the given data-set by such resampling with replacement (i.e. drawing samples from the data-set without checking for uniqueness of the draw).

2.3.1 Leave One Out Bootstrap

When given a data-set of size P, the Leave One Out Bootstrap constructs P additional datasets, each of size P by iterating over the original, excluding exactly one element from it, and sampling from the remainder with replacement. A classification model is then trained on the original data-set, and its accuracy (termed *score*) is measured on each of the generated test-sets. These accuracies are then averaged to obtain the Bootstrapped measure of generalisation error. This technique may be represented as in figure 1c.

2.3.2 .632 and .632+ Bootstrap

It was noted by Efron and Tibshirani [3] that the Leave One Out Bootstrap tends to provide an overly optimistic estimate of the generalisation error when used in this manner. The reason is that the bootstrap and original training data have common samples, and so there is information leakage [5]. They proposed the .632 Bootstrap as a remedy to this problem, which is calculated as

$$score_{.632} = (0.368 \times score_{train}) + (0.632 \times score_{l1o})$$

According to Efron and Tibshirani [3] fractions 0.632 and 0.368 used in this formulation were due to the fact that when sampling with replacement the proportion of supported (i.e. unique) observations in the resampled data set tends to $(1 - \frac{1}{e}) \approx 0.632$.

This measure, however, was found by the authors to have a notable downward bias. As a modification, they further proposed the .632+ Bootstrap which determines a relative overfitting rate that further diminishes the Leave One Out score in cases of severe overfit.

3 Methods

In order to develop an understanding of the behaviour exhibited by these disparate validation techniques and construct meaningful evaluation criteria, a ”Comparative Framework” was created, varying a set of parameters, namely, data-set sample space size, data-set noise, nature of the data set, and machine learning technique employed. This paper limits itself to the construction of such a framework for classification techniques. The framework may be described formally as in 3.2, and the statistical basis for the formulation of this framework is given below.

3.1 Problem Statement and Statistical Justification

The framework to be constructed must identify the most appropriate validation technique for a given set of circumstances. In order to do this it is necessary to identify a measure of how accurately a given validation technique is able to arrive at the generalisation error of the model.

The estimate of generalisation error created by a validation technique is itself subject to bias and variance. This allows for the treatment of estimator accuracy as an exercise in determining the deviation of the estimated generalisation error from the actual generalisation error (i.e. as a regression problem), using the Mean Squared Error as a primary evaluation metric. The evaluation of predicted generalisation error in this manner is in accordance with the findings of Efron and Tibshirani [3], and was previously used by Borra et al. in their evaluation [1] using regression trees. Identifying the the most suitable validation technique is then a matter of selecting that technique that provides the lowest error estimate.

3.1.1 Controlling the actual generalisation error

The principal assumption made in formulating the problem statement is that the actual generalisation error of a given classifier is known, in order to determine the validation techniques that provide the lowest deviation from it. On an unknown data-set this would lead to a circular dependency, where the actual error requires estimation using the techniques under evaluation. In order to avoid this, synthetic data sets would need to be prepared with known, decision boundaries and thereby known upper bounds on classifier accuracy, measured as the misclassification error.

3.2 Comparative Framework

Given a dataset with size P representing some dichotomy $D \subset \mathbb{R}^N$, with known classes E and known irreducible error σ, the Bias B, Variance Var, Mean Squared Error MSE, and Execution Time T_{exec} of a given Classifier C are observed, subject to 5 fold Cross Validation (`K-CV`), Leave-One-Out Cross Validation (`1-CV`), Nested Cross Validation (`KR-NCV`), Leave-One-Out Bootstrap (`l1o-bootstrap`), .632 Bootstrap (`.632-bootstrap`) and .632+ Bootstrap (`.632+-bootstrap`).
The behaviour of B, Var, MSE, and T_{exec} for each validation technique are observed on independent variation of:

1. Population Size P
2. Irreducible Error σ
3. Nature of data set D
4. Nature of Classifier
5. Dimensionality of the data set

The classifiers selected for comparison are as follow:

1. 5 Nearest Neighbours (`knn-05`)
2. Support Vector Machines (Linear and RBF Kernels)
3. Generalised Linear Vector Quantization (`lvq`)

were chosen to serve as a representation of the classification techniques commonly used in Machine Learning applications. The implementations of `knn-05`, `svm-l`, and `svm-r` were taken from the standard Scikit Library [11], while the implementation of `lvq` was taken from the `scikit_lvq` module [9], based on the approach by Schneider, Bunte, Biehl, et al. [12]

The effects of the parameters outlined was observed on the bias, variance, Mean Squared Error, and execution time for each technique.

3.2.1 Data-Sets Under Test

As outlined in 3.1.1, a critical assumption that was required for the construction of the framework was that the irreducible error σ in D is known. In order to ensure this, synthetic data-set generators were designed in $\mathbb{R}^2$ with known decision boundaries. These data-set generators, adapted from the Scikit[11] library with modifications may be described as below

1. Box Generator

$$D \subset \mathbb{R}^2 \sim (x_i, y_i) \sim \begin{cases} [0, 5+\eta] & e_i = 0 \\ [5+\eta, 1] & e_i = 1 \end{cases}$$

2. Circle Generator

$$D \subset \mathbb{R}^2 \sim (x_i, y_i) = \begin{bmatrix} (r_i+\eta)cos\theta_i \\ (r_i+\eta)sin\theta_i \end{bmatrix}$$

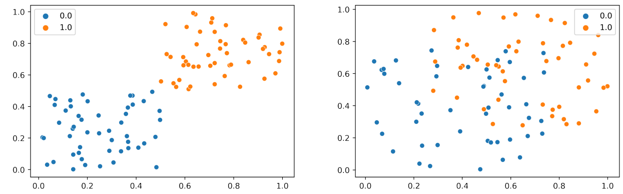
$$\theta \sim [0, 2\pi],\ \eta \sim \mathcal{N}(0, \sigma^2),\ r = \begin{cases} 0.7 & \text{if } e_i = 0 \\ 1 & \text{if } e_i = 1 \end{cases}$$

3. Gaussian Blob Generator

$$D \subset \mathbb{R}^2 \sim (x_i, y_i) \sim \begin{cases} (\mathcal{N}(-1, \sigma^2), \mathcal{N}(0, \sigma^2)) & \text{if } e_i = 0 \\ (\mathcal{N}(+1, \sigma^2), \mathcal{N}(0, \sigma^2)) & \text{if } e_i = 1 \end{cases},$$

where σ is a controllable parameter.

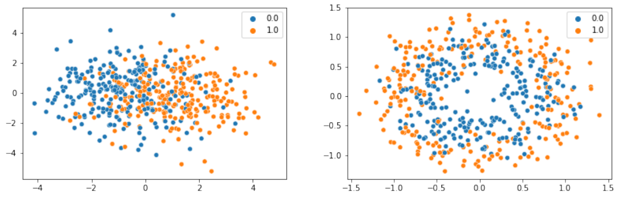
In all cases, e_i represents the class assigned to the sample (x_i, yi). Sample data-sets generated in this manner were obtained similar to those below.



(a) 100 samples, no misclassification

(b) 100 samples, 22.9% misclassification

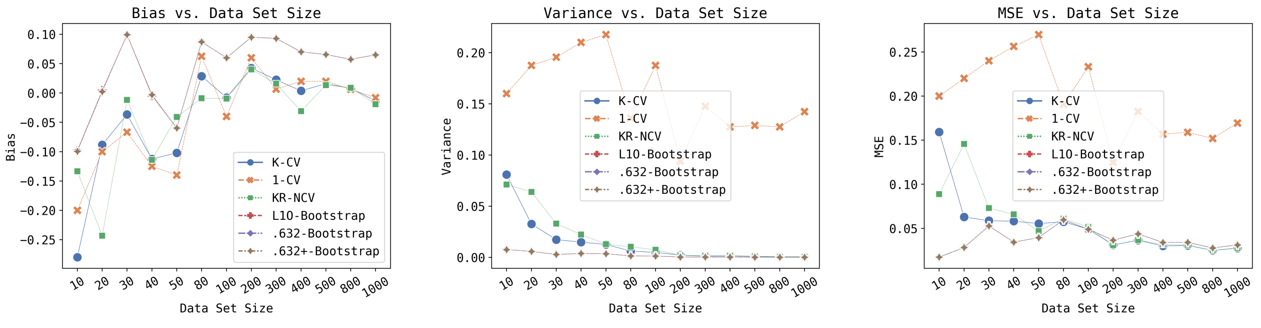
Fig. 4: Box Generator



(a)

(b)

Fig. 5: Gaussian Blob and Circle Generator

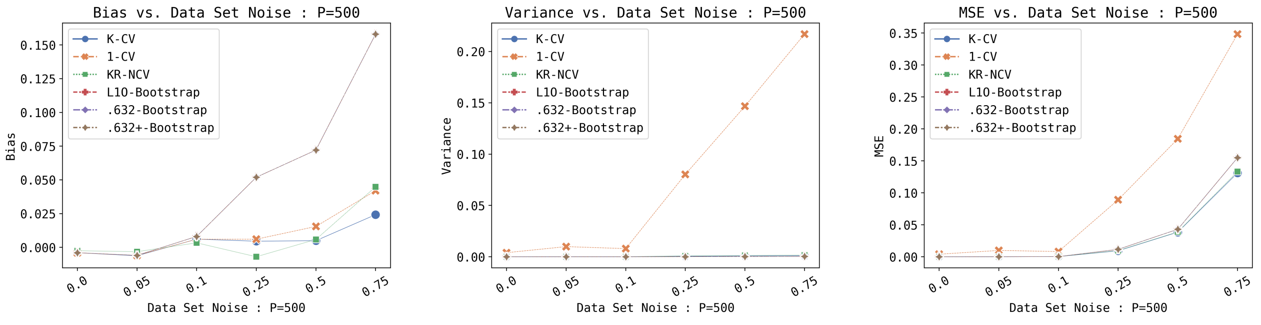


(a) Bias vs. Data Set Size

(b) Variance vs. Data Set Size

(c) MSE vs. Data Set Size

Fig. 2: Effect of data set size on bias, variance, and MSE respectively



(a) Bias vs. Data Set Noise : P=500

(b) Variance vs. Data Set Noise : P=500

(c) MSE vs. Data Set Noise : P=500

Fig. 3: Effect of data set noisiness on bias, variance, and MSE respectively

Through the pre-determined decision boundaries, it was possible to arrive at a value for the irreducible error in each data-set by dividing the number of misclassified points by the size of the sample space. Finally, in order to test the effect of dimensionality, the Gaussian blob generator was extended to retain a decision boundary along the primary axis in N dimensions.

3.2.2 Experimental Setup

The hyperparameters as defined in 3.2 were tested independently in sequence. Each experiment was repeated 30 times, with new datasets drawn, and the mean accuracy and variance were obtained for the experiment. Bias and Mean Squared Error were then computed as outlined in 2.1. A summary of hyperparameters tested is given below. All experiments excluding the data-set size were repeated for $P = 100$ and $P = 500$ to account for the size as a potential confounding factor.

Data-set Size	10, 20, 50, 100, 200, 500, 1000
Noise	0, 0.05, 0.1, 0.25, 0.5, 0.75
Data-set Types	Squares, Circles, Blobs
Classifiers	K-NN,SVM-l, SVM-r, LVQ
Dimensionality	Range - [2,50]

4 Results

The results obtained from the experiments run have been represented graphically for ease of interpretation. In order to demonstrate the behaviour of bias and variance, their graphs have been shown in addition to the principal criterion of Mean Squared Error for the first and second experiments. The effect of data set size on execution time has also been shown.

4.1 Data-Set Size Experiment

The effect of the data-set size P on the bias of the validation technique, as P varies from 10 to 1000 is represented in figure 2a. An ideal validation technique is expected to have a Bias approaching 0.

Similarly, the effect of P on the variance of the generalisation error estimates has been captured. A value approaching 0 is considered to be ideal. See figure 2b.

The effect of P on the primary evaluation metric, i.e. Mean Squared Error has been recorded as in figure 2c. Given the derivation of Mean Squared Error as a function of Bias and Variance, an ideal validation technique should have an MSE approaching 0.

The time taken (in seconds) per iteration of each validation technique has been captured as below. Lower values per iteration are generally preferable.

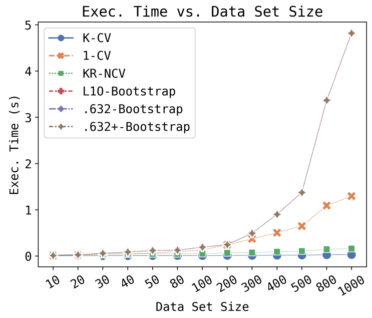
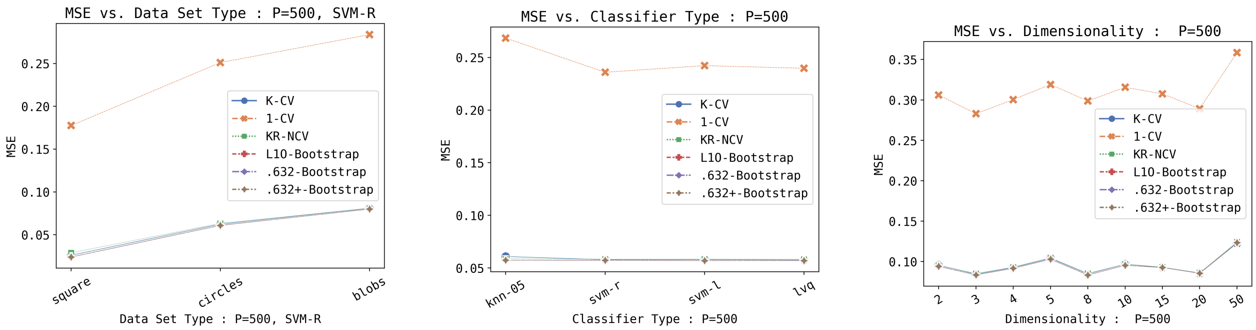


Fig. 6: Effect of data set size on execution time

4.2 Noise Experiment

In this experiment, the effect of irreducible error, i.e. noise within the data-set on the validation techniques was captured. Similar to 4.1, the effect of noise on the Bias, Variance, and Mean Squared Error has been captured for $P = 500$. A minor skew in favour of Bootstrap was observed in MSE compared to this for $P = 100$.



(a) MSE vs. Data Set Type : P=500, SVM-R

(b) MSE vs. Classifier Type : P=500

(c) MSE vs. Dimensionality : P=500

Fig. 7: MSE of remaining experiments

4.3 Data-Set Type Experiment

For each Data-Set generator as defined in 3.2.1, the validation techniques were tested using the Support Vector Machine as a classifier with a Radial Basis Function kernel. The results for $P = 500$ have been represented in figure 7a. The results for $P = 100$ were similar, with a minor reduction in MSE for Bootstrap.

4.4 Classifier Experiment

The effect of the classifier in use was tested with the Gaussian Blobs generator with an irreducible error ~ 0.25. The results for $P = 500$ are depicted in figure 7b.

4.5 Dimensionality Experiment

Finally, the effect of data set dimensionality was tested with the Gaussian Blobs generator and SVM with a linear kernel as classifier. The results for $P = 500$ are shown in figure 7c.

5 Discussion

From the results of the experiments, a number of patterns made themselves apparent, addressed in brief below.

5.1 Data-Set Size Experiment

This experiment was performed on data sampled from the Box distribution defined in 3.2.1, with a Noise Ratio of 0.4, resulting in irreducible error ~ 0.25. From the graph of Bias vs. Population Size in 4.1, it may be observed that in scenarios where the available data-set is of limited size, all validation techniques exhibit significant downward bias. This is notably consistent with the downward bias described by Efron, Tibshirani et al. [3] for bootstrap techniques. This downward bias reduces in magnitude as P approaches larger values, as observed when $P \to 100$. However, as P increases beyond 100, bootstrap validation techniques begin to exhibit upward bias, while cross validation techniques continue to demonstrate downward bias with smaller magnitude.

5.1.1 Performance measured by Mean Squared Error

Considering Mean Squared Error as the primary evaluation metric, it would appear that both Cross Validation and Bootstrap techniques are able to produce comparable results, with the notable, consistent exception of Leave-One-Out Cross Validation, which shows characteristic signs of overfitting, and performs objectively worse than all other techniques. It may also be noted that when $P \leq 100$ Bootstrap techniques produce a small but measurable reduction in MSE compared to Cross Validation. Additionally, it is observed that Nested Cross Validation is able to achieve marginally lower Bias for large datasets, approaching 0, with low Variance, indicating high stability of results.

5.1.2 Performance by Execution Time

The graph of Execution Time vs. Population size indicates superlinear growth of the execution time for Bootstrap techniques, approaching $O(n^2)$ in O-notation, while Cross validation techniques appear to experience growth in execution time in $O(n)$. This behaviour would strongly discourage the use of Bootstrap techniques for validation on larger data-sets.

5.1.3 Data-Set Size as a Confounding Variable

The effect of data-set size on the performance of the validation techniques was significant enough to affect the results of the remaining experiments. It was observed in each case that the results pr $P = 100$ showed marginally better performance for Bootstrap techniques over Cross Validation, which was not so at $P = 500$. The latter set of results proved to be less biased based on the results of the data set experiment, and were thus favoured over $P = 100$.

5.2 Noise Experiment

Using the Box distribution generator, with Noise Ratios $\in [0, 0.75]$ the resulting irreducible error was varied between 0 and 0.45. While it may be noted that the Bias exhibited by the Cross validation techniques oscillated with increasing amplitude around $Bias = 0$, the Bias exhibited by Bootstrapping techniques rose sharply at a Noise Ratio of 0.25, representing around 15% nominal misclassification. This suggests that the estimates produced by Bootstrapping techniques may be overly optimistic for noisy data, and thus may not be an appropriate choice. This observation is also reflected in the graph for Mean Squared Error vs. Noise Ratio, where Bootstrapping and Cross validation techniques diverge.

5.3 Data-set Type Experiment

In order to account for the effectiveness of the classifier as a potential confounding factor, for the purposes of this experiment, the Support Vector Machine was chosen, with separate sub-experiments for the Linear kernel and the Radial Basis Function kernel.

It can be seen that in the case of the linear kernel the Mean Squared Error values converge in the case of Box and Blob distributions, but appear to diverge in the case of the Circle distribution. Upon examination of the graph for the Radial Basis Function however, there is near complete convergence.

This provides strong evidence to support an argument that the nature of the data-set is immaterial to the choice of validation technique.

5.4 Classifier Experiment

It can be observed that there is little variation in the results of each validation technique due to the classifier itself, suggesting strongly that each technique is equally valid for all classifiers, once again, with the exception of Leave-One-Out CV.

5.5 Dimensionality Experiment

Similar to the results of the Data-Set and Classifier experiments, the behaviour exhibited by both sets of validation techniques is near identical, with the exception of Leave-One-Out CV.

5.6 Practical Implications

From the results, a simple heuristic arises, that bootstrapping techniques provide more accurate estimates of generalisation error in low sample count and low noise scenarios, with factors such as the choice of classifier, or the distribution of data being immaterial to the choice of validation technique.

From the chosen bootstrap techniques, .632+ provides marginally better Mean Squared Error, and is well justified mathematically [3]. However, its quadratic complexity highly discourages its use for large datasets. Among the chosen Cross validation techniques, it was found that Nested Cross Validation provides measurable improvement in terms of Mean Squared Error over K-Cross Validation, at the expense of higher linear computational load, which may prove a worthwhile trade-off.

Finally, Leave-One-Out Cross Validation performs objectively worse in all scenarios due to extremely high variance and high computational load, in line with the findings of Borra et al [1].

The results so obtained may thus be distilled into a simple 2 × 2 matrix, with data set size and noise as key parameters.

	Low Noise	**High Noise**
Small Sample Size	.632+ Bootstrap	NCV
Large Sample Size	NCV	NCV

Table 1: Selecting an appropriate validation technique

5.7 Comments on Theoretical Conformance

In their work on the .632 bootstrap [4] Efron and Tibshirani noted that the asymptotic performance of Bootstrapping and Cross validation techniques would be identical, that Cross validation techniques would generally deliver lower bias, but higher variance, and that Bootstrap estimates would tend to have a downward bias while Cross validation techniques would remain consistent.

The results of the experiment largely validate the first two assertions as the Mean Squared Error estimates for both classes of techniques converge for large P, the variance of Cross validation techniques are slightly larger compared to Bootstrap techniques throughout the experiments, and the bias observed with Cross Validation is consistently below the Bootstrap estimates for large P.

The third assertion, however, appears to be falsified by the experiments for large P, where Bootstrap estimates are consistently optimistic, with positive Bias, while Cross validation techniques are consistently pessimistic. Only when $P < 100$ does the downward bias for Bootstrap techniques become apparent, and there also, it is observed that Cross validation techniques provide more pessimistic estimates. The latter may be explained by virtue of Cross validation techniques effectively reducing the number of samples used for training in any given iteration of the validation process, which would cause the model to under-train at small P.

6 Conclusion

This paper presented a framework for the evaluation of two classes of validation techniques in their ability to estimate the generalisation error of a given classifier, namely Bootstrapping and Cross Validation. Three metrics, Bias, Variance, Mean Squared Error were evaluated subject to changes in data-set size, extent of noise, nature of the distribution within the data-set, the classifier used, and the dimensionality of the data-set.

Through this framework, it was observed that there is a significant deviation in performance between these two classes of techniques with respect to the data-set size, and the noise, but not with respect to the distribution, dimensionality, or the classifier.

It was further observed that Bootstrapping techniques demonstrate a notably lower Mean Squared Error when the size of the data-set is small, and the noisiness is low. In other cases, Cross validation techniques provided lower Mean Squared Error.

A practical guideline for the selection of an appropriate validation technique to estimate generalisation error was thus created, encapsulated by the table presented in Table 1.

7 Future Work

The evaluation of the two techniques was of an exploratory nature. Thus, while the results obtained appear to provide strong evidence to suggest the use of one technique over another given knowledge of a data-set's sample size and noisiness, these results would require formal statistical hypothesis testing in order to ensure conclusiveness.

While the comparison of these validation techniques was performed for a number of parameters, the effect of ratio between the two classes in the system was not observed. Similarly, the applicability of these techniques in multi-class and regression problems was not explored. These may prove to be worthwhile lines of further research due to the endemic nature of such problems in practical application.

Acknowledgements

The authors wish to thank Prof. M. Biehl for his peerless instruction that inspired us to pursue the subject of this paper. They would also like to thank C. Fernandes, W. Meijer, and J. Majumdar for their invaluable feedback.

References

[1] S. Borra and A. Di Ciaccio. Measuring the prediction error. a comparison of cross-validation, bootstrap and covariance penalty methods. *Computational Statistics & Data Analysis*, 54(12):2976–2989, 2010.

[2] U. Braga-Neto. Error Estimation for Classification. In *Fundamentals of Pattern Recognition and Machine Learning*, pages 151–183. Springer International Publishing, Cham, 2020.

[3] B. Efron and R. Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548, 1997.

[4] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. Chapman & Hall/CRC, 1998.

[5] T. Hastie, R. Tibshirani, and J. Friedman. Model Assessment and Selection. pages 219–259. 2009.

[6] H. Jaeger. Machine Learning - Lecture Notes. 2021.

[7] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning with applications in R*. Springer, 2014.

[8] J.-H. Kim. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, 53(11):3735–3745, 2009.

[9] MrNuggelz. Mrnuggelz/sklearn-lvq: Sklearn compatible implementation of generalized vector quantization algorithms.

[10] S. Ounpraseuth, S. Y. Lensing, H. J. Spencer, and R. L. Kodell. Estimating misclassification error: a closer look at cross-validation based methods. *BMC Research Notes*, 5(1):656, dec 2012.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] P. Schneider, K. Bunte, H. Stiekema, B. Hammer, T. Villmann, and M. Biehl. Regularization in matrix relevance learning. *IEEE Transactions on Neural Networks*, 21(5):831–840, 2010.

[13] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

Isolation Techniques in Software Verification at Runtime

Antonin Thioux, Patrick Lindner

Abstract—We live in a world that expects software systems to constantly evolve through updates, extensions, hotfixes, and maintenance. Simultaneously, society relies upon software systems to function more than ever and expects software to run correctly with minimal bugs. The combined expectations of correctness and evolution of software have led to an increase in the need for software runtime testing. Runtime testing provides a solution to both expectations of software without undermining its volume and depth of test cases. Additionally, runtime testing executes tests in realistic environments, unlike testing done at the development, which suffers from testing in lab-like environments. These lab-like environments problematically eliminate most of the disruptive factors which affect software.

Five isolation approaches were introduced in the survey "A survey on runtime testing of dynamically adaptable and distributed systems" by Mariam Lahami and Moez Krichen. Their paper suggests that these isolation approaches can improve system runtime testability. Specifically, exploring the following approaches: Built-In-Test, Aspect-Based, Tagging Components, Cloning Components, and Blocking Components.

Through a synthesis of research papers on these five approaches, our research details and evaluates different isolation approaches against each other and found that each technique presented unique advantages and disadvantages. We found that the analyzed approaches can be divided in two groups. The built-in approach, the aspect based approach and the tagging approach require the implementation of the code to be modified, while the blocking and the cloning approach do not. Rather, the latter require the deployment of the system to be adjusted. Additionally, we found that the mentioned isolation techniques differ in resource requirements, implementation complexity, and limitations.

Index Terms—Runtime Testing, Test Isolation, Software Verification

1 Intoduction

The current digital society we inhabit is hugely dependent on its underlying software support. We expect this software to execute correctly at all times with minimal bugs and errors. During the development of this software: unit tests, integration tests, and system tests all ensure that written code meets its requirements. However, these development tests are applied in lab-like environments where most disruptive factors are eliminated. While these tests mostly focus on finding errors in code implementation, they can not find errors related to the software's deployment in the production environment. The production environment of a software system mostly differs from the development and the test environment in an external system and library dependencies. Therefore, unknown bugs and errors, related to these dependencies, might occur. Some errors might not even be detected right away after deployment. For instance, pointer or memory allocation/disallocation defects are most often detected when the system has been deployed and running for a certain time [10].

Furthermore, modular systems, where components can be added, removed, and updated during runtime, have to be tested after a reconfiguration in order to assure a working system. Testing the reconfiguration is impossible during development time since it happens during runtime and future behavioral/structural evolutions are unknown at development time [5]. Applying runtime tests can verify the correctness and availability of running software in a production environment. Since they are conducted in the production environment, the actual integration of the system is tested in the same setting in which the end-user uses it.

Test sensitivity is used to describe to which extent software can be tested without enduring side effects. Low test sensitivity is desirable to perform runtime testing [5, 4]. To improve a system's test sensitivity, various isolation techniques have been developed. Isolation techniques allow a system to split test processes and business processes [5, 4].

This research takes a deeper dive into the existing literature on isolation techniques to provide a detailed analysis and evaluation. Specifically, the following research questions are answered in this research.

RQ1 What are the runtime isolation techniques, and how are they implemented?

RQ2 What are the strengths and weaknesses of the approaches?

RQ3 Which isolation techniques are more appropriate for which system types?

These questions are answered with an in-depth literature review. Various researches have introduced different isolation techniques (i.e. Build-in [10], Aspect Based Approach [8], Tagging Approach [11], Cloning/Blocking Components [5]) which we analyze. The analysis of the different techniques mainly focuses on the effectiveness and meaningfulness of their results, as well as on their resource intensity. Furthermore, strengths, weaknesses, and limitations are part of the analysis. From this analysis, we fill in gaps in the research by suggesting different isolation techniques for different system types based on the results of the analysis.

The paper is structured as follows. In Section 2 we give background information on runtime testing and dynamic system updating. Section 3 introduces the method we used to conduct the research, while Section 4 introduces runtime testing isolation approaches. In Section 5 we discuss the different approaches and present the results of our analysis. Section 6 concludes the paper.

2 Background

In this section, we outline some background information related to the research field.

2.1 Runtime testability

Runtime testing refers to all testing activities conducted on a system running on the target/production environment, while still being able to execute production work [5]. This form of testing is, next to development testing, very important in current software systems. Development tests are focused on verifying the correct implementation of the produced software. Therefore, they are executed in lab-like environments where most disruptive factors are eliminated. These environments differ in hardware as in dependencies to other components and

- *Antonin Thioux is with University of Groningen: S3791378, E-mail: a.p.thioux.student.rug.nl.*
- *Patrick Lindner is with University of Groningen: S4895851, E-mail: p.m.lindner@student.rug.nl.*

Approach	Paper Title	Relevant	Used
Build-In-Test	Principles of Built-In-Test for Run-Time-Testability in Component-Based Software Systems [10]	Yes	Yes
Aspect-Based	AOP-based Testability Improvement for Component-based Software [8]	Yes	Yes
Tagging Components	Automating Integration Testing of Large-Scale Publish/Subscribe Systems [11]	Yes	Yes
Cloning Components	Dependency Isolation for Thread-based Multi-tier Internet Services [2]	No	No
Blocking Components	None	-	-

Table 1. Summary of Lahami's cited papers

systems to the target environment, where the software system will be deployed to execute its productive work. So the software components might exhibit different behavior in the target environment than in a development test.

Runtime testing is supposed to cover errors that occur in the target environment, but not in the development test. In a dynamic software system, for instance, where components can be added, removed, and updated during runtime, a reconfiguration can introduce unknown errors [5]. This is relevant especially with the rise of Dynamically Updatable Systems [1], where continuous availability is vital. Additionally, software components with improper or competing resource handling can introduce major resource bottlenecks after a certain amount of uptime [10].

However, periodically testing a running software system and monitoring the resulting outcome can help to ensure that a provided service is available to the customer. These results might be logged to be used in audits for availability critical systems.

To ensure that runtime testing is available and safe in a running software system, isolation techniques can be used.

2.2 Related Works

Most of the work done on defining isolation techniques was done by Mariam Lahami and Moez Krichen [5, 4]. However, their work does not provide an in depth comprehension of the different techniques, nor does it compare the different techniques to one another.

3 METHOD

We performed a literature review to answer our research questions. The research started with the paper "A survey on runtime testing of dynamically adaptable and distributed systems" by Mariam Lahami and Moez Krichen. Their survey introduces the five isolation approaches that we explore in this paper. These are Build-In-Test, Aspect-Based, Tagging Components, Cloning Components, and Blocking Components. We expanded our research in two ways. Firstly, we explored papers cited in Lahami and Moez's work regarding isolation techniques. Secondly, we searched for relevant scientific papers using search engines.

Table 1 summarizes the papers cited in Lahami and Moez's work. The three citations for Build-In-Test, Aspect-Based, and Tagging Components were relevant to our research. Our literature review excluded the citation for Cloning Components, as it did not contribute to answering our research questions. Lastly, the Blocking Components approach had no cited literature.

To find relevant scientific literature, we followed the following steps.

1. Search for literature through GoogleScholar and SmartCat with the following keywords "(**runtime** OR **testing** OR **validation** OR **isolation strategies**) AND (**build-in** OR **aspect-based** OR **tagging** OR **blocking** OR **cloning**)."

2. Read the abstracts of the papers found.

3. Exclude papers based on the abstract if they aren't relevant or not sufficiently different from other papers found.

4 INTRODUCTION OF ISOLATION APPROACHES

In this section, we answer **RQ1** by introducing five different isolation approaches.

4.1 Build-In-Test Approach

The Build-In-Test (BIT) isolation approach has been introduced by Vincent et al. [10]. Its main concern is to ensure that components are implemented correctly. Additionally, the correct integration of a component in the whole system can be verified using this approach. This is achieved by providing test functionality to a component, which can be invoked during runtime.

The structure of a built-in test environment is depicted in Figure 1. This figure is a simplistic version of a figure from J. Vincent, G. King, P. Lay, and J. Kinghor [10].

The BIT approach is supposed to be a standardization on how to structure runtime-testable components. Such a component (BIT-Component) consists of the component under test itself, a BIT configuration interface (CIF), and optional test modules. The BIT configuration interface is mainly concerned with managing test modules and proper error handling. The test code itself will be implemented in the test modules, where every module is concerned about a different test target.

The testing infrastructure, outside the BIT-Component, comprises a handler and optional external testers. The handler is concerned about handling system-level errors properly to prevent the system from crashing. The external tester is a component where actual assertions on the system level take place. It receives calls from a test module of a BIT-Component, evaluates the results, and takes suitable action in case of an error.

J. Vincent, G. King, P. Lay, and J. Kinghor [10] introduced a number of testing targets suitable for the BIT approach. These targets include: *deadlock testing*, *testing for residual defects*, *user conformance testing*, *code integrity checking*, *data integrity checking*, *resource monitoring*, *real-time constraints*, and *tracing*. Each of these testing targets can be added to the testing environment by adding its respective testing module to the BIT-Component.

Third-party software components, implementing this approach, can easily be tested during runtime without exposing their source code. This is beneficial for both, the provider of proprietary software components and the user, who builds a software system from the proprietary component. The provider does not have to ship their source code, and the user can verify that the proprietary component is integrated correctly and works properly during runtime.

4.2 Aspect Based Approach

One of the suggested ways to improve the runtime testability of software is through an Aspect Based Approach. Aspect Based Approach to testing leverages Aspect-Oriented Programming (AOP) to facilitate the creation of relevant tests, provide self-checking options, and detect faults at runtime. AOP is a paradigm that encourages the separation of concerns (such as logging, self-checking, and error handling) from the actual business logic through aspect code, see figure 2. This separation prevents the cluttering of business logic. Pointcuts are inserted

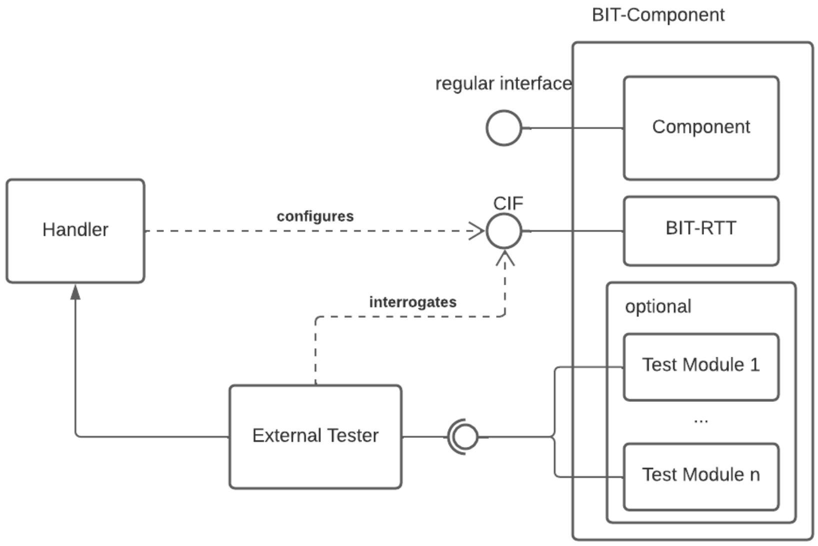


Fig. 1. BIT Environment

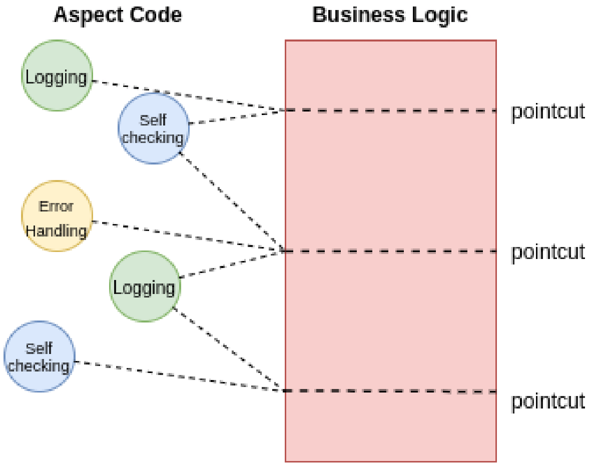


Fig. 2. Aspect Based Overview

into the business logic which weave the aspect and business code together. AOP works on top of other programming paradigms and does not replace them.

As mentioned, one of the goals of the Aspect Based Approach is to improve the tests made for components. Often, component/system testers do not have enough information to create good tests. Using AOP's logging capabilities to enhance the observability of components gives the testers more information about their internal and external behavior through logs. Testers can use these logs to inform their decisions when creating tests [8].

Another goal is to provide runtime self-checking through AOP. Execution invariants are monitored at runtime execution to achieve this goal [8]. In the cases of an external component provider, this provider will make these invariants test, since they are familiar with the code structure. Monitoring execution invariants is not unique to AOP. However, AOP does provide reduced coupling between maintenance and business code. Which in turn leads to simple code evolution based on running phase requirements.

The last goal of the Aspect Based testing Approach is to detect faults at runtime [7]. AOP can be used to again separate business code from error/exception handling code, which allows error handling in a modular fashion and reduces lines of code related to exception handling by a factor of four [6]. However, for complex error/exception handling, the Aspect Based Approach does more harm than good [9].

4.3 Tagging Approach

The tagging approach has been introduced by [11] for runtime testing event based systems safely. Executing runtime tests in a production environment can make unwanted changes to stateful components or have side effects beyond system boundaries. To tackle this problem, the tagging approach marks input data, which are supposed to runtime-test the system, with a testing flag. The event based components can subscribe to these flagged events and process them accordingly. All data resulting from a flagged input will also be flagged. Testable components are supposed to expose a configuration interface which can toggle the subscription to test events. Since the component is aware, that it is processing test data, it can adapt its behavior accordingly. Components with this ability are called *test-aware*. A stateful component for instance, whose output is generated based on a certain state, would keep a second testing state for the tests to prevent test data altering the production state. A component with side effects to the "outside world" would suppress the effect in a test scenario or start a simulation instead. Components which are not stateful and do not have any side effects to the outside world are called *test-insensitive*. These components can be used unmodified when runtime tests in the production environment are conducted.

In Figure 3 the environment of the tagging approach is visualized. Components with a red background are subscribed to incoming testing data. The figure describes an update in component C to C'. This change can now be verified by subscribing all components except C to the incoming testing data. While the reconfiguration of the environment is tested using component C', production data can still be processed by component C.

4.4 Blocking and Cloning Components Approach

The following section introduces both blocking and cloning isolation techniques. The most high-level description of blocking components refers to the practice of stopping a component at runtime to perform tests [5]. Cloning refers to duplicating a component at runtime to perform tests on the identical clone [5]. Blocking and cloning are similar in that neither one is running tests on a component at runtime. In a proceeding "Towards Self-Testing in Autonomic Computing Systems" by Tariq King, Djuradj Babich, Jonatan Alava, and Peter Clarke, both blocking and cloning isolation techniques have been proposed extensions of Self-Management systems.

Self-Management systems manage themselves through self-configuration, self-optimization, self-healing, and self-protection. They follow a basic workflow to achieve these properties, Monitor → Analyze → Plan → Execute. A natural extension of these properties would include self-testing. Either safe adoption or replication with validation is used to achieve this [3], the former mapping to blocking and the latter mapping to cloning.

The blocking isolation technique was preferable when it is too expensive, impractical, or impossible to clone components [3]. When using the blocking isolation technique, resource managers (or touchpoints) perform tests on the actual deployed resource component. Resource managers employ the following steps in testing. First, they bring the Self-Management system to partial operation by blocking the resources in testing. Second, they keep the system in a safe state while tests are performed. Next, after running the tests on blocked components, these are unblocked, and the system returns to full operation. Lastly, the state of the system is returned to a running state.

The cloning isolation technique allows the testing of resource components without stopping or bringing the system to partial operation. Additionally, testing systems do not need to impact the system's performance if the testing is performed on a different node. However, the downside is that the cloning approach endures a high overhead cost caused by the generation or maintenance of the component clones [3]. The only difference between cloning and blocking is that, in cloning, resource managers (touchpoints) do not interact with the original components. Instead, they interact with clones that have either been created for testing or maintained for testing over time.

Lastly, it is necessary to mention that in their paper [3] Tariq King, Djuradj Babich, Jonatan Alava, and Peter Clarke suggested that both blocking and cloning isolation strategies can coexist in the same system. By creating and using validation policies on a resource component basis, a system administrator can decide which approach applies better to each component. Overall, implementation on top of a self-managing system of both isolation strategies is easy.

5 Discussion

In this section, the introduced approaches from section 4 are being compared and discussed answering **RQ2** and **RQ3**.

Generally, the five given approaches can be divided based on the modification required to be executed. The built-in, aspect-based, and tagging approach requires the production code to be changed. The blocking and cloning approach, however, do not require the code to be changed. Rather, a reconfiguration of the deployment needs to be undertaken. Changing the code is generally more complex and expensive than a reconfiguration of the deployment.

5.1 Evaluation of Build-In-Test Approach

In this section, the built-in test approach will be evaluated. In section 4.1 we stated the large number of testing targets covered by this approach, which is a huge advantage. A testable unit in the built-in approach can be as small as it is defined by the developer. This is advantageous over the tagging approach, where a testable unit must have the ability to process events and is therefore limited by the system design. Furthermore, it can be implemented very near to the hardware and allows a straightforward option to conduct tests that include more than one component, like deadlock testing. Hence, built-in testing offers more than pure function testing based on test input data.

Additionally, continuous monitoring of different targets can be implemented using this approach. Thanks to its configuration interface and modular structure, testing targets can be added and removed from BIT-Components easily. A downside of the built-in approach is the strict and complex structure, the BIT-Components have to retain. Additionally, test code is included in the production software artifact, which could introduce a larger chance of bugs if not (development-) tested carefully.

5.2 Evaluation of Aspect Based Approach

Since the Aspect Based Approach is more of a paradigm that can be used to improve runtime testability rather than an actual strategy, it is hard to evaluate it compared to the 4 other strategies. On its own, the aspect-based approach is often not enough to perform runtime testing. However, it can be combined with other isolation strategies to further improve the testability of a resource component. The aspect-based approach shines when performing runtime testing on externally provided components, since it is a proven way to effectively transfer information from the component manufacturer to the component user. This information is especially useful when the component user is selecting or creating their test suite based on the functionality that they expect from the external manufactured component.

If a component is already created, it is not advised to try and improve its runtime testability by using an aspect-based approach, as refactoring the code to accommodate this new paradigm will take a lot of time. However, if a component manufacturer is in the process of creating a new component, using the aspect-based paradigm will give a clear separation between business code and testing code, which will decrease technical debt.

Additional functionality such as logging, self-checking, fault-detection can be added to the business code with relative ease, which can lead to even further potential to improve runtime testability. The downside of the aspect-based approach is that it puts a burden on the component users to use additional information given through logging and exception tracking. In cases where the user is not expected to engage with additional information productively, it would be better to use a built-in test suite and avoid flooding the component users with irrelevant information.

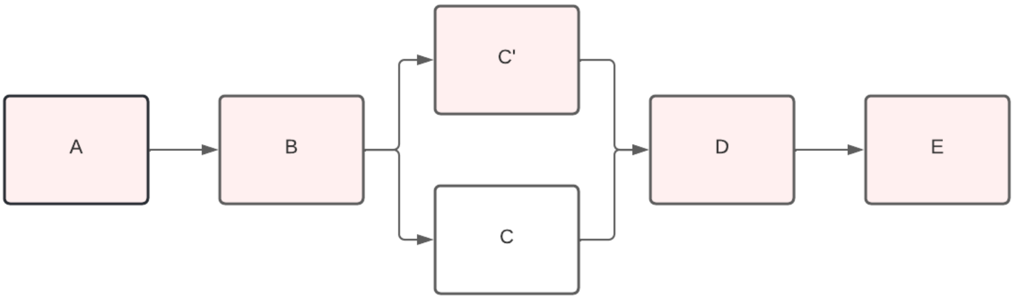


Fig. 3. Tagging Approach

5.3 Evaluation of Tagging Approach

The tagging-based runtime testing approach includes testing code in the production software artifact. It is supposed to be less hardware near than the built-in approach due to its abstraction to event processing. Furthermore, it might be easier to implement due to the absence of a predefined structure of the testing component and the testing environment. Instead, the components under test are loosely coupled through their event-based nature. The only change of implementation required in the tagging approach is some change of behavior for stateful and side-effect-producing components.

In the tagging approach, the actual deployed functionality in the production environment is tested using flagged test data. That means it can not be utilized for monitoring.

Starting a tagging-based runtime test might be beneficial, especially after a component has been added, removed, or modified, to ensure that the system still works and the change of configuration has not introduced any unwanted side effects. However, like the built-in approach, the tagging-based isolation approach requires the production software artifact to include testing code, which increases the level of code complexity along with the chance of bugs.

5.4 Evaluation of Blocking Approach

Strictly speaking, this method is not an isolation approach, since the system stops its production data processing. However, it describes testing a running system that is deployed in its target environment and is therefore included in this paper. Using the blocking approach for runtime testing means stopping the production service of the system (or component) until the test is completed. To make the most use of this technique, tests should be started when the system gets statistically fewer production requests. For a software system deployed for a single time zone, this would mean starting the test during nighttime. Additionally, software systems that are deployed completely redundant can transfer the production requests to the backup system while the primary system is runtime tested. This way, the backup takeover is tested simultaneously. If only a certain component of the system should be blocked and tested, it is advised to also have a backup component, which can take over the production workload during the test to reduce the downtime of the system.

Compared to the built-in, aspect-based, and tagging approach, the blocking approach can be seen as the purest form of runtime testing, since no test code is deployed to the production environment. Additionally, developers do not need to be concerned about including test functionality in their software or sticking to a strict and complex structure. Instead, the production code “as-is” will be tested in the exact target environment where production data is processed. The disadvantages of this approach include the absence of monitoring and resource testing. Other isolation approaches can measure and trace the resource extensiveness of software components during a high workload measured over a long period. The blocking approach can not provide such measurements and traces, since a test executed during blocking of other requests will only be a snapshot of the current state of the system. Since this approach blocks production processing, it is suited best for systems with dependencies to hardware components, like robotic arms for instance. The whole system “as-is” can be tested in the production environment, and test data will be treated exactly as production data by the system. Therefore, test results of this approach are more meaningful than results from other approaches.

5.5 Evaluation of Cloning Approach

The cloning approach is just an extension of the blocking method. It allows the system to be tested during runtime, without having to stop the production data processing. Therefore, the whole system (or component) under test, will be duplicated including their state, and be responsible for running the test while the original system proceeds its normal work. Like the blocking approach, the cloning approach enables runtime testing, without providing testing code in the component artifact. Additionally, it allows testing software components at any time without being concerned about blocking services. A downside of this approach is the huge resource consumption, as the system (or component) under test requires additional resources to be executed since it is not executed in the production environment directly.

Compared to the blocking approach, there is one significant disadvantage: Hardware components, like robotic arms for instance, cannot be cloned like software components. The cloned system therefor needs to be deployed in an environment which simulates or omits calls to such hardware components. Hence, systems without dependencies to non-duplicatable hardware are more suitable for this approach. Note that a cloned component including its state is not deployed in the actual production environment, which could introduce biases in the testing result, rendering it less meaningful. Testing availability of the service to the customer is not possible using this technique, due to a different testing environment. Container based applications executed by an orchestrating platform nowadays are extremely easy to clone. By taking a snapshot of the current container, the running system, including its state, can be cloned and deployed on a different machine. No reconfiguration of the production system has to be done during a test, rendering this approach minimal invasive to the production environment.

The cloning approach of all isolation techniques is the least complex to implement. It purely relies on the deployment configuration and does not infer the code implementation and the runtime environment.

5.6 Overview

To provide a clear overview of the evaluation, Table 2 is provided. It describes multiple properties of the approaches. The column “*Modification*” provides an overview on which modification has to be done in the software setup to use the respective approach. The “*Complexity*” column depicts the complexity of actions, which need to be performed to set up runtime testing. It is rated on a scale from 1 to 4. Approaches with complexity 4 are supposed to have the most complex setup. Last, the “*Suitable For*” column gives an overview of system types and deployment models, which are more suitable for the respective isolation approach.

6 Conclusion and Future Work

In the scope of this research, **RQ-1**, **RQ-2**, and **RQ-3** have all been answered. Five runtime testing isolation approaches have been identified and analyzed. It has been found that some isolation approaches are more suitable for specific types of systems and deployment structures than others. The paper contributes to the runtime testing research field by assembling relevant information on different isolation approaches in one single paper. Furthermore, the introduced approaches have been analyzed and compared to each other, which may support a decision-making process when choosing a runtime isolation approach.

A limitation of this research is the focus on the isolation approaches, introduced in the work of Mariam Lahami and Moez Krichen [5]. Hence, the research might not include all possible runtime testing isolation approaches. Furthermore, the analysis is based on synthesized knowledge from the cited papers, rather of real-world experiments. Future research could focus on an experimental evaluation of the introduced runtime testing isolation approaches. Hereby, every approach could be applied to the same simple code project in order to support or refute our analysis results.

Acknowledgements

Thanks to Mostafa Hadadian Nejad Yousefi for providing the initial papers to get started with the proposal and suggesting an additional study when we decided to focus on runtime testing. Also, thanks to the reviews that reviewed our paper and gave us quality feedback.

Approach	Modification	Complexity	Suitable For
Built-in	code	4	Hardware-near Systems with low level of abstraction
Aspect based	code	3	Externally provided Components, monitoring
Tagging	code	3	Event based system
Blocking	deployment	2	System with backup deployment, Systems with low workload at certain time points, Systems with non-clonable hardware dependencies
Cloning	deployment	1	System with high availability requirements

Table 2. Overview

References

[1] B. H. Ahmed, S. P. Lee, M. T. Su, and A. Zakari. Dynamic software updating: A systematic mapping study, 10 2020.

[2] L. Chu, K. Shen, H. Tang, T. Yang, and J. Zhou. Dependency isolation for thread-based multi-tier internet services. 2005.

[3] T. M. King, D. Babich, J. Alava, P. J. Clarke, and R. Stevens. Towards self-testing in autonomic computing systems. In *Eighth International Symposium on Autonomous Decentralized Systems (ISADS'07)*, pages 51–58, 2007.

[4] M. Lahami and M. Krichen. Test isolation policy for safe runtime validation of evolvable software systems. pages 377–382. IEEE Computer Society, 2013.

[5] M. Lahami and M. Krichen. A survey on runtime testing of dynamically adaptable and distributed systems. *Software Quality Journal*, 29:555–593, 6 2021.

[6] M. Lippert and C. V. Lopes. A study on exception detection and handling using aspect-oriented programming. In *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, pages 418–427, 2000.

[7] J. Manson, J. Vitek, and S. Jagannathan. Dynamic aspects for runtime fault determination and recovery. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 8 pp.–, 2006.

[8] C. Mao. Aop-based testability improvement for component-based software. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 2, pages 547–552, 2007.

[9] M. Z. Meetei. Aspect-oriented software for testability. In *2012 5th International Conference on BioMedical Engineering and Informatics*, pages 1326–1331. IEEE, 2012.

[10] J. Vincent, G. King, P. Lay, and J. Kinghorn. Principles of built-in-test for run-time-testability in component-based software systems. *Software Quality Journal*, 10:115–133, 2002.

[11] Éric Piel, A. González, and H.-G. Gross. Automating integration testing of large-scale publish/subscribe systems. *Proceedings of the 22nd IFIP WG 6.1 international conference on Testing software and systems*, 2010.

A Review of Transparency and Accessibility in Automated Machine Learning

Igor Pidik – s3358097
Marijn Schokker – s3510239

Abstract—Automated Machine Learning (AutoML) is a field which has been gaining considerable traction over the last decade. The field has several driving factors behind it, such as the prospect of making machine learning tools accessible to all. Additionally, critically considering and automating the machine learning workflow contributes to acceleration of research within the machine learning field itself.

In this work we present a novel framework to put AutoML tools into context when it comes to transparency and accessibility. Ten of the most prominent AutoML tools are then evaluated based on this framework.

It is learned that most of the evaluated AutoML tools are not accessible to the average user. This is problematic since it is one of the main things behind the development of automated machine learning. Additionally, a number of sophisticated AutoML tools are not transparent with the methods they use to derive an optimal machine learning strategy.

As the field of automated machine learning progresses, it is important that developers and researchers must keep accessibility for average users in mind. Similarly, for the field to progress quickly, the inner workings of AutoML tools should be presented in a clear way.

Index Terms—Automated machine learning, AutoAI, AutoML.

1 Introduction

The field of automated machine learning (AutoML) attempts to partly or entirely automate the machine learning workflow. A primary goal of this is to give people with little knowledge of the machine learning field access to powerful machine learning tools. Additionally this could provide valuable insights into the machine workflow itself, and accelerate the research within the field of machine learning because of that.

AutoML has already shown promise for solving machine learning tasks in fields such as health-care [23, 1, 9], astrophysics [16, 20] and international relations [5]. Multiple useful surveys [21, 8, 18, 24, 4]. of the field show that can AutoML tools can also achieve performance close to hand-made models.

However, often there is lack of user trust in AutoML tools [6], and with it comes a need for transparency. El Shawi et al. [19] show that a considerable amount of AutoML tools are not open source projects, which is another detriment to transparency. Wang et al. [22] have acknowledged the importance of transparency in AutoML tools, and consequently developed ATMSeer, a tool to visualize multiple components of the automated machine learning process. However, ATMSeer is directed mainly at expert users, and thus does not contribute much to transparency for other users. Simply visualizing potential model configurations and performance results can go a long way increasing trust of users [6]. Drozdal et al. [6] have delivered excellent work in identifying factors which contribute to trust in AutoML tools, but have not addressed in what sense these factors exist in state-of-the-art AutoML tools.

Increased transparency causes increased the trust of expert users in AutoML tools as discussed by Wang et al. [22] but is also crucial in achieving clarity and accessibility for non-expert users. Consequently, the aim of this paper us to highlight the state of transparency and accessibility within a number of the most well-known AutoML tools.

Firstly, in Section 2 we will define a novel framework suit to rank AutoML tools on accessibility and transparency. Following this, a number of existing AutoML tools are ranked with this framework. The results of this are presented in Section 3. Additionally, specific experiences with each tool are discussed in depth. Finally, in Section 4 we address a number of concerns with our framework, and present a fitting conclusion.

2 Methods

To compare the transparency of different AutoML tools, we define a novel framework. The framework consists of seven different criteria, of which six are dedicated to measuring transparency. The additional criterion *Accuracy* is provided to put the models into context, and highlight possible relationships between performance and transparency.

In addition to the ranking of the AutoML tools based on the framework, we include an in-depth review of each specific tool, where different notable characteristics of each tool are highlighted. This serves to highlight the transparency and accessibility properties beyond the framework.

Now we move on to a description of the framework and all of the criteria that exist within it.

2.1 Framework

Accuracy – This corresponds to a very basic metric to show the performance of the AutoML tool on the MNIST[1] dataset. This aims to put each AutoML tool into context when it comes to performance. A more extensive evaluation of the performance of existing AutoML tools can be found in numerous great surveys of the field [8, 25, 21].

SaaS – This criterion reflects whether the AutoML tool can be run locally, or whether it is provided in the form of Software as a Service (SaaS). An AutoML tool that can be run locally can be run whenever the user desires, and on whatever hardware the user desires. Users can also monitor the activity of an AutoML tool more directly when it is run on a system which they have full access to.

Accessibility – Accessibility is of great importance when it comes to AutoML tools, since one of the goals of the automated machine learning field is to make machine learning accessible to users that do not have a background in IT.s

We define accessibility as a score which ranges from 0 to 2. Within our framework a higher accessibility score means a tool is easier to

- *Igor Pidik is a MSc Student Computing Science at University of Groningen. E-mail: i.pidik@student.rug.nl.*
- *Marijn Schokker is a MSc Student Computing Science at University of Groningen. E-mail: m.j.schokker@student.rug.nl.*

[1] `http://yann.lecun.com/exdb/mnist/`

use. We define a number of different groups of users corresponding to the different levels of accessibility.

0. *Machine Learning Expert* – To operate the tool the user must be an expert in the field of machine learning. This group of users includes for example experienced data scientists and developers.

1. *Developer* – This group of users includes users who have programming expertise, but do not necessarily have experience with machine learning.

2. *Domain Expert* – A domain expert is assumed to have technical expertise, but no real programming experience.

Search Space – When searching for optimal configurations, AutoML tools traverse a so-called search space [14]. This is the space which contains all possible different model configurations for a given AutoML tool.

This criterion then describes whether the tool is transparent with the parameter search space or not. For experienced users especially, it is important to know what configurations will be tested [19], and in what fashion since users can then avoid testing these configurations again.

Final Configuration Known – After traversing the search space, an AutoML tool must settle on a final model configuration. This final configuration is chosen based on a set number of metrics which measure performance. Often various configurations with the best performance are shown to the user.

Here we note whether the precise structure of the final pipeline is disclosed to the user. This is crucial, since it enables the user to re-create the model easily for example. This also enables expert users to test the model separate ouside of the environment of the AutoML tool.

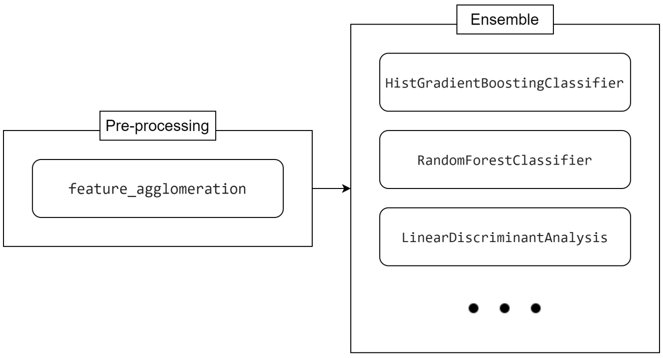


Fig. 1. Example configuration generated by auto-sklearn

Open Source – Open source AutoML tools allow expert users to analyze the internal workings of the code. This helps to progress the AutoML field since experts can learn from the workings of previously designed AutoML tools. Additionally, expert users can modify existing tools to work to their own preference. Thus, this criterion simply notes whether the tool is an open source project or not.

Free To Use – An important part of the accessibility of AutoML tools is whether they are free to use. When testing AutoML tools a number were found to be locked behind a paywall. Thus, this column simply denotes whether the AutoML tool is free to use or not.

2.2 Experimental Setup

Experiments which required us to run any code were conducted on Google Colab[2], which provides free computing resources. Free accounts on Google Colab are equipped with a NVIDIA Tesla K80 with 12 GB of RAM [3]. Auto-WEKA comes as an optional package with the WEKA software and hence could not be run on Google Colab, instead, this software was tested on a 2015 Macbook Pro quipped with 2,5 GHz Quad-Core Intel Core i7 and 16 GB of RAM. Tools which are only available as SaaS, needed to be run on the provider's platform.

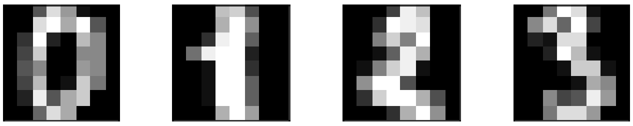


Fig. 2. Sample digits from the training dataset. Each sample has 64 features. A feature is an integer with a value between 0 and 16.

To test the tools, sklearn's digits dataset was used[3], this is often also referred to as the MNIST dataset. The dataset consists of 1797 samples of 10 hand-written digits. Each sample has 64 features (integers between 0 and 16). A few examples are displayed in Figure 2. There is roughly 180 samples per class. Furthermore, the dataset is split into train and test partitions. The test partition corresponds to 25% of the original dataset. Each tool is run for at most 10 minutes and then the best found pipeline is evaluated using the test set and accuracy as the primary metric.

3 Results

First, the results of evaluation using the defined framework are presented. Following this, we include the in-depth experiences of operating the different tools.

3.1 Framework

A total of ten AutoML tools were evaluated using our framework. The results can be viewed in Table 1.

For a number of the AutoML tools, we could not generate accuracy values. The reasons for this will be discussed in later sections. The same holds for two of the *Search Space Known* values.

It can be seen that most of the tools are not accessible to the average user, or as defined our framework; the domain expert. Furthermore, all AutoML tools that were evaluated do disclose the final model configuration.

Notably, most of the AutoML perform quite well on the MNIST data set. MLBox is the only exception to this, achieving a considerably lower accuracy. Furthermore, there seems to be a correlation between the rows *Final Configuration Known*, *Open Source* and *Free To Use*.

3.2 Experiences

Now follows a description of all of the automated machine learning tools, in addition to a description of the experience with using them. The focus is on overall accessibility level, ease of use, and a description of the results and experiments.

3.2.1 auto-sklearn

auto-sklearn[4] is an open source automated ML toolkit, which helps the user with pipeline engineering and hyperparameter tuning. To do so it utilizes techniques from fields such as Bayesian optimization, meta-learning and ensemble construction [7]. auto-sklearn is very easy to run, a classifier can be initialized, trained and used to make predictions in just 3 lines. Furthermore, it provides methods to display the best configuration (which is usually an ensemble of models) in a table as well as showing a leader board of all attempted configurations. Printing statistic overview gives the user the number of configuration considered, number of algorithms that crashed, exceeded the time/memory limits and the best validation score.

However, it does not explicitly state why certain configurations were tried out or why certain ensemble weights were assigned. It is possible to get additional details about each of the attempted runs

[2] https://colab.research.google.com/

[3] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

[4] https://automl.github.io/auto-sklearn/master/

	Accuracy	SaaS	Accessibility	Search Space Known	Final Configuration Known	Open Source	Free To Use
auto-sklearn	0.986	×	Level 1	✓	✓	✓	✓
TPOT	0.988	×	Level 1	✓	✓	✓	✓
AutoKeras	0.942	×	Level 0	✓	✓	✓	✓
PyCaret	0.991	×	Level 1	✓	✓	✓	✓
MLBox	0.593	×	Level 0	✓	✓	✓	✓
IBM AutoAI	-	✓	Level 2	✓	✓	×	×
AutoML-Zero	-	×	Level 0	✓	✓	✓	✓
AWS ML	-	✓	-	-	-	×	×
Vertex AI	-	✓	-	-	-	×	×
Auto-WEKA	0.986	×	Level 2	✓	✓	✓	✓

Table 1. Results of evaluation of ten commonly used automated machine learning tools.

and their results, however, this requires certain level of programming knowledge and willingness to look into their code base or find solutions on the internet. Nevertheless, their documentation is easy to navigate and provides implementation details as well as several examples on how to use the tool.

auto-sklearn's final ensemble reached the accuracy of 0.986 on our withheld test dataset. auto-sklearn requires its users to write code, however, it does not require them to have very deep understanding of machine learning concepts, hence Level 1 accessibility was assigned.

3.2.2 TPOT

Tree-based Pipeline Optimization Tool (TPOT)[5] tries to be a data science assistant. It tries to help its users by building and recommending optimized pipelines to solve any given task. Genetic programming is used to traverse the search space and generate candidate pipelines [13]. TPOT is an open source project built on top of scikit-learn.

Similarly to auto-sklearn, TPOT requires only a few lines of code to train and test a model. Therefore, Level 1 accessibility was assigned. Additionally, it provides methods to list details about all the ran pipelines and their validation scores.

Lastly, TPOT can export the best pipeline by generating a python script. The full pipeline, including the pre-processing and feature engineering steps, as well as the initialzed model with its proper configuration is present in this script. Furthermore, the script can be used to run the pipeline on novel data. However, to make it run, the user needs to edit this script and provide path to the data. This requires some programming and scikit-learn experience, since one should be able to understand what the script does before editing or running it blindly. Nevertheless, this script could be a great starting point for a junior data scientist to learn the best practices related to setting up pipelines to be used on novel data. Moreover, TPOT provides clear and concise documentation with number of detailed examples for various datasets and tasks. On the other hand, TPOT does not provide any easy to use API to visualize various metrics of the model.

TPOT's final pipeline reached an accuracy of 0.988 on our withheld test dataset,

3.2.3 AutoKeras

AutoKeras[6] is an open source automated ML tool, which uses Bayesian optimization theory to morph the topology of neural networks and efficiently explore the search space. As opposed to existing neural architecture search algorithms such as NASNet, PNAS, which commonly suffer from expensive computational cost [10]. AutoKeras is built on top of Keras, a popular deep learning framework[7]. Hence, the API closely resembles the API of Keras. AutoKeras claims that its goal is to make machine learning accessible to everyone. It does a decent job at this. Level 0 accessibility was assigned to AutoKeras.

Firstly, AutoKeras does not do any data pre-processing, the data is fed directly to the artificial neural network. In some cases this may not be required, however, in real world problems the data is often noisy and the algorithm could greatly benefit from normalizing and cleaning the data beforehand. Secondly, the user either has to choose one of the specific models, e.g. ImageClassifier or TextRegressor, or they can use the so-called AutoModel. However, even when the AutoModel is chosen the user is required to specify input type (e.g. ImageInput, TextInput) and output type (e.g. ClassificationHead, RegressionHead). This means that the user is required to have a certain level of machine learning knowledge to be able to understand the differences between them and to chose the correct option. On the other hand, AutoKeras comes with detailed documentation providing a wide range of code examples. This makes AutoKeras an extremely easy-to-use tool for anyone with prior programming and machine learning experience.

After AutoKeras has run, the final model with the best topology and trained weights can be exported. The result is a Keras model instance, which can be handled with the regular Keras API. It can be either used in the pipeline right away or it can be saved to a file and loaded later.

Similarly to the majority of the reviewed tools, AutoKeras lacks a simple API which would help less skilled users review various performance or error metrics of the best model.

AutoKeras' final model reached the accuracy of 0.942 on our withheld test dataset. The performance is slightly lower than that of auto-sklearn or TPOT. Possible reasons for this are discussed in Section 4 of this paper.

3.2.4 PyCaret

PyCaret[8] is an automated end-to-end machine learning tool, which additionally helps with model management. It tries to speed up the development process by automating tasks such as data analysis, data pre-processing, model training, evalution and even deployment. PyCaret allows the user to setup a session with their data and target feature. Then it automatically analyses the data and determines certain properties of the session, such as whether there is class imbalance in the dataset. The user is then prompted to apply methods to solve this issue. The tool suggests principal component analysis (PCA), removing of outliers, normalization and many more. This large overview may seem overwhelming at first, however at the same time it gives a very good overview of the data and possible problems with it. As well as its strategy to fix these problems. Afterwards, the

```
compare_models()
```

method can be used to build up a list of models and compare their performance. It also returns the best model. As opposed to all other tools discussed in this paper, PyCaret includes a DummyClassifier in this overview, hence giving a baseline to which all the other performances can be compared. Furthermore, this feature is very important, since it enables the user to identify skew in datasets and instances of over- or under-fitting. Another unique feature of PyCaret is its simple API to which can visualize various performance metrics. A call such as

[5] http://epistasislab.github.io/tpot/

[6] https://autokeras.com/

[7] https://keras.io/

[8] https://pycaret.org/

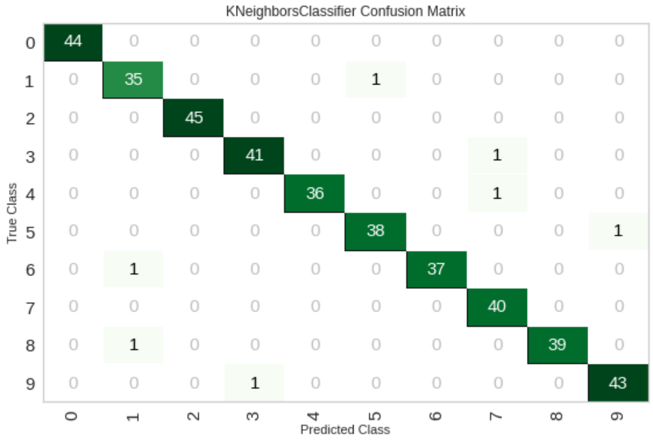


Fig. 3. Result of using PyCaret to plot model's confusion matrix.

```
plot_model(model, plot = "confusion_matrix")
```

will produce results such as the one shown on Figure 3. PyCaret can plot metrics such as the confusion matrix, area under the curve, error rate, feature importance and several others.

PyCaret's best model reached the accuracy of 0.991 on our withheld test dataset. This is the highest obtain accuracy from all of our experiments. Furthermore, PyCaret was very easy to use and get running. On the other hand, a little programming experience and willingness to read PyCaret's documentation was required to efficiently use it.

3.2.5 MLBox

MLBox[9] is an open source Python library for AutoML. It provides a number of features such as distributed data pre-processing, cleaning, formatting, feature selection and leak detection, hyperparameter optimization, models for classification and regression and others [2]. This library requires users to describe their search space in detail. On one hand this provides total transparency with regards to what experiments are being run. On the other hand, a domain expert without much technical knowledge and even a developer without a solid background in machine learning will not be able to define this search space optimally. Because of this, users might spend more time trying pipelines which are bound to fail than testing promising configuration. In light of this we assign Level 0 accessibility to MLBox, since non-trivial knowledge in ML is required to setup the experiments properly.

This library seems to be easy to use when predefined narrative is followed, however, it is very inflexible as soon as the user strays from this narrative. For example: training and testing data needs to be loaded from a file, it is impossible to process an in-memory data frame. After thorough examination of the source code, it became apparent that this is a result of poor component architecture. MLBox's Reader component, does not only read the dataset, but it also fixes missing values, trains the model and saves an encoder. All of this is done in a single massive method. Therefore, if the user skips the file Reader and tries to work with the data already loaded in memory, soon they will be unable to continue since they will be missing a label encoder. This could be easily avoided by either splitting the Reader component into smaller components with a single responsibility or at least splitting the method which does all the steps into smaller methods which can be called separately as required by situation bases. Furthermore, the documentation is lacking severely, therefore deep diving into the code of the library was often a necessity to debug the errors produced by our attempt to get the experiment running.

When testing MLBox it miss-classified the task as a regression, rather than a classification. We were unable to push MLBox to correctly recognize the task, even with attempts such as converting the labels into string digits. Furthermore, no straight-forward option to set the task type was found after examining the source code. Misclassifying the task and approaching it as regression resulted in poor performance. MLBox's best model reached the accuracy of 0.593 on our withheld test dataset. This is the lowest recorded accuracy among all the tools that we were able to run.

Lastly, it should be noted that MLBox does present the user with the best performing pipeline, which then later can be applied to novel data.

3.2.6 IBM AutoAI

IBM AutoAI[10] also known as Watson Machine Learning service, tries to discover candidate pipelines which suit the given problem the best. It automatically performs data cleaning and pre-processing, feature engineering, model selection, hyperparameter optimization.

For model selection IBM AutoAI uses the Data Allocation using Upper Bounds strategy. The goal of this strategy is to sequentially take subsets of the training dataset and apply a large set of classifiers on it [17]. This enables IBM AutoAI to test and select a near-optimal classifier from a large number of classifiers, without the need of training them all on the full training dataset, effectively saving both computational resources and time. When it comes to feature engineering AutoAI leverages algorithms based on [12] and [11] to automate this task. This system is inspired by the usual "trial and error" approach, applied by many machine learning experts. Furthermore, it tries to discover data transformations such as principle component analysis, square or word to vector to improve the performance of the candidate pipeline.

IBM AutoAI has a clear and very extensive documentation, which describes the inner working of this tool in detail. Even citing original papers to algorithms and approaches used in their tool. A sense of transparency and trustworthiness is established by this practice. However, the full product is hidden behind a pay wall. Nevertheless, an interactive demo is available. This demo presents IBM AutoAI's very simple, interactive and intuitive UI, which could be easily used by non-techincal users, such as domain experts and business owners. Users can easily select files with their dataset, highlight a target feature and let AutoAI figure out the rest. Level 2 accesibility was assigned to IBM AutoAI.

The experience overall was very smooth. IBM AutoAI presents the user with a wide variety of visualisations for the final pipeline, which provides further insight into the chosen model and what its strong or weak points may be. Moreover, the pipeline can be deployed and served with a few clicks. However, it should be noted that this demo is scripted and the users are allowed to only perform pre-defined actions. Hence, it was not possible to upload our custom dataset and evaluate the tool.

3.2.7 AutoML-Zero

AutoML-Zero[11] is a promising open source tool for research. It can discover or improve existing algorithms via an evolutionary algorithm. These novel or existing algorithms found by AutoML-Zero are discovered symbolically as opposed to numerically, therefore it is easier to explain the solutions. Its search space is very general therefore the possibilities for the pipelines are practically endless. AutoML-Zero places emphasis on requiring very little manual work when designing the algorithms. The authors claim that it can automatically search for complete ML algorithms, while placing very little restriction on the form and using only simple mathematical operations as building blocks. Furthermore, they speculate that our bias towards human-designed algorithms can limit the innovation in the ML space. AutoML-Zero tries to return the focus on the search method and look for novel solutions in rather generic search space[15].

[9] https://mlbox.readthedocs.io/en/latest/

[10] https://www.ibm.com/docs/en/cloud-paks/cp-data/4.0?topic=models-autoai

[11] https://github.com/google-research/google-research/tree/master/automl_zero

It is important to note that a machine learning expert is still necessary to design an evaluation method, feature engineering, interpretation of the results, deploying of the models et cetera. In light of these requirements Level 0 accessibility was assigned to AutoML-Zero.

Lastly, when running AutoML-Zero the users are required to provide a configuration to define the task and search space parameters. A certain level of familiarity with the project and an understanding of its inner working seems to be required when crafting a reasonable configuration. Moreover, we were unable to produce a custom configuration which yields any meaningful solutions to our task.

3.2.8 AWS Machine Learning & Vertex AI

AWS Machine Learning[12] and Vertex AI[13] are cloud based application developed by Amazon and Google respectively. Both of these products are marketed as tools which enable its users to empower their products with machine learning without having to write code or have expertise in the machine learning field. Nevertheless, neither of these tools are very transparent about how they achieve this. Additionally, they both provide a free tier "demo", however, to sign up for it, the user has to provide their credit card information, the lack of transparency and trust deterred us from providing this information, hence, we were unable to get hands on experience with these tools.

3.2.9 Auto-WEKA

Auto-WEKA[14] is an open source package, built on top of WEKA. WEKA is a collection of ML algorithms used for tasks such as data mining. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization[15]. All of these tools are powerful on their own, however, they all have hyperparameters which need to be tuned. That is where Auto-WEKA comes into the picture. Auto-WEKA builds, fine-tunes and evaluates pipelines. Similarly to other tested tools, Auto-WEKA leverages innovations in Bayesian optimization to traverse the search space efficiently. Auto-WEKA has a friendly user interface, which can be used to load the data, select a target feature and start the searching process without having to write any code. After, Auto-WEKA reaches its time limit it stops the search and displays information about the run. This information includes total number of run experiments, estimated error rate, best classifier and its attributes. Furthermore, it provides JAVA code which could be copied into the user's application to run the classifier in the user's own program. However, that is not necessary since Auto-WEKA lets you save the model, which then can be loaded into classification section of WEKA and used to apply it on novel data. Due to the ease of use and no need to write any code, accessibility Level 0 was assigned to Auto-WEKA. Auto-WEKA's best model reached the accuracy of 0.986 on our withheld test dataset.

4 Discussion, Summary, Conclusion

We have evaluated the transparency and accessibility of a number of the most notable AutoML tools that exist within the field currently. Additionally, we have included a number of in-depth descriptions of each tool to illustrate their working beyond the criteria. The results show that users often still need a background in IT to be able to use AutoML tools. In addition to that, a number of prominent AutoML tools refrain from making their code open source. Having summarized the results briefly, we will now move on to critically consider a number of issues regarding our method.

In this work, only free to use AutoML tools were run and tested on the data set. Due to this, a number of accuracy values are missing in Table 1. In addition to that, the search space could not be discerned for two of the AutoML tools. Which tools were not free to use is also included in Table 1. The remaining information concerning the premium AutoML tools was gained by researching the tools, rather than testing them.

This leads us to the second point of concern, which is that the columns *Open Source* and *Free To Use* in Table 1 are identical, and *SaaS* is the complement of them both. This seems to be the result of the property that SaaS projects in general tend not to be open source or free. The main issue with this is that not much new information is gained with each column due to this correlation. However, we would argue that these criteria are still distinct enough to justify adding all three of them since there are SaaS projects that are free and open source, and this might hold for AutoML tools in the future as well.

Regarding time limits for training, some tools did not allow for the setting of a specific time limit. As noted before, in the case that it was possible to set a time limit, the time limit was set to 10 minutes. However, even with a set time limit, the run-time varied a lot, since in certain cases the time limit corresponded to the time limit for generating new experiments. The AutoML tool will then still continue to evaluate already generated experiments, and thus go beyond the time limit in terms of run time. The key problem with this is that the accuracy values are no longer completely comparable. This is because it is no longer a fair fight when different tools have different resources. Having discussed a number of concerns with the methods used in this work, let us now turn to our conclusion.

Regarding accessibility, It has been shown that there quite a lot of AutoML tools that are still not accessible to the average user. Researchers and developers within the field of AutoML must not lose sight of one of the initial goals of the field, which is making machine learning accessible to all.

When it comes to the general experiences with the AutoML tools in this work, there seems to be quite a lack of visualization tools for performance of model configurations. This is quite surprising since this is exactly what Drozdal et al. [6] refer to as a way to increase trust in AutoML. Clever visualization is also a simple way to increase the transparency and accessibility of a tool, thus it is surprising that this does not exist in most of the tools.

On the question of transparency, most of the state-of-the-art AutoML tools are transparent when it comes to their model search space and final model configuration. However, there are also a small number of tools which are not free to use and not open source. For the sake of the field it is to be hoped that the majority of AutoML tools stay open source, since this allows developers and researchers to build upon each other's work.

In terms of future work, it would be interesting to see the insights gained from the work of Drozdal et al. [6] incorporated in novel and existing AutoML tools. Additionally, research in making AutoML tools accessible to users outside of the field of machine learning could be fruitful, since research on this is lacking currently.

[12] https://aws.amazon.com/machine-learning/

[13] https://cloud.google.com/vertex-ai

[14] https://www.cs.ubc.ca/labs/algorithms/Projects/autoweka/

[15] https://www.cs.waikato.ac.nz/ml/weka/

References

[1] A. M. Alaa, T. Bolton, E. Di Angelantonio, J. H. F. Rudd, and M. van der Schaar. Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 uk biobank participants. *PLOS ONE*, 14(5):1–17, 05 2019.

[2] Axel Aronio De Romblay. Mlbox, 2022. https://mlbox.readthedocs.io/en/latest/index.html, Last accessed on 2022-03-21.

[3] T. Carneiro, R. V. Medeiros Da NóBrega, T. Nepomuceno, G. Bian, V. H. C. De Albuquerque, and P. P. R. Filho. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685, 2018.

[4] K. Chauhan, S. Jani, D. Thakkar, R. Dave, J. Bhatia, S. Tanwar, and M. S. Obaidat. Automated machine learning: The new wave of machine learning. In *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 205–212, 2020.

[5] V. D'Orazio and Y. Lin. Forecasting conflict in africa with automated machine learning systems. *INTERNATIONAL INTERACTIONS*.

[6] J. Drozdal, J. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. Muller, L. Ju, and H. Su. Trust in automl: Exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, IUI '20,

page 297–307, New York, NY, USA, 2020. Association for Computing Machinery.

[7] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[8] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[9] S.-J. Hong, H. Kim, D. Schrader, N. Bernasconi, B. C. Bernhardt, and A. Bernasconi. Automated detection of cortical dysplasia type ii in mri-negative epilepsy. *Neurology*, 83(1):48–55, 2014.

[10] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

[11] U. Khurana, H. Samulowitz, and D. Turaga. Feature engineering for predictive modeling using reinforcement learning, 2017.

[12] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307, 2016.

[13] T. T. Le, W. Fu, and J. H. Moore. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256, 2020.

[14] L. Parmentier, O. Nicol, L. Jourdan, and M.-E. Kessaci. Tpot-sh: A faster optimization algorithm to solve the automl problem on large datasets. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 471–478, 2019.

[15] E. Real, C. Liang, D. R. So, and Q. V. Le. Automl-zero: Evolving machine learning algorithms from scratch, 2020.

[16] D. Rohde, M. Drinkwater, M. Gallagher, T. Downs, and M. Doyle. Applying machine learning to catalogue matching in astrophysics. *MONTHLY NOTICES OF THE ROYAL ASTRONOMICAL SOCIETY*, 360(1):69–75, JUN 11 2005.

[17] A. Sabharwal, H. Samulowitz, and G. Tesauro. Selecting near-optimal learners via incremental data allocation, 2015.

[18] S. K. K. Santu, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni. Automl to date and beyond: Challenges and opportunities, 2021.

[19] R. E. Shawi, M. Maher, and S. Sakr. Automated machine learning: State-of-the-art and open challenges. *CoRR*, abs/1906.02287, 2019.

[20] F. Tarsitano, C. Bruderer, K. Schawinski, and W. G. Hartley. Image feature extraction and galaxy classification: a novel and efficient approach with automated machine learning. *Monthly Notices of the Royal Astronomical Society*, 511(3):3330–3338, 02 2022.

[21] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, and R. Farivar. Towards automated machine learning: Evaluation and comparison of automl approaches and tools. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1471–1479, 2019.

[22] Q. Wang, Y. Ming, Z. Jin, Q. Shen, D. Liu, M. J. Smith, K. Veeramachaneni, and H. Qu. Atmseer: Increasing transparency and controllability in automated machine learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[23] J. Waring, C. Lindvall, and R. Umeton. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, 104:101822, 2020.

[24] Q. Yao, M. Wang, H. J. Escalante, I. Guyon, Y. Hu, Y. Li, W. Tu, Q. Yang, and Y. Yu. Taking human out of learning applications: A survey on automated machine learning. *CoRR*, abs/1810.13306, 2018.

[25] M. Zöller and M. F. Huber. Survey on automated machine learning. *CoRR*, abs/1904.12054, 2019.

An overview of the Graph Neural Network pipeline

David Boerema and Auke Roorda

Abstract—The graph data structure is a very useful data structure for modeling many applications like chemistry, social media and recommender systems. With the increasing availability of large amounts of graph data for these applications, there is an increasing demand for the processing of this data. This gives rise to Graph Neural Networks; models that are able to learn features from graph data. Over the last decade, there have been many new and different models constructed for graph data, resulting in greater applicability and accessibility of the typical machine learning tasks on graph data. In this paper, we provide an overview of the IO-pipeline when working with graph neural networks, describing the different relevant aspects of the input graphs and how to accommodate for these variations, followed by the various layers of a graph neural network, and a concise list of possible tasks. Finally, a discussion of the strengths and weaknesses of the current methods is provided.

Index Terms—Graphs, Neural networks, pipeline.

1 Introduction

In computing science, discrete graph data structures are a valuable type of data representation that can be used to model many types of data. Graphs are very powerful, as they provide efficient solutions for many problems through for instance graph trees. As such, storing data in graphs can often provide many benefits through the representation alone.

For this purpose, graph neural networks (GNNs) have been designed to bridge the gap between traditional artificial neural networks and graphs. Artificial neural networks typically learn from vector data and are able to provide great generalisability for different kinds of data and tasks. GNNs are able to provide the same kind of generalisability but then for graph data, as the difference in representation leads to different tasks and data properties.

The data flow in a GNN follows a specific pipeline from input to output while learning. This pipeline consists of 3 core steps that any GNN should take:

1. Preprocess the graph as input of the GNN.
2. Learn graph features through GNN layers.
3. Apply learned features to a specific task.

This pipeline is very similar to the IO-pipeline of artificial neural networks and is used in this paper to introduce all the important concepts of GNNs. This paper aims to be an accessible, high-level overview of the IO-pipeline when working with graph neural networks. Multiple important concepts of GNNs will be discussed and compared to their counterparts in artificial neural networks. This paper aims to contribute an overview of GNNs through a comparison to artificial neural networks as well as a discussion of the current strengths and weaknesses of the state-of-the-art GNNs.

In section 2 an overview is given of the various types of graphs, how they can be preprocessed and how they can be transformed to be used as an input of a GNN. In section 3 the inner workings of GNNs are explained, explaining the roles of the different layers in a GNN. Then, in section 4, the various tasks for which GNNs can be used are addressed. In section 5 a small overview of similar works and some core GNN papers is given. Lastly, in section 6 a discussion about the current state-of-the-art is presented. Note that in this paper, the term ”Graph Neural Network” (GNN) is used to encompass all neural networks that work with graph data, and unlike [1], it is not used for the specific Graph Neural Network model as described in [16].

This paper serves as a literature review of the core concepts of the graph neural network research field. This literature review was conducted through an analysis of multiple similar literature reviews and a subset of their references. The literature selection favored earlier and more popular works as these can often be identified as the core results of the research field upon which many later works improve. An overview of some of the considered works is given in section 5.

2 Graph data as input

In general, the input of a neural network consists of multiple scalar values, such as a feature vector describing a particular example. Depending on the task at hand, the neural network tries to classify this example, or use this input for regression. However, not all data can straightforwardly be represented by feature vectors. Graphs in particular, do not conform to standard sizes or explicit, predefined structures that feature-vector oriented neural networks expect. Therefore, a so-called *transduction* is done, which converts the nodes of the graph into node embeddings (sometimes called node states, node representations) [1]. These are not just a function of the feature vector associated with the node, but also influenced by the vertices connected to the node, in an effort to not lose the connectivity information of the graph. After the transduction, this representation of the graph can be put into a neural network.

2.1 Graph types and representations

There are multiple variations on graphs. A standard graph is a data structure that can describe the relational structure between nodes, by means of connections between the nodes. Formally, a graph $G = (V_G, E_G)$ consists of at least a set of vertices V_G and a set of edges $E_G \subseteq V_G \times V_G$, consisting of pairs of vertices. The set of edges E_G is sometimes represented by an adjacency matrix A_G, a $|V| \times |V|$ matrix, with elements $a_{u,v} \in \{0,1\}$ denoting the presence of an edge from vertex v_u to v_v. A variation of standard graphs, hypergraphs, which have edges between more than two nodes, are not discussed in this paper.

In many practical applications of graphs, the nodes and edges of a graph are given properties to represent certain features. The properties added of course depend on the context in which the graph neural network is applied.

Some elements of the data are not encoded by the feature vectors on nodes or edges, but are part of the structure of the graph itself, such as the direction of edges in a directed graph. A second aspect is that of node and edge types; in homogeneous graphs, each node is of the same type as other nodes, and the same holds for edges. In heterogeneous graphs, the type of each node and edge can differ from others.

- *David Boerema is a master student at the University of Groningen, E-mail: d.h.boerema@student.rug.nl*
- *Auke Roorda is a master student at the University of Groningen, E-mail: a.c.roorda@student.rug.nl*

2.2 Graph structures

An important distinction that [19] makes is between structural (explicit) graphs, and unstructured (implicit) graphs. In structural graphs, the data is inherently structured, such as in social networks or knowledge graphs. But it is not always the case that the data being worked on has this explicit structure. With natural language processing, the sequence of words is usually transformed into a graph, in which the references between words are stored. In this context, the graph structure has to be deduced from the data first, before a graph becomes apparent. This is called an unstructured setting [19].

Finally, there is the distinction between static and dynamic graphs. Dynamic graphs are updated as time progresses. These updates themselves can be important sources of information [19].

2.3 Preprocessing graphs

Some of these structural variations of graphs can be consolidated, to reduce the amount of different graph types that have to be worked with later on.

A trivial transformation is that from an undirected graph to a directed graph; every edge $e_{u,v}$ in the undirected graph is replaced by a pair of symmetrical directed edges $(e_{u,v}, e_{v,u})$.

For heterogeneous graphs, an approach using so-called *meta-path-based methods* can be used to reduce a single heterogeneous graph into multiple homogeneous graphs [19]. A meta-path consists not of the vertices in the path, but the types of the vertices and the types of the edges that connect them. An example as given by [13] is on a graph of an academic network, with e.g. author (A), paper (P), conference (C) nodes. A meta-path (A)-(P)-(C)-(P)-(A) would then indicate that two different authors (A) with contributions to two different papers (P) have presented these at the same conference (C). This captures a somewhat weak relation between the two authors. A meta-path (A)-(P)-(A) on the other hand would indicate that two authors contributed to the same paper, which can be a sign of a stronger relation between two authors. Using this method, a new graph of just author nodes can be constructed, even though the author nodes are not directly connected in the original graph. This results in a homogeneous graph, where every node is of the same type.

2.4 Representation learning

As already seen in the case of meta-paths, the connections of a node are able to capture information that is not present on the node itself, but that can still be of relevance. In general, this kind of information is either aggregated using the direct neighbours, or by a random walk from the node. The former can be seen as the equivalent of applying a kernel on an image, where the neighbouring pixels are replaced by the nodes that reside in the neighbourhood of each node.

The neighbourhood of a vertex v consists of all vertices u that can be directly reached with an edge from v: $N_v = \{u \in V_G | e_{v,u} \in E_G\}$. Aggregating data using the neighbourhood can of course be recursively applied to aggregate data from further neighbours as well.

A second option is to initialize a so-called random walk from a node. This captures the topological features of the node by taking a sequence of steps to neighbouring nodes. One example of this is [7], whose Node2Vec framework is based on a parameterized version of random walks, in which a balance can be stricken between a depth-first- and a breadth-first-search approach. Collecting the relevant neighbouring information of a node, to represent it in a single feature vector, is the processing of representation learning, in which not just the features of the node itself, but also an encoding of its topological features is captured.

3 Learning from Graphs

In this section, the concept of ‘learning’ from graph data along with the possible learning tasks will be introduced. Zhou *et al.* go into great detail about the learning methods [19], but the goal of this paper is to provide an easy to understand overview of the inner workings of GNNs through a comparison to the traditional neural network framework.

Traditional neural networks for continuous data typically consist of a couple of core components: activation functions, loss functions and the computational neuron layers. With a change of input and learning task also comes a change of these components, as the learning happens through these components. Graph neural networks need new types of computational neuron layers as well as loss functions that suit the learning objective.

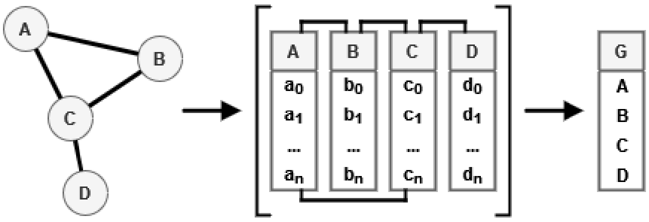


Fig. 1. An example of how a simple undirected graph **(left)** is represented as a hidden state **(middle)** as well as the hidden aggregated graph state **(right)**.

3.1 Computational layer types

The layers of an artificial neural network come in multiple types, all having different functions in the network as a whole. In image classification, convolutional layers are commonly used to select specific image features while pooling layers can be used to up- or downsample the data. For both of these, GNN variants exist which specifically act upon graph properties. Additionally, GNNs define new types of layers specific to graphs, as for instance the graph topology and neighborhoods of vertices matter. Zhou *et al.* divided the different layer types for GNNs into 3 groups [19]:

- **Propagation layers**, which diffuse vertex neighborhood information as well as capture feature and topological information of the graphs.
- **Sampling layers**, which sample specific parts of the graph in an effort to reduce the complexity of neighborhood aggregation.
- **Pooling layers**, which provide lower level representations of parts of a graph.

Through a combination of these layers and other neural network components, a GNN can learn graph features that can be used for different kinds of tasks. Graph features can consist of node, edge and whole graph-level features and the type of features learned is determined by the task. More detail about the possible tasks of GNNs is given in section 4.

The layers of a GNN work on the internal representation of the graph. An example of the conversion from graph to internal representation can be seen in figure 1. This representation can be seen as a hidden state represented by a feature vector per vertex, which can be operated on and aggregated to receive the internal graph representation. This type of representation is consistent across many GNN architectures [1].

3.1.1 Propagation layers

Propagation in graph neural networks serves the purpose of performing the actual learning in the network, as these layers contain trainable parameters. In graph neural networks, these layers also have the job of *context diffusion*, where the goal is to spread the information from one vertex to the other. Propagation is achieved using 2 types of operators: convolutional and recurrent operators.

Convolution

One of the problems one might encounter when processing graphs is the variable size and shape, which can form problems when applying any input sensitive method. This problem has been solved through processing graphs at the vertex level rather than graph level, which

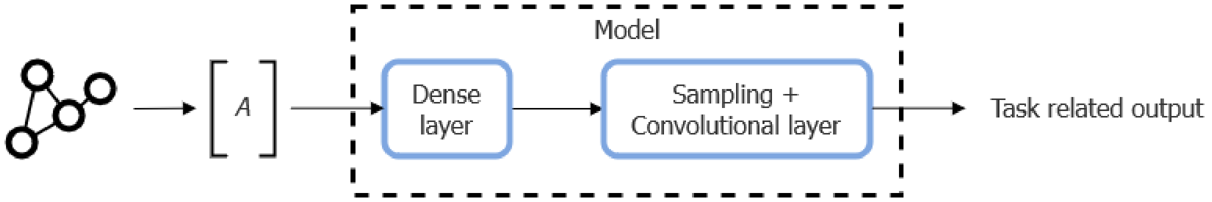


Fig. 2. An overview of the FastGCN [3] pipeline. *A* denotes the preprocessed graph data vector, which is used as the input of the GNN model.

is one of the most important building blocks in graph neural network models [1]. For this to work, the data of neighboring vertices needs to be propagated across all vertices of the graph. One of the most popular propagation layers to use in a standard artificial neural network is a convolutional layer. This is a trainable layer that can be used for aggregating neighboring data, and as such is also very practical for context diffusion in GNNs.

For graph based convolution, there are 2 types of approaches: spatial and spectral [19]. Spatial approaches operate on the graph topology directly and can make use of attention mechanisms [18]. Multiple spatial methods can also be combined into a framework like MoNet [14]. Spectral approaches use the graph Fourier transform to apply convolution using the spectral representation of the graph [19][11][17].

Recurrent

Recurrent operators are similar to convolutional ones, with the major difference being that recurrent operators share their weights within a layer [19]. These methods originally required the hidden states to converge, but methods using the gate mechanism (like LSTM [9]) have been developed to overcome some of the shortcomings of these operators and only run a certain amount of steps.

3.1.2 Sampling layers

One of the main issues with large and dense graphs is the number of neighbors for an arbitrary node in the graph. When neighborhood aggregation is performed on these nodes, the computational load scales in the number of neighbors. For this reason, sampling layers sample the neighborhood of a node to reduce the computational load. An example of this is the GraphSAGE network [8], which samples a fixed number of neighbors within a subset of nodes of the graph. Other sampling methods that sample at layer or subgraph level also exist [19].

3.1.3 Pooling layers

In feed forward neural networks, it is common to follow a convolutional layer by a pooling layer to reduce the data dimensions and process the convoluted data. This is also the case for pooling layers for GNNs, which have the aim to reduce the data and discover important communities within the graph [1]. These layers range from performing simple operations (sum, min, max) to trainable operations that perform clustering to apply topological pooling.

3.2 Example model

To exemplify how some of these layers can work in conjunction, an example model using the FastGCN network [3] is shown. This network had 3 tasks on 3 data sets:

1. Classify research topics on the Cora citation data set.
2. Categorise academic papers from the Pubmed database.
3. Predict the community structure of a social network using Reddit posts.

For these tasks, a model using 2 layers was used. An overview of the model and pipeline of the FastGCN model can be seen in figure 2. A similar model was also used in [3] to model GCN [11] and GraphSAGE [8], with the major difference being the inner workings of the second propagation layer. Important to note is that much like GraphSAGE[8], the sampling and convolutional layers are combined into one layer for the FastGCN. This means that a node's neighborhood is sampled first and aggregated afterwards. Before the data is used in the model, it is preprocessed to generalise the input and optimize the training process. One example of preprocessing is the downsampling of the edges of the graph in the Reddit data set [8]. Across all tasks, the models were able to reach a minimum F1-score of `0.82` with GraphSAGE and FastGCN providing substantial improvements in training time over the classic GCN [3].

4 Tasks

Due to the ubiquitous nature of graphs, the amount of applications of GNNs is vast. The tasks in these applications can be categorized by the components of the graph they focus on: the nodes, the edges or the topology of the graph. Within each of these categories there are different possibilities, depending on the goal of the task and the availability of labels.

4.1 Tasks on nodes

Node classification aims to provide the most likely label for a given node in a network. In the graph of e.g. an academic network, nodes would be classified as being a paper, an author or a conference. For this to be possible, at least a segment of the nodes in the network should have their labels available, to facilitate training. On the other hand there is node regression, concerned with prediction of continuous features of a node. These could be objective measures, such as the word count of a paper, as well as more subjective features, e.g. a scale indicating "how related is this paper to the field of neural networks?". One of the frequently occurring goals of unsupervised learning is clustering. The aim is then to group similar nodes together [19].

4.2 Tasks on edges

Link prediction has the aim of deciding whether there should be a link between two existing nodes. Approaches for this are based on the principle of *homophily*, which states that nodes that exhibit more similar features are more likely to be connected to each other [12]. An exemplary case for link prediction is in the context of social networks. Typical social network graphs consist of nodes representing people, and edges to represent their relationship, e.g. whether they are friends or not. In an effort to connect users with each other, the social network wants to be able to provide suggestions to users to connect with other users they might already know. These suggestions are the result of link prediction on the social network graph. When edges are given properties themselves, link regression is possible as well.

4.3 Topological tasks

The Node2Vec framework by [7] is able to classify nodes according to their structural equivalence; nodes that are fully connected within their cluster are distinguished from more 'hub'-like nodes. These can be indicative of the role these nodes play in the network [7].

A different example is to predict a property of the complete graph, a task that finds its purpose especially in chemistry. The structure of molecules can be represented exceptionally well, with nodes representing atoms and different edges to match the different bonds between atoms. These two factors determine the properties of the molecule as a whole, which can be thought of as the properties of the graph. As discussed in [4], this can be used to predict different chemical properties of molecules. The properties can then be used for computer-aided drug design [19].

5 Related work

One of the initial problems when dealing with graphs was the presence of cycles. The first works in the area of neural networks for graphs, the Recursive Neural Networks [2], could not easily be extended to deal with this, and were therefore limited to processing acyclic graphs. One of the earliest models to be able to deal with cycles is the Graph Neural Network [16], a recurrent model using a state transition system, which implements a function that maps the nodes of a graph to a feature vector in a new euclidean space. It is based on information diffusion techniques, as well as approaches using random walks.

In 2014, a novel approach for random walks was presented by [15], named *DeepWalk*. It builds on the at-the-time recent progress in the field of language modelling, and employs a learning method that treats the random node walks as sentences. This improves the resulting embeddings significantly, while also being applicable on graphs that are too large to run spectral methods on [15].

However, the work of [7] notes that these feature learning algorithms are not able to capture certain topological aspects of the graph, due to their limited expressiveness. They introduce *node2vec*, a generalization of the *DeepWalk* approach, i.e. the *DeepWalk* approach can be simulated by *node2vec* by setting parameters accordingly. It introduces a tunable random walk, which can strike a balance between depth-first and breadth-first search.

In recent years, many publications have been made in the field of neural networks for graphs, albeit with little systematization, resulting in the need for a more solid framework. The work of [1] provides a new tutorial-like overview of the field of GNNs, with clear definitions of the concepts used as well as a set of building blocks for the construction of neural networks for graphs. It constructs this overview by summarizing and consolidating the methods of previous impactful works, without focusing on the most recently published articles in the field.

For an explicit overview of the design pipeline for GNNs, we refer the reader to [19], in which a four-step approach to the model design is presented, as well as a wide variety of frameworks and methods, with their use cases.

6 Discussion and Future work

The field of research of graph neural networks has come quite far since its inception. Many different methods with different purposes have been proposed as shown in [19], constantly trying to improve over previous work. Consequently, the current field of research has some strengths and weaknesses that will be highlighted in this section along with recommendations for future work.

6.1 Strengths and weaknesses

One of the benefits of the maturing of the field of GNNs is that GNNs are now more accessible than ever. With libraries like PyTorch Geometric [6], which provides PyTorch implementations of many important works, it is easier to get up and running with different GNN architectures. Surveys like [1] and [19] also contribute to this by providing overviews of general concepts and methods respectively.
However, the amount of research in this field also introduces the problem of over-saturation. Models like the GCN [11] and GraphSAGE [8] are well recognised and will be used when comparing the performance of a new model, even though superior models may have already been proposed but have gone under the radar. This pitfall is not specific to the research field of GNNs, but reduces the speed at which progress is made even though more studies are being performed on the subject. To solve this, an assessment should be made of what is state-of-the-art and what is not to set a new baseline to compare to.

Another strength of GNNs is the performance on tasks. As has been shown by Dwivedi *et al.* [5], GNNs can have pretty good performance on medium-size data sets for different kinds of tasks like node classification and link prediction. The caveat however is that the input of GNNs is not always generalizable to multiple types of graphs. Architectures like the GCN [11] do not directly account for edge directionality in graphs as well as heterogeneous graphs. While these deficiencies can be accounted for through preprocessing most of the time, this can have an impact on model efficiency by multiplying the dimensions of the input data by a factor, for instance by turning a single heterogeneous into multiple homogeneous graphs.

Finally, GNNs have also been shown to be vulnerable to multiple types of adversarial attacks just like other deep neural networks [10]. An adversarial attack tries to fool a GNN by modifying the model input with the goal of getting a wrong prediction. This is a big security concern, as the opaqueness of the operation of neural networks in general does not allow for detecting these vulnerabilities up front. This prevents the use of GNNs in security sensitive applications unless a model robust to adversarial attacks is used, and as such forms a weakness for most standard GNN architectures.

7 Conclusion

Graph neural networks have become a viable solution for performing machine learning tasks on big graph structured data sets. Various preprocessing and learning methods have been developed to solve numerous different machine learning tasks, and the field is ever expanding in search of more improvements. In this paper, the core concepts of GNNs have been discussed through the use of the IO-pipeline of artificial neural networks. First, the graph data structure along with preprocessing steps have been discussed to detail the input of GNNs. Secondly, the inner workings of GNN models have been explained through a comparison to artificial neural networks. Finally, the applications and strengths and weaknesses of the current GNN research field have been reviewed to form some recommendations for the future direction of the GNN research field.

Acknowledgements

The authors wish to thank the organizers of the 2022 Student Colloquium, as well as our student reviewers and expert reviewer Andrés Tello.

References

[1] D. Bacciu, F. Errica, A. Micheli, and M. Podda. A gentle introduction to deep learning for graphs. *Neural networks : the official journal of the International Neural Network Society*, 129:203–221, 2020.

[2] A. M. Bianucci, A. Micheli, A. Sperduti, and A. Starita. Application of cascade correlation networks for structures to chemistry. *Applied Intelligence*, 12(1):117–147, 2000.

[3] J. Chen, T. Ma, and C. Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[4] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[5] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020.

[6] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[7] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[8] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[10] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang. Adversarial attacks and defenses on graphs. *SIGKDD Explor. Newsl.*, 22(2):19–34, jan 2021.

[11] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2017.

[12] S. A. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8(34):935–983, 2007.

[13] G. Mei, L. Pan, and S. Liu. Heterogeneous graph embedding by aggregating meta-path and meta-structure through attention mechanism. *Neurocomputing*, 468:276–285, 2022.

[14] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2017.

[15] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: online learning of social representations. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

[16] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[17] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[19] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *ArXiv*, abs/1812.08434, 2020.

A comprehensive guideline for Human Activity Recognition with Deep Learning

Philip Andreadis and Mariya Shumska

Abstract—Human Activity Recognition (HAR) is the challenging problem of automatically identifying actions performed by individuals during their day-to-day routine, formulated as time-series classification. HAR has recently become a prominent area of research, as it can serve as a basis for many innovative applications in a variety of different fields like healthcare, smart homes, wellness, and entertainment. Moreover, the advances in activity-capturing techniques and the rapid growth of the Internet of Things (IoT) have led to the availability of a large amount of sensory data, triggering the application of state-of-the-art deep learning methods. In our study, we make a comprehensive review of the most promising deep learning techniques proposed by researchers, with respect to the various challenges and end goals of HAR. The plethora of deep learning morphologies and the vast space of hyperparameters make the search for the optimal solution a by far non-trivial task. Therefore, we will analyze different methods, highlight their strong and weak points and infer a set of guidelines for choosing the most appropriate deep learning approach for the HAR problem at hand.

We define five fundamental aspects that influence the choice of the DL methodology for HAR, namely the target users, type of sensor modality, characteristics of activities, training data peculiarities, and the need for real-time recognition. As the field is just emerging, it is not trivial to find a universal framework that tackles the challenges of all the above-mentioned criteria. Therefore, using them as the basis, we suggest a set of guidelines for building case-specific DL pipelines for different HAR problems along with the corresponding visual representation for easier navigation.

Index Terms—Human activity recognition, HAR, sensors, deep learning, neural networks, CNN, RNN, LSTM.

1 Introduction

Human Activity Recognition (HAR) refers to the task of classifying physical activities through the use of various sensor modalities. HAR systems have many areas of application, including health monitoring, elderly care, intelligent surveillance, smart homes, virtual reality gaming, etc. [33]. Recent advances in IoT and hardware, cheapening of the sensors and their presence in ubiquitous devices like smartphones and smartwatches attract the attention of researchers and the general public. Moreover, as the data collection becomes easier, deep learning (DL) for HAR has arisen and proved to be effective. However, due to the plethora of DL approaches and wide range of HAR applications, building an appropriate pipeline is not a trivial task.

In our work, we introduce a taxonomy consisting of five major criteria that influence the choice of the methodology, and suggest relevant deep learning techniques that may help to achieve the end-goal of a HAR task. One major factor that motivates this categorization is the distribution discrepancy between training and test data. The discrepancy can be accredited to three reasons, namely the monitored users, time, and sensors. As for the users, each of them has a unique motion pattern for the same activity, moreover, the data distribution even within one user can vary over time. Regarding the sensors, their types or even slight changes in device positioning would produce different data patterns. Thus, a model fitted for one user at a particular point in time will not necessarily perform well for other users or the same user in the future.

The possible HAR use cases can vary a lot depending on the aforementioned discrepancy sources, as well as on the end-goal of each scenario. Specifically, we highlight the problem of real-time HAR systems, as the goals of many HAR applications can be achieved only with online classification. Therefore, we try to summarize the applied methodologies and provide indicative state-of-the-art implementations with respect to different sensor and activity types, user-related discrepancy, data imperfections, and the ultimate objectives of each application. Moreover, we create visual guidelines that will facilitate the navigation in the choice of the most appropriate DL techniques for HAR.

In the following Section 2 we give an overview of related work that we used as a basis and mention major differences and contributions of our work. Section 3 provides relevant background information regarding the different ways to attain human activity data and introduces the DL methods mentioned in this paper. In Section 4, we discuss the five aforementioned aspects of HAR, along with our proposed visual guidelines, and report corresponding state-of-the-art solutions for each one of them. Finally, in Section 5, we provide an overview of the current state of the HAR field and the shortcomings of the ongoing research.

2 Related work

We base our work on the paper by Chen et *al.* [8] where an extensive taxonomy of HAR-related challenges and appropriate deep learning techniques have been proposed. However, it concerns the issues and solutions inherited from both DL and HAR, which might be harder to navigate having specific requirements and peculiarities for the activity recognition task. We do, however, extract the main challenges of deep learning recently outlined by its “founding fathers” in [3] and put them in the context of HAR. We consider other works such as [17, 4] that compared the performance of most common models like deep feed-forward, convolutional, and recurrent neural networks. Moreover, we also present and give references for specific state-of-the-art frameworks that were designed to fulfill particular HAR tasks in terms of e.g. synthetic data generation [13] or performance on mobile devices [10]. In general, our categorization is more concise than the one proposed in [8]. It can also be seen as more use case-oriented and practical since the reader can focus on specific criteria that are applicable for their HAR task or data at hand. Additionally, we put emphasis and elaborate more on real-time activity recognition and computationally cheap techniques as many recent use cases, especially in the entertainment or wellness realm, need almost instant classification on personal gadgets. Finally, one of the contributions of our work is visual aids in a form of flowcharts/decision trees which we could not find in other overview or survey papers on HAR.

- *Philip Andreadis is a master’s student of Computing Science at the University of Groningen, E-mail: f.andreadis@student.rug.nl.*
- *Mariya Shumska is a master’s student of Computing Science at the University of Groningen, E-mail: m.shumska@student.rug.nl.*

3 Background information

In general, activity recognition can be realized with video-based or sensor-based systems, however, the latter is a dominant approach due to fewer privacy concerns [8]. Thus, we focus on sensor tracking modalities in our paper. In turn, they can be divided into two main categories, namely ambient and wearable sensors.

Ambient. Ambient sensors are usually placed around the environment of target subjects. One obvious advantage of this type of sensors against wearables is that they can be used to simultaneously capture the behavior and interactions of multiple individuals, in multi-occupant scenarios. The most common setting where ambient sensors are used is that of the so-called smart-home, in which multiple sensor units are installed in a residence in order to capture the everyday activity of the dweller(s), enabling the facilitation of a variety of smart services. Examples of such sensors could be WiFi, radar, or radio-frequency identification (RFID) devices.

Wearable. Wearables can be attached directly to body parts and gather physiological and movement information about the individual wearing them. The advances in mobile technology have allowed for the embedding of these units (e.g. accelerometer, gyroscope) into mobile devices, such as smartphones, smartwatches, and wristbands making the process of data collection considerably seamless. In medical settings, the examples of wearable sensors are EMG and EEG modalities that measure the electrical activity of muscles and the brain, respectively.

The recorded sensory data is a time series, which, especially in the case of medical settings may be high dimensional. Time series classification is a challenging task for classical Machine Learning since domain knowledge might be required for suitable pre-processing methods and manual feature engineering. Therefore, as DL methods emerge, their potential has been actively discovered in the time series classification realm. In particular, Artificial Neural Networks (ANNs) are considered, since they might operate on raw data and are able to automatically extract features. Moreover, they can be used not only as classifiers but as data augmentation tools. ANNs try to mimic the human brain and thus consist of many connected neurons. Typically, they consist of many layers of neurons as it allows high-level feature extraction from the data and therefore improves the performance of the model. There exist different classes of neural networks to address particular problems or types of data. Below we will briefly describe the main ideas behind the most used networks in the context of HAR.

Deep Neural Network (DNN). A common practice in most of the Artificial Intelligence applications [6], DNN comprises a multi-layer feed-forward architecture, which consists of hidden units, synapses, weights, biases, and activation functions. DNN is a supervised machine learning algorithm, that can be used for both regression and classification tasks.

Fully Connected Network (FCN). FCN is essentially a DNN, where each neuron of a layer is connected to every neuron of the next layer. Although they yield lower performance than specific-purpose networks, they have a wide variety of applicability. For instance, FCN is frequently used as the last segment of a Convolutional networks architecture, purposed for image classification.

Convolutional Neural Networks (CNN). A most commonly used Deep Learning architecture to analyze images (e.g. extract image features, image segmentation, etc.). Inspired by the biological virtual cortex, convolution kernels (or filters) with learnable weights slide along the input image and provide a feature mapping of lower dimensionality.

Recurrent Neural Network (RNN). A network adapted for working with sequential or time series data since it has an 'internal memory' which helps them store the information about previous inputs to generate the following output of the sequence. This is realized through feedback loops which are absent in feed-forward architectures.

Long Short Term Memory (LSTM). An extension of RNN that allows remembering inputs over longer periods of time. This is achieved with three different gates that decide on accepting the new input (input gate), removing the information (forget gate), or allowing it to influence the output (output gate). LSTM is very suitable for processing time series where the lengths of the time lags are not known.

Autoencoder. A special type of feed-forward network that is trained to reconstruct its input to its output. The goal is to achieve a lower-dimensional representation, which is also referred to as latent space, of the input data.

Variational Autoencoder (VAE). A modification of autoencoder with regularized latent space. It assumes that the data has some underlying probability distribution and aims to find the parameters of that distribution. VAE is a generative model, i.e. is able to produce new content.

Generative Adversarial Network (GAN). Belongs to generative models. The training process is realized with two competing networks: generator for producing new samples and discriminator for distinguishing true data from the output of the generator.

Deep Belief Network (DBN). A generative graphical model, consisting of multiple layers of hidden units. Connections between layers exist, however, units within each layer are not connected with each other. This type of network, when trained without supervision, can learn to probabilistically reconstruct its input. DBNs can be trained further with supervision in order to perform classification.

4 Deep Learning approaches for Human Activity Recognition

In the following subsections, we will provide a summary of the challenges posed by five crucial aspects and the corresponding state-of-the-art solutions.

4.1 User-related discrepancy

Users are of course the integral component of the HAR problem. Each person has unique activity patterns and behavior. Therefore, the assumption that the model trained on one person's data would perform equally well for another or even the same individual after some time period will not hold.

Clearly, personal biological characteristics and different environments affect the way the task is performed. For example, people of different ages will have a diverse walking or running pace. Since HAR models may be intended to be applied to previously not seen subjects during the working phase, the aforementioned factors account for user-related distribution discrepancy between training and test data. Obviously, using both training and test data collected from the same user is not a viable tactic, because while it may showcase high performance during training it will not yield good generalization when deployed against new targets. Thus, researchers have proposed methods to address this form of distribution divergence in an effort to mitigate the loss of performance when models are tested across individuals. [36] propose building personalized RNN models for each specific individual, while [21] uses Learning Hidden Unit Contributions to achieve user adaptation of CNN models and [27] employs transfer learning for the same purpose. On a different direction, [29] have used GANs to generate target domain data directly from the source domain in order to introduce the needed discrepancy.

Another important characteristic is inherited from the fact that activity data is temporal. Especially, if the HAR system is designed to be long-term, the performance of the model trained on the initially collected data may decrease due to the natural changes in human activities. Such changes can occur within the same class of activity, like gait changes, for example. To adjust to the new user's characteristics, active learning can be incorporated into DL models. The idea is to continuously update the system using upcoming data and requesting a ground truth label. For instance, Gudur et. *al* [16] proposed a Bayesian Neural Network with dropout for uncertainties approximations. Another long-term-related issue is that new types of activities may occur over time, that were not included in a training set. One approach to deal with it would be decomposing the new activity into mid-level features like leg (arm) down (up), however, it requires experts to define these features [8]. Another way of dealing with new classes is treating HAR as an open-set problem, meaning that the activity is not exactly classified, but rather labeled as part of the target activity or not. In order to obtain a so-called negative set consisting of all non-target activities, GANs can be used [38].

We can see that user-related discrepancy is inevitable in the context of HAR and need urgent addressing if the HAR system is not intended to be personalized or has to work in a long-term perspective. The flowchart that suggests solutions for different types of discrepancies can be found in Figure 1.

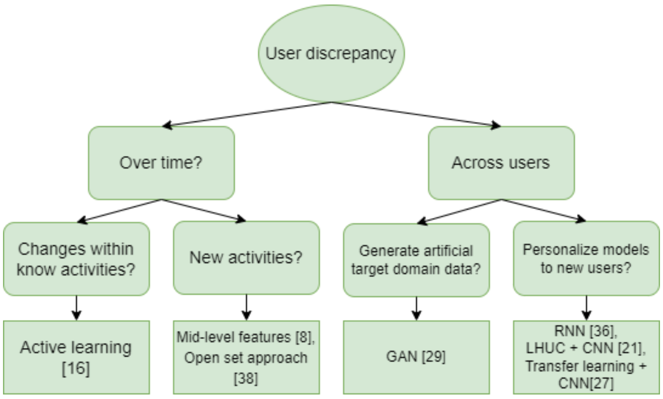

User discrepancy
Over time?
Across users
Changes within know activities?
New activities?
Generate artificial target domain data?
Personalize models to new users?
Active learning [16]
Mid-level features [8], Open set approach [38]
GAN [29]
RNN [36], LHUC + CNN [21], Transfer learning + CNN[27]

Fig. 1. Visualized guidelines for choosing the appropriate DL approach in terms of user-related distribution discrepancy along with references to relevant papers.

4.2 Training data peculiarities

Supervised learning of neural networks needs a large amount of labelled training data in order to be successful. On the one hand, the ubiquity of various sensor devices can provide a lot of data, however, the labelling process might be quite expensive, time-consuming and error-prone. On another hand, some activities or events are rather rare and hard or unethical to obtain, such as falls of elderly people or disease-related motor complications. Therefore, one should consider the peculiarities of the available training set in both cases of the ”excess” of (unlabelled) data or lack of samples.

Semi-supervised techniques may alleviate the tedious annotating task [8]. The example of such technique is pre-training of a network on unlabelled data to extract features which then are used during a supervised training with a limited supply of labels. At this stage, the weights in the feature extractor can be fine-tuned with backpropagation [3]. Deep generative models like DBNs [2] and stacked autoencoders [11, 15] are frequently used as feature extraction in a pre-training phase for HAR. Another semi-supervised approach is called co-training which trains two classifiers on different views of data [19]. This is applicable for HAR tasks only if multiple sensing modalities were used [7], since then different data views are available.

Synthesizing human activity data may solve the issue of lack of training samples. Artificial data can be produced by generative models such as VAE and GANs [14]. Different frameworks were developed and tested for human activity generation, such as Sensory-GANs [34], Adversarial Variational Embedding [40], Control-HAR-GAN [13]. These approaches, however, are not unsupervised, therefore, labelled data is required which brings us back to the annotation issues mentioned at the beginning of this subsection. Visual guideline in a form of a decision tree in Figure 2 summarizes main approaches to tackling training data problems.

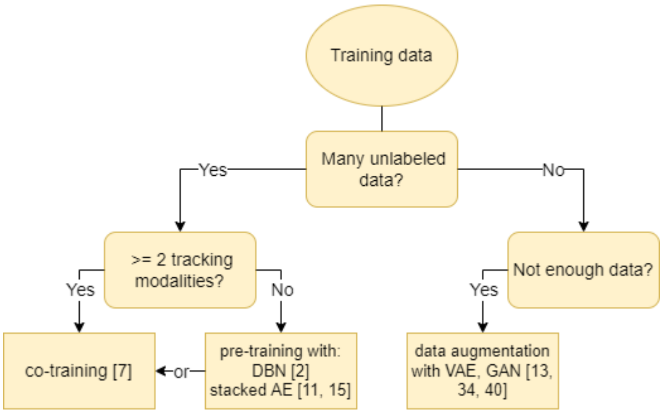

Training data
Many unlabeled data?
Yes
No
>= 2 tracking modalities?
Yes
No
Not enough data?
Yes
co-training [7]
or
pre-training with: DBN [2] stacked AE [11, 15]
data augmentation with VAE, GAN [13, 34, 40]

Fig. 2. Visualized guidelines for choosing the DL approach for tackling potential training data challenges along with references to relevant papers.

4.3 Activity tracking modalities

Due to their sensitivity, a small variation in the sensors could translate to significant changes in the collected data. Additionally, the plethora of available modalities accounts for data with totally different shapes, sampling frequencies, and scales. To name a few of these factors, the sensor instances influence parameters such as the sampling rate, while wearable sensors output data according to the body part they are attached to. Lastly, in the case of ambient sensors, the surrounding layout greatly affects the generated signal.

In order to address the heterogeneity of sensor instances, researchers have used GANs for data augmentation [20] to synthesize data from different sensors with sufficient discrepancy. Moreover, in an effort to tackle the issues deriving from the different types and positioning of wearables, [1] have created a domain adaptation method to expand training algorithms from known wearable sensors to new sensors, based on generative autoencoders. Since smart devices can be placed on multiple positions, many source domains (i.e. preexisting sensors with labeled data) can emerge, which are needed for annotating the target domain (i.e. sensor placed on new position) in the context of transfer learning. [35, 9] propose methods of similarity distances between the target and source domains to facilitate the accurate transfer of knowledge between them and recognize activities on the former. Lastly, regarding ambient sensors, which are considerably influenced by the surrounding layout in both domestic and wild environments, the need to mitigate the discrepancy coming from different domains has directed the research towards one-fits-all models that are able to capture domain-independent features and fit to varying HAR scenarios [42].

Asides from the distribution discrepancy sources, the most obvious division of sensor types is that of wearables against device-free and ambient sensors, placed at a fixed point in space. [17] provides an extensive comparison among typical DNNs, CNNs, and RNNs. Their results show that regular DNNs require more investment in parameter exploration and show significantly higher spread in performance compared to the most sophisticated alternatives of CNN and RNN. On the other hand, [5] propose a novel method that uses fully convolutional networks combined with frequency-based encoding with word embedding from Natural Language Processing for automatic feature extraction, applied to HAR in smart-homes. A diagram showing the most suitable techniques w.r.t. sensor types can be found in Figure 3.

4.4 Target activity types

Even though existing methods yield promising performance against simple and repetitive actions, e.g., jogging or walking and stationary activities such as standing, little progress has been made towards recognizing more complex scenarios. The so-called composite activities consist of sequences of simpler actions, (e.g. preparing dinner) and

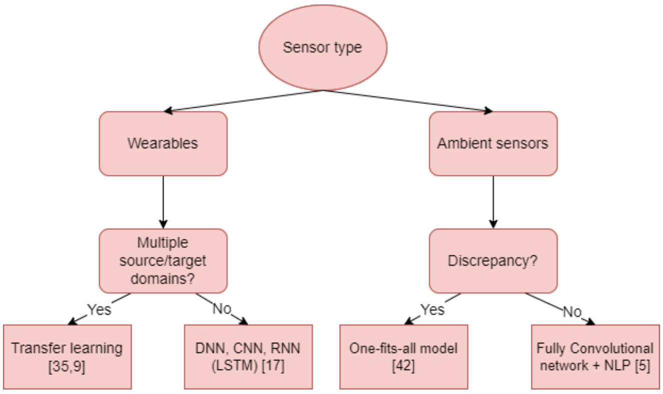


Fig. 3. Visualized guidelines for choosing the DL approach for the corresponding sensor types along with references to relevant papers.

their recognition is greatly desired as they tend to be a more complete reflection of people's daily life. [31] mixes complex and simple activities and uses a DNN to categorize both in a unified way. The model shows a high accuracy of 90%, however, its training is limited to samples from a single subject. In contrast, [23] treats composite and simple activities separately. Specifically, a CNN is used to extract features from simple activities, whereas an LSTM network is applied to complex activities to learn the temporal dependencies. Joint learning of the two networks is then realized by a shared structure between them.

A single person can perform multiple actions at a given period of time, e.g., eating while watching television, hence the recognition of concurrent activities has become another point of interest for researchers. The first approach to this problem is to consider each concurrent activity separately and classify them individually. For instance, the authors in [41] designed a single fully-connected network for each candidate activity on top of a shared multi-modal feature space, where independent softmax layers classify each one of them. A major disadvantage of this architecture is the fact that the computational cost will significantly increase as more activities are added. In an opposite way, [22] realized a multi-layer combined CNN and LSTM framework, which targets overlapping activities concurrently, considering every possible activity combination. It is important to mention that this specific HAR task has not yet been extensively explored and the approaches discussed above are only early steps.

Finally, an even more challenging aspect of activity recognition, with significant practical value, is multi-occupant activities. Most state-of-the-art studies provide solutions only for scenarios where the generated activity signals come from a single person. However, in most residential and working environments the existence of multiple individuals is very likely. Two distinctive cases are observed here, namely, the instance where two or more persons carry out independent activities (e.g. one is speaking on the phone while the other is eating), termed parallel activities, and the instance of collaborative activities where multiple individuals perform the same task in a synergetic manner (e.g. playing pool). In [30] a proposed multi-label RNN, with each RNN cell corresponding to the activity of a single occupant, is trained on ambient sensor data and used for daily parallel activity recognition in the context of smart home. An important remark is that parallel activity recognition from wearables can be treated as multiple single-person activity recognition tasks and tackled with one of the methods discussed formerly. On the contrary, authors in [28] leveraged data from both ambient and body-worn sensors as inputs into a multimodal sequential network composed of a RBM layer, a DBN with multiple RBM layers and a final Multi-layer Perceptron (MLP), for the inference of group activities performed by pairs of occupants. Figure 4 summarizes the main types of activities and appropriate DL methods according to our findings.

4.5 Real-time feasibility

For some applications, a real-time classification might be favorable. It might be especially important for medical scenarios where a dangerous event such as falling must be recognized instantly. In general, we observe two dominant architectures for real-time HAR, which are CNNs due to relatively low number of connections and high parallelism [18] and recurrent networks like LSTM which model movement at the sample level. LSTM-based approaches usually perform better since they make use of long-term dependencies, but CNNs are faster to train [17]. Unlike recurrent models though, they require fixed input size, meaning that the data needs to be segmented. Although choosing a suitable segment size that results in fast and accurate classification might be challenging, using a sliding window allows for real-time activity recognition.

One of the biggest issues that hinder real-time HAR is that the most accurate DL models are resource-hungry and therefore, are not suitable for portable devices. In order to solve the problem of high computational costs, researchers proposed different modifications for the neural networks' architecture. Although deep models are superior for feature extraction, shallower models that have lower resource requirements can still perform well if combined with manually engineered features as demonstrated in [25, 26, 24]. Another strategy would be to optimize the network by, for instance, filter size reduction for improved memory consumption in the case of CNN [26] or network quantization [12, 39] which is efficient due to bitwise operations. Some specific architectures were proposed, such as self-gated RNN, where the complexity is lower in comparison to a standard LSTM model [32], CNN with CondConv [37] layers instead of conventional convolutions [10]. Therefore, we cannot conclude that either CNN- or RNN-based model is the most feasible for accurate real-time activity recognition, both can perform comparatively well if optimized. However, as noted in [17], CNNs might be more suitable for prolonged and repetitive activities, while RNNs outperform them significantly in the case of short actions. For a visual overview of available approaches for real-time systems refer to Figure 5.

5 Discussion and Conclusion

As we can see, various DL techniques can be applied in the context of HAR problems to solve sensor- or training data-related issues. Moreover, the end goals of the application with respect to the target activities and need for real-time recognition, non-personalized or long-term system may influence the architecture of the classifier network significantly. Finding the ultimate DL model for HAR seems not to be possible due to unavoidable trade-offs between the desired requirements, such as speed and accuracy. However, ensuring sensible prioritization and timely detected data peculiarities will lead to the optimal pipeline. To make this process easier, we summarized existing works on challenges and DL techniques for HAR and derived five crucial criteria to look into. To facilitate it even further, we generate the visual support of our guidelines.

Although many of the works discussed in our review show promising results and already comprise deployable solutions, there is a number of challenges that have not been addressed yet. This is due to the fact that, as already mentioned, deep learning for HAR is a fairly recent field and many research directions are yet to be explored. Additionally, the specificity and the diverse characteristics of each HAR setting render the standardization of a unified evaluation method a very complex task. Different experimental setups, dissimilarities in datasets, and the varying scope among studies, generally lead to skewed comparison. Hence, coming up with a rigorous universal approach of evaluation for DL in the context of HAR is an open area of work for future research.

References

[1] A. Akbari and R. Jafari. Transferring activity recognition models for new wearable sensors with deep generative domain adaptation. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, IPSN '19, page 85–96, New York, NY, USA, 2019. Association for Computing Machinery.

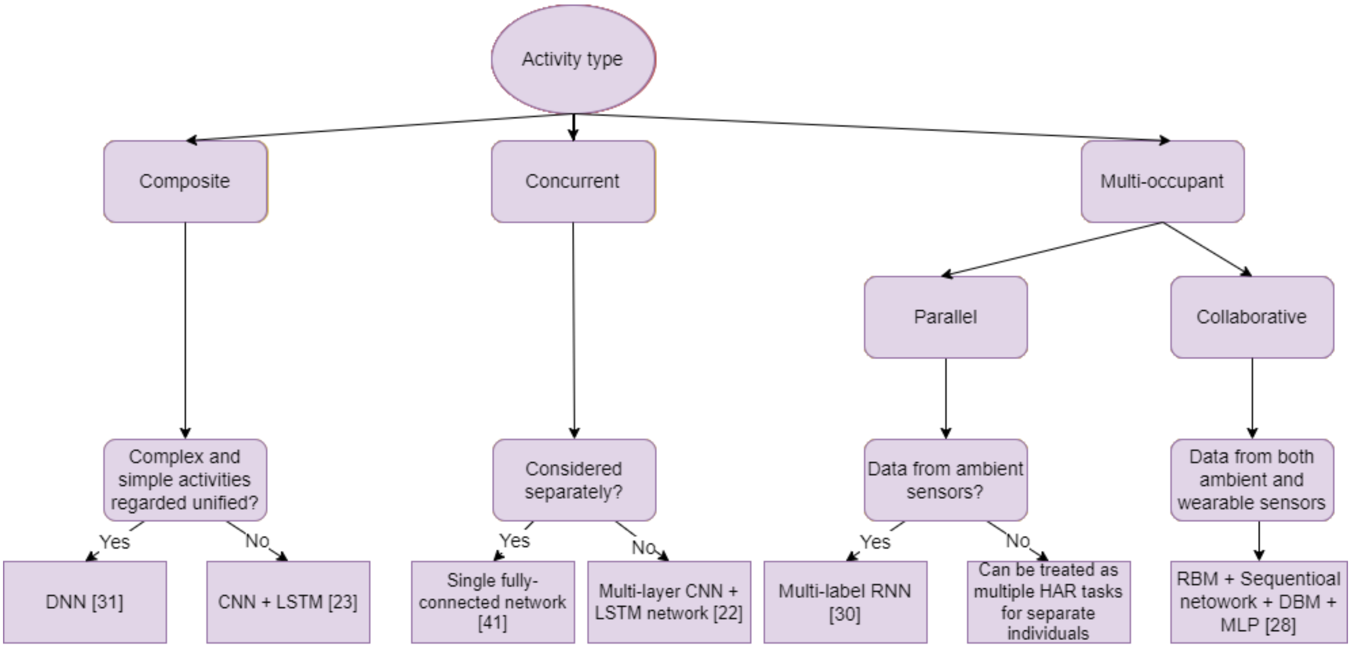

Activity type
Composite
Concurrent
Multi-occupant
Parallel
Collaborative
Complex and simple activities regarded unified?
Considered separately?
Data from ambient sensors?
Data from both ambient and wearable sensors
Yes
No
DNN [31]
CNN + LSTM [23]
Single fully-connected network [41]
Multi-layer CNN + LSTM network [22]
Multi-label RNN [30]
Can be treated as multiple HAR tasks for separate individuals
RBM + Sequential netowork + DBM + MLP [28]

Fig. 4. Visualized guidelines for choosing the DL approach for the corresponding activity types along with references to relevant papers.

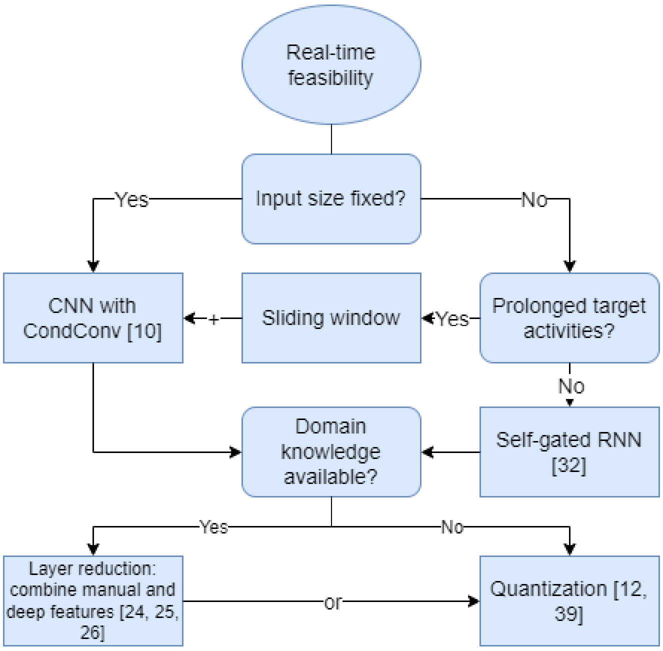


Fig. 5. Visualized guidelines for choosing the DL approach for the real-time HAR application along with references to relevant papers.

[2] M. A. Alsheikh, A. Selim, D. Niyato, L. Doyle, S. Lin, and H. P. Tan. Deep activity recognition models with triaxial accelerometers. *CoRR*, abs/1511.04664, 2015.

[3] Y. Bengio, Y. Lecun, and G. Hinton. Deep learning for ai. *Communications of the ACM*, 64(7):58–65, 2021.

[4] D. Bouchabou, S. Nguyen, C. Lohr, B. Leduc, and I. Kanellos. Fully convolutional network bootstrapped by word encoding and embedding for activity recognition in smart homes, 2020.

[5] D. Bouchabou, S. M. Nguyen, C. Lohr, B. Leduc, and I. Kanellos. Fully convolutional network bootstrapped by word encoding and embedding for activity recognition in smart homes. In *International Workshop on Deep Learning for Human Activity Recognition*, pages 111–125. Springer, 2021.

[6] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina. An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Future Internet*, 12:113, 07 2020.

[7] K. Chen, L. Yao, D. Zhang, X. Wang, X. Chang, and F. Nie. A semisupervised recurrent convolutional attention model for human activity recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 31(5):1747–1756, 2020.

[8] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu. Deep learning for sensor-based human activity recognition: Overview, challenges and opportunities, 2021.

[9] Y. Chen, J. Wang, M. Huang, and H. Yu. Cross-position activity recognition with stratified transfer learning. *Pervasive and Mobile Computing*, 57:1–13, 2019.

[10] X. Cheng, L. Zhang, Y. Tang, Y. Liu, H. Wu, and J. He. Real-time human activity recognition using conditionally parametrized convolutions on mobile and wearable devices, 2020.

[11] B. Chikhaoui and F. Gouineau. Towards automatic feature extraction for activity recognition from wearable sensors: A deep learning approach. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 693–702, 2017.

[12] M. Edel and E. Köppe. Binarized-blstm-rnn based human activity recognition. In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–7, 2016.

[13] W. Gerych, H. Kim, J. DeOliveira, M. Martin, L. Buquicchio, K. Chandrasekaran, A. Alajaji, H. Mansoor, E. Rundensteiner, and E. Agu. Gan for generating user-specific human activity data from an incomplete training corpus. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 4705–4714, 2021.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[15] F. Gu, K. Khoshelham, S. Valaee, J. Shang, and R. Zhang. Locomotion activity recognition using stacked denoising autoencoders. *IEEE Internet of Things Journal*, 5(3):2085–2093, 2018.

[16] G. K. Gudur, P. Sundaramoorthy, and V. Umaashankar. Activeharnet: Towards on-device deep bayesian active learning for human activity recognition. *CoRR*, abs/1906.00108, 2019.

[17] N. Y. Hammerla, S. Halloran, and T. Ploetz. Deep, convolutional, and recurrent models for human activity recognition using wearables, 2016.

[18] A. Ignatov. Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied Soft Computing*, 62:915–922, 2018.

[19] G. Katz, C. Caragea, and A. Shabtai. Vertical ensemble co-training for text classification. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(2):1–23, 2017.

[20] A. Mathur, T. Zhang, S. Bhattacharya, P. Veličković, L. Joffe, N. D. Lane, F. Kawsar, and P. Lió. Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '18, page 200–211. IEEE Press, 2018.

[21] S. Matsui, N. Inoue, Y. Akagi, G. Nagino, and K. Shinoda. User adaptation of convolutional neural network for human activity recognition. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 753–757. IEEE, 2017.

[22] T. OKITA and S. INOUE. Recognition of multiple overlapping activities using compositional cnn-lstm model. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, UbiComp '17, page 165–168, New York, NY, USA, 2017. Association for Computing Machinery.

[23] L. Peng, L. Chen, Z. Ye, and Y. Zhang. Aroma: A deep multi-task learning based simple and complex human activity recognition method using wearable sensors. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(2), jul 2018.

[24] I. M. Pires, N. Pombo, N. M. Garcia, and F. Flórez-Revuelta. Multi-sensor mobile platform for the recognition of activities of daily living and their environments based on artificial neural networks. IJCAI'18, page 5850–5852. AAAI Press, 2018.

[25] D. Ravi, C. Wong, B. Lo, and G.-Z. Yang. Deep learning for human activity recognition: A resource efficient implementation on low-power devices. In *2016 IEEE 13th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 71–76, 2016.

[26] D. Ravì, C. Wong, B. Lo, and G.-Z. Yang. A deep learning approach to on-node sensor data analytics for mobile or wearable devices. *IEEE Journal of Biomedical and Health Informatics*, 21(1):56–64, 2017.

[27] S. A. Rokni, M. Nourollahi, and H. Ghasemzadeh. Personalized human activity recognition using convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[28] S. Rossi, R. Capasso, G. Acampora, and M. Staffa. A multimodal deep learning network for group activity recognition. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2018.

[29] E. Soleimani and E. Nazerfard. Cross-subject transfer learning in human activity recognition systems using generative adversarial networks. *Neurocomputing*, 426:26–34, 2021.

[30] S. Tran, Q. Zhang, V. Smallbon, and M. Karunanithi. Multi-resident activity monitoring in smart homes: A case study. *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 698–703, 2018.

[31] P. Vepakomma, D. De, S. K. Das, and S. Bhansali. A-wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities. In *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 1–6, 2015.

[32] T. H. Vu, A. Dang, L. Dung, and J.-C. Wang. Self-gated recurrent neural networks for human activity recognition on wearable devices. Thematic Workshops '17, page 179–185, New York, NY, USA, 2017. Association for Computing Machinery.

[33] S. Wan, L. Qi, X. Xu, C. Tong, and Z. Gu. Deep learning models for real-time human activity recognition with smartphones. *Mobile Networks and Applications*, 25, 04 2020.

[34] J. Wang, Y. Chen, Y. Gu, Y. Xiao, and H. Pan. Sensorygans: An effective generative adversarial framework for sensor-based human activity recognition. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

[35] J. Wang, V. W. Zheng, Y. Chen, and M. Huang. Deep transfer learning for cross-domain activity recognition. In *Proceedings of the 3rd International Conference on Crowd Science and Engineering*, ICCSE'18, New York, NY, USA, 2018. Association for Computing Machinery.

[36] S. Woo, J. Byun, S. Kim, H. M. Nguyen, J. Im, and D. Kim. Rnn-based personalized activity recognition in multi-person environment using rfid. In *2016 IEEE International Conference on Computer and Information Technology (CIT)*, pages 708–715, 2016.

[37] B. Yang, G. Bender, Q. V. Le, and J. Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference, 2020.

[38] Y. Yang, C. Hou, Y. Lang, D. Guan, D. Huang, and J. Xu. Open-set human activity recognition based on micro-doppler signatures. *Pattern Recognition*, 85:60–69, 2019.

[39] Z. Yang, O. I. Raymond, C. Zhang, Y. Wan, and J. Long. Dfternet: Towards 2-bit dynamic fusion networks for accurate human activity recognition. *IEEE Access*, 6:56750–56764, 2018.

[40] X. Zhang, L. Yao, and F. Yuan. Adversarial variational embedding for robust semi-supervised learning. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2019.

[41] Y. Zhang, X. Li, J. Zhang, S. Chen, M. Zhou, R. A. Farneth, I. Marsic, and R. S. Burd. Car - a deep learning structure for concurrent activity recognition: Poster abstract. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '17, page 299–300, New York, NY, USA, 2017. Association for Computing Machinery.

[42] Y. Zheng, Y. Zhang, K. Qian, G. Zhang, Y. Liu, C. Wu, and Z. Yang. Zero-effort cross-domain gesture recognition with wi-fi. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, page 313–325, New York, NY, USA, 2019. Association for Computing Machinery.

A Comparison of In-Process & In-Network Methods for Global Order of Messages

Arjan Dekker and Jesse Maarleveld

Abstract— Temporal and causal ordering of messages in distributed system is a large area of research. Historically, such message ordering was implemented by coordination between different processes (in-process). Ongoing research has lead to the development of more programmable network switches. One example development is the P4 language for parsing and processing packet headers. Such techniques can be used to implement and enforce a message ordering using the switches in the network, allowing for the development of new algorithms. One such algorithm is the newly proposed method 1Pipe. In this research, we review literature in order to investigate the difference between in-network and traditional in-process methods. We find that in-network methods achieve lower latency, and better throughput and scalability compared to in-process global ordering methods. By comparing 1Pipe with other in-network algorithms, we find that it can be implemented on less expressively programmable devices than those with traditional CPUs, because of the simplicity of the computations.

Index Terms—Total Order Communication, Programmable Switches, In-Network Scheduling, Global Event Order

1 Introduction

Ordering of messages in distributed systems is hard. For many applications the order of messages is important. An example application is distributed database replication, where different ordering can lead to conflicting states. Another application is complex event processing, where events have to be detected in continuous real-time data streams. Wrong ordering could lead to false positives or false negatives. Those mistakes can be crucial in a medical setting for example [8, 16].

In the past, various algorithms, such as sequencers or token-based approaches, have been proposed that tackled this total ordering problem [6]. Especially the early algorithms had many drawbacks, such as poor scalability and lacking the ability to tolerate failures directly, that have been somewhat addressed in later research [18, 7, 12]. Such algorithms always operated at the process level, and were not deployed in the network.

A new development over the recent years, is the development and deployment of programmable switches in data center networks [21, 3, 22, 20]. Previously, it was difficult if not impossible to deploy custom algorithms on switches. Changing algorithms in the field would be almost completely impossible [21, 3].

As a result of these developments, there is more flexibility for network-operators to use – and change – the algorithms to fit their needs. Additionally, programmable switches allow switching from process level handling of message order, to an in-network determined order [15, 16]. An example algorithm using such developments, is 1Pipe: a communication abstraction with a global message order for unicast and multicast, particularly well-suited for use in data centers. 1Pipe can scale its throughput to a total of 256 processes with ease and only have a small decrease in throughput for 512 processes, whereas the throughput of even the best in-process methods deteriorates when the number of processes exceeds 16 [15].

In this research, we want to look at the developments made with regards to fully ordered communication algorithms in literature. We want to investigate what methods do and do not operate in the network. Our goal is to see to what extent such algorithms leverage the power of programmable switches, and what the benefits are of doing so.

In the remainder of the article, we will first discuss some relevant background information, including the programmability of switches, and different types of message ordering. This will be done in section 2. Next, we will provide an overview of various categories of methods for global order of messages in section 3. Then, we will further discuss and compare strengths and weaknesses of those algorithms in section 4. We will further discuss and summarize those findings in section 5. We will end with our final conclusions in section 6.

2 Background

This section contains the background information about different types of message ordering and a number of developments with regards to programmable switches.

2.1 Lamport Causality

One type of totally ordered communication is a causal order of messages. Often, this is causality as defined by Lamport in [13]:

Suppose that we have n processes $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$, with corresponding clocks $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_n$. Let a and b be events (e.g. send or receive) in processes $\mathcal{P}_i$ and $\mathcal{P}_j$, respectively ($i = j$ is allowed). The values of the clocks at the times of these events occurring are given by $\mathcal{C}_i(a)$ and $\mathcal{C}_j(b)$, respectively. Lamport then defines that $a \implies b$ (a occurs before b) by $a \implies b \equiv \mathcal{C}_i(a) < \mathcal{C}_j(b) \vee (\mathcal{C}_i(a) = \mathcal{C}_j(b) \wedge \mathcal{P}_i \prec \mathcal{P}_j)$. Here, $\prec$ is some arbitrary totally ordering on the processes $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ which can be used to break ties and obtain a *total causal order*; otherwise, we only have a partial ordering. An example of such an ordering could be on process ID, e.g. $\mathcal{P}_i \prec \mathcal{P}_j \iff i < j$ [13].

To fully define the causality principle, we also need to define how the clocks $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_n$ should behave. All clocks must satisfy two properties. i) if a and b are events in the same process $\mathcal{P}_i$ with a occurring before b, then $\mathcal{C}_i(a) < \mathcal{C}_i(b)$. ii) if a represents a send operation from process $\mathcal{P}_i$ and b the corresponding receive operation in process $\mathcal{P}_j$, then $\mathcal{C}_i(a) < \mathcal{C}_j(b)$ [13]. The first condition essentially mandates an ordering of events within a single process, while the second property mandates some sort of synchronization between processes (e.g. a receiving process cannot keep having its clock run late).

A simple method which is sufficient to implement this causal ordering, is the so-called Lamport timestamp. Instead of a physical clock, a logical clock is used. Such a clock can satisfy the first condition by simply having a logical clock increase its value when sending a message. A receiving process must then update its clock accordingly when receiving a message [13].

2.2 Broadcast and Multicast

It is important to understand some basic notions regarding broadcast and multicast. Broadcast means sending a message to all devices connected to a network, whereas multicast means sending a message to a subset of all devices in the network. In the limiting case where a message is only sent to one device, we speak about unicast.

- *Arjan Dekker is with University of Groningen, E-mail: a.j.dekker.5@student.rug.nl.*
- *Jesse Maarleveld is with University of Groningen, E-mail: j.maarleveld@student.rug.nl.*

Within systems, we can have open and closed group algorithms for multicast. In closed group algorithms, processes that are not members of the group cannot multicast messages to the group. The opposite group of algorithms is called open group, where any arbitrary process in the network may multicast to a group [6].

2.3 Atomic Operations

Another type of ordered communication, is atomic communication operations. Atomic operation ordering constraints are different from the causal ordering constraints specified by Lamport [1].

Perhaps the easiest operation to understand is *atomic broadcast*. In atomic broadcast, messages broadcast in a network by producing processes (producers), are received by all consuming processes (consumers) in the same order. In the following, we will use "deliver" to say that a message is "delivered" to the application by the underlying network software. There are four properties which must hold for atomic broadcast: i) every consumer delivers a message at most once, and only if it was generated by a producer, ii) if some producer generates a message, then eventually some consumer delivers that message, iii) if some consumer delivers a message, then eventually all consumers deliver that message, iv) if two consumers deliver messages m and m', then they deliver them in the same order [1].

The atomic broadcast problem, can be generalized to the atomic multicast problem. In this version, instead of considering the set of all consumers, we only consider the properties (i), (ii), (iii), and (iv) with respect to the consumers to which the messages were specifically sent (i.e. the members of the multicast group). Intuitively, this means that all members in the multicast group receive a message, or none – and they receive all messages in the same order [1]. Note that this gives rise to only a partial order of messages – there is no enforced global order between disjoint multicast groups.

The final problem, global atomic multicast, is largely equivalent to atomic multicast. However, it has a stronger requirement with respect to the ordering of messages. It also requires a consistent, global ordering of messages across different multicast groups. Formally, this can be stated as follows: there exists an ordering $m_1, m_2, \ldots, m_n$ of messages such that if any consumer delivers m_i before m_j, then $i < j$ [1]. Note that global atomic multicast is somewhat similar to total order. Total ordering dictates the same complete ordering of messages, but does not dictate the other properties of atomic operations.

2.4 Programmable Switches

In this section, we will discuss some developments which made switches programmable. We will look at some languages and hardware features. The goal is to show how programmable switches have been realized, and to give an idea of their potential and their limitations.

2.4.1 Basic Switch Architecture

We will briefly discuss the architecture of network switches, so that it may be understandable how they can be programmed. There are various abstract switch models in literature (e.g. P4-14, P4-16, Banzai), but most of them agree on the basics [3, 23, 20]. Incoming packets are first parsed by a parser [3, 20]. Next, they enter the ingress pipeline. In the ingress pipeline, so-called match+action tables are executed [20]. These tables are used to execute actions which may modify the packet headers, or modify some persistent state [3, 20]. After this, packets are buffered in a queue. Once they are almost ready for departure, they are dequeued, and will pass through the egress pipeline [20]. This pipeline once again features a set of match+action tables. After the actions have been performed, the packet is transmitted [3, 20]. This abstract model is shown in figure 1.

2.4.2 P4

P4 is a high-level programming language for packet processing. P4 can be used to program the handling of custom headers for packets. The language has several aims. The language is re-configurable, allowing re-definition of packet parsing and processing in the field. Additionally, the language is protocol independent. The underlying switch should not be tied to specific packet formats. It is possible to define parsers for packets and match+action tables for processing these headers. Finally, the language is meant to be target independent, like languages as C are not tied to a single particular CPU [3]. The language had an initial version P4-14, superseded by P4-16, targeted towards switches conforming to the P4-14 and P4-16 abstract switch models, respectively [3, 23].

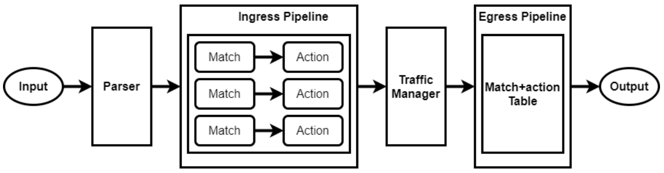


Fig. 1. Simplified architecture of a P4-14 switch ([3]); Also compatible with Banzai ([20]); P4-16 has separate parsers/deparsers in the ingress and egress pipelines [23]. Queues are located in the traffic manager.

In P4, programmers can define a header structure, a parser for the custom headers (using state machines), match+action table entries, and actions. The header structure defines the structure of a header, and specifically the bits per field. The parser definition can be used to determine how to parse custom headers. For instance, some portion of the header may be parsed differently based on some value of another field. Next, the match+action tables are used to determine the next actions of the program, based on some control program. Based on the header field and metadata, actions from this table may be selected and executed. Actions are small "programs" composed of primitive actions directly supported by the switch written as expressions [3, 23]. It should be noted that actions in P4 have no higher level constructs, such as variables or control flow. Finally, the control program is a short program which can trigger switching on the match+action tables, and perform some basic control flow based on metadata to determine what tables to consider [3]. P4 has some support for cross-packet data storage in registers [15].

2.4.3 Domino

P4 uses match+action tables, which are relatively low level. The domain specific language Domino was developed for data-plane programming, trying to provide a level of abstraction closer to that of C (but without loops). Domino is meant to be more expressive than P4, and should be applicable to a larger number of use cases [20]. The language uses the concept of so-called packet transactions: the same piece of code is executed for every packet. This piece of code can, amongst other things, examine headers and schedule packet departure order. Domino targets an abstract machine model called *Banzai*. Under the hood, this machine model also has match+action tables. However, the primitive actions supported by the switch are designed in such a fashion that a simple imperative language can be implemented based on them. Note that it is also possible to have some shared data between transactions. Nowadays, various switches are programmable using P4 [15], but we were unable to find any programmable with Domino.

2.4.4 Customizable Scheduling

P4 enables the parsing and processing of packet headers. However, it does not enable programmable packet scheduling [21, 3]. Domino can be used to program scheduling algorithms [22].

All of this programming is based on switches which support so-called PIFO (push in, first out) queues. Such queues allow inserting packets into an arbitrary place in the queue based on some field, and can be efficiently implemented in hardware. Such queues can be used to implement some ordering based on priority, or based on some scheduled "departure time" (calendar queue). Most scheduling algorithms can be implemented by using either a priority queue or calendar queue (or both), which can thus be implemented using this single hardware primitive – allowing for efficient programmability [21]. One potential way to implement scheduling this way in practice, is have a piece of code compute an index for a packet, and then insert it [20].

3 Total Order Methods: History & State of the Art

In this section, we will be looking at various methods of establishing a total ordering of messages. We will be looking at methods operating in processes (in-process methods), and in-network methods.

3.1 In-process Methods

3.1.1 Sequencer Algorithms

Sequencer algorithms use a single process that is elected to be the sequencer of the messages that are being sent, i.e. the sequencer determines the order of the messages. Since there is only a single sequencer, there cannot be a conflicting order of messages. The order is total, and broadcast and multicast can be made atomic, but not causal.

An example of a sequencer algorithm is the Amoeba group communication system (AGCS) [10], which is used for group communication. Processes that are part of the group can communicate with all other group members with a single message. The AGCS algorithm guarantees that all group members see messages in the same order.

In terms of reliability, AGCS can deal with lost, garbled, and duplicate messages. Besides the reliable variant, the AGCS algorithm also supports an unreliable variant with better performance. Hence, these variants trade-off performance with reliability. However, since the AGCS algorithm had to be implemented in the kernel and the creators of AGCS preferred to keep the kernel and the software simple, they only implemented the reliable variant. That decision forced some users lose part of the performance for reliability, which is not needed for them. The same holds for the point-to-point communication protocol that AGCS supports, as it only support RPC [10].

Since AGCS is a sequencer-based algorithm, only a single process in the group is selected as the sequencer. Due to the implemented fault-tolerance, when the sequencer crashes, a new sequencer is elected by the other processes in the group.

Various derivatives and improvements of the sequence based method exists. One example is Ridge, which uses Paxos to alternate sequencers (called ”distributors”) in order to distribute the workload across multiple sequencers. Ridge uses timestamps in combination with deterministic merging (section 3.2.2) at receivers in order to guarantee a proper order of messages [2].

3.1.2 Token-based Algorithms

Token-based algorithms make use of tokens. Tokens are concise messages that contain the global status of the protocol in a process group. In this process group, tokens are passed from process to process in a predetermined order and therefore the structure of the passing can be seen as a logical ring (see figure 2).

Only the process that possesses the token can send messages. The order of these messages is appended to the global order of messages. When the process that possesses the token is finished with that, it can pass the token to the next process. A causal ordering can be achieved as well, since only one process is able to send messages at any point.

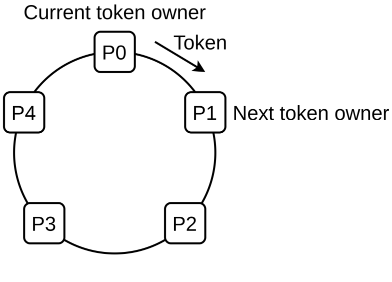


Fig. 2. Visualisation of a token-based algorithm

An example of a token-based algorithm is Token-Passing Multicast (TPM) [19]. TPM guarantees a few properties. First of all, message delivery is atomic. Second, TPM has support for dynamic groups. This means that the group membership can change during a multicast operation and that after the change each member in the group has a consistent view of the group in a finite amount of time. Third, TPM has failure recovery. In the event that the network of the group partitions, the group with the majority membership can continue operation. Fourth, when the network is reconnected again, the portioned groups can regroup again [19].

One of the drawbacks of the naive implementation of a token-based algorithm is the limitation to networks that have a low packet loss rate and a low latency. Essentially only LANs are suitable for token-based algorithms, because of the packet loss rate that is typically less than 0.0001 [19]. However, the Totem protocol makes it possible to have a token-based algorithm on multiple LANs. Exchanging messages between LANs is facilitated by Lamport timestamps (see section 2.1) to deliver the messages in order according to those timestamps [18]. Furthermore, token-based algorithms have a high stability time, i.e. the time it takes until all process have confirmed that they received a message. By using a dynamic solution to pass the tokens around, this stability time can be reduced and the buffer to store unconfirmed messages can be made smaller [12].

Most token-based algorithms use a group-membership service, since those algorithms do not tolerate failures directly [7]. The idea of this service is to detect processes in the group that have crashed and remove them from the group. This is needed to avoid blocking the algorithm, since otherwise the algorithm would wait until the crashed process receives the messages. Besides removing a process, another process is sometimes added to the group to keep the replication degree identical, leading to two costly operations. However, removing a crashed process from the group and adding a new one is not needed when using failure detectors instead of the group member service [7].

3.2 In-network Methods

3.2.1 Tree-Based Methods

One particular method used for totally ordered communication, is a tree based approach. An example is given in [5]. In this paper, an algorithm for reliable, total order broadcast in wireless sensor networks is proposed. Starting from the root, nodes send tokens through the network to their directly adjacent children in a breadth-first fashion, allowing for the construction of a logical tree. To ensure reliability, nodes plan tree repair actions for cases where other nodes fail. Total order is achieved trivially since messages are only broadcast from the root of tree, along edges in the logical tree, and are assumed to be processed in FIFO order by other nodes in the network. The use of the same tree every time guarantees a total ordering of messages. Parent nodes buffer messages, so they can be re-send to new child nodes after tree reconstruction in case of node failure [5]. It is important to note that this approach only allows sending messages from the tree root.

There are also methods which perform only the opposite operation: sending data from all nodes towards one root node. This is referred to as converge cast. One method for this is PPVG (proposed in [4]), which is specifically designed for use in wireless body area networks (i.e. a network of on/in-body sensors). A child node transmits its message to a single parent for a specific amount of times, determined by the reciprocal of the probability of that link existing (based on the position of the person). Parent nodes propagate messages further in the same fashion, until the sink node is reached. Because the same path is always used and FIFO order handling of messages is assumed, the messages from a single node always arrive at the sink node in the same order that they were sent by the original child node. In this sense, this method guarantees total order delivery. On the other hand, it is not reliable, and data loss may be significant [4].

3.2.2 Deterministic Merging

A method for achieving atomic broadcast and global atomic multicast, is through so-called deterministic merging algorithms. In these types of algorithms, message streams are merged in a deterministic manner by nodes in the network, resulting in a consistent ordering of messages [15, 1]. Such algorithms are usually deployed in networks with a (logical) tree structure, which means they are sometimes also called tree-based [15]. One example implementation of deterministic merging is given in [1].

The merging algorithm is fairly straightforward for a many-to-one application. Nodes in the network record the time of the last package they received from senders. For every sender, they also have a bias, based on the rate of incoming messages from said sender. By adding the bias to the time of the last received message, an estimate for the arrival time of the next message is made. The sender with the closest expected arrival time is then determined. The algorithm waits for a message from that sender, updates the time of the last received message for that sender, and further processes (e.g. deliver to application or send to other nodes) the message. Note that this aggregation of information is depend on a tree structure of the network (i.e. no circular information feedback is possible) [1].

The merging algorithm can also be extended to support atomic multicast. The principles remain roughly the same. However, the algorithm no longer waits for a message from a specific sender and instead executes the same procedure when receiving from any producer. Additionally, messages from a node i are now sent with two timestamps (p,t). Here, p is the time at which the previous message was sent, and t is the timestamp at which this message was sent. In receiving nodes, the values $\hat{p} = p + bias[i]$ and $\hat{t} = t + bias[i]$ are stored. Additionally, the last message received from every sender is stored as $last[i] = \hat{t}$. Due to the addition of the bias, $\hat{p}$ can be seen as the time of arrival of the current message, and $\hat{t}$ is the arrival time of the next message. All messages with an arrival time ($\hat{p}$) less than any future arrival time ($\min(last)$), can thus be further processed. For ordering, this is done in ascending order of $\hat{p}$ (with process IDs i used for tie breaks) [1]. One important thing to note that in this particular implementation, node failure was not taken into account.

3.2.3 In-network Multi-Sequencer

One example application of message ordering in networks, is in increasing the performance of databases with multiple shards and replicas. The Eris protocol aims to implement coordination-free, consistent transactions for such database systems. One of its features, is using the network for concurrency control and ordering of transactions [16].

Eris achieves a partial, consistent order of transactions using an in-network multi-sequencer. The usual problem with sequencers, is that all messages have to be send to all receivers, so that they are aware of the global transaction sequence number. This is what makes the sequencer a severe bottleneck [6]. To avoid this problem, Eris has a sequence number per multicast group; hence why we speak of multi-sequencing. Additionally, it introduces the notion of *groupcast*: a communication primitive allowing to send a message to multiple multicast groups at once (in order to support transactions spanning multiple shards) [16]. This method is able to guarantee a consistent ordering of messages, which need not be causal; operations are also not atomic.

3.2.4 1Pipe

We will now examine the in-network algorithm 1Pipe. 1Pipe offers two guarantees: messages are delivered to all receivers in the same order, and 1Pipe delivers messages in causal order [15]. We start with a treatment of the best-effort version of 1Pipe.

1Pipe, when deployed in a data center, makes use of programmable switches and the DAG structure of the routing topology. It makes use of the fact that there are very accurate clock synchronization methods for use in data center networks. A timestamp T is added to messages when they are sent, which are used to determine message order [15].

Since buffer size in switches is small, messages are not buffered in switches themselves, but are forwarded to recipients directly. A so-called barrier timestamp B is attached to all messages, which is updated by all switches in the network. The barrier timestamp is used to determine what messages can be delivered to the application [15].

If a node sends a barrier timestamp B, it promises that all its future messages have a timestamp $\geq B$. It is computed as the minimum over the timestamps of all messages that still have to be sent. Only messages with a timestamp $T < B$ are delivered to the application. Aggregating the timestamp information requires a hierarchical (DAG) structure (either logical or physical) of the network. This makes 1Pipe especially efficient in data center networks [15].

When a node has no messages to send, it may periodically send a beacon after a specific time interval, in order to update the barrier timestamp of other devices, to make sure that it does not halt the delivery of other messages. If a node sends no messages or beacon for some specified amount of time, it is considered dead.

1Pipe can also be adjusted to be reliable. Reliable 1Pipe makes the additional promise of failure atomicity: multicast messages are either received by all or none of the target processes. To deal with packet loss and crashing nodes, a 2 phase commit (2PC) system is used. In the preparation phase, the sender sends messages to its receivers, who respond with acknowledgements (ACK). The receivers do not update their barrier timestamp based on these messages yet. During the commit phase, the sender sends a commit message containing a commit barrier B'. This commit is send to all neighbouring switches, who use it to update their own commit barrier, and in turn send new commit barriers to their own neighbours. The commit barriers serve the same purpose as the original barrier timestamps. The ACKs are used to detect and recover from packet loss [15].

To deal with device failure, 1Pipe uses a central controller (e.g. elected using Paxos). The central controller watches for failed (disconnected) devices. Once it detects one, it broadcasts the failed process P and a failure timestamp T' (maximum last commit timestamp reported by neighbours of P). Processes discard messages from P with timestamp $> T'$, processes discard messages to P in their send buffer, and, in case of a scattering operation, a recall message is send so that no processes in the group receive the message. Finally, failure callbacks are executed in the processes involved in the failed delivery [15].

4 Comparison

In this section, we will highlight various drawbacks, benefits, or performance characteristics of the algorithms covered in section 3.

4.1 In-process Methods

One general remark that can be made about in-process methods is about reliability. In-process algorithms are not limited in that regard. Various fault-tolerance methods can be used to improve reliability, such as failure detectors and group membership services.

The following subsections contain the comparisons that are specific for the corresponding type of in-process algorithm.

4.1.1 Sequencer Algorithms

The first drawback of sequencer algorithms such as AGCS is scalability. The paper of AGCS mentions that its scalability depends on the message processing time. In practice, depending on a single process (the sequencer), is not scalable. Another drawback of AGCS in comparison with open-group algorithms is the fact that message transmission is limited to processes that are part of the same group, e.g. AGCS is a closed-group protocol. This limits the possibility of sending messages to other destinations outside the group, whereas with open-group algorithms this is possible [10].

More sophisticated sequencer algorithms such as Ridge can achieve high throughput and low latency, which does not deteriorate as rapidly as the number of destinations increases. However, especially for reliable messaging, there is significant communication overhead [2].

4.1.2 Token-based Algorithms

Token-based algorithms such as TPM are considered to be relatively simple, and they also maintain a relatively small amount of state information [19]. Token-based algorithms are considered to have an efficient throughput [7]. However, they are not very scalable, due to the ring structure which is used. As expected, when the number of processes in the ring becomes larger, it takes longer before a process is allowed to broadcast, thus increasing the latency. On top of that, most token-based algorithms use a predefined token passing order. A drawback of this is that it can take, in worst case scenario, a full circle before the required processes have confirmed that they received the message. However, this problem is partly solved by using a dynamic token passing scheme such as DTP [12]. DTP reduces both the stability time and buffer size needed to store unconfirmed messages.

Due to the fact that a token is passed around in the group of processes, it essentially only performs well in low latency and low packet loss environments [18]. This is somewhat overcome in the Totem protocol. By using token-rings *in* and Lamport timestamps *between* LANs for total order of messages, Totem can be used as a communication protocol between different LANs.

Another drawback of most token-based protocols is the usage of the group-membership service. This essentially implies two expensive operations when a process crashes. However, this drawback is dealt with by using failure detectors for detecting crashed processes [7]. The impact of the failure detector on the performance of the algorithm was not tested. Therefore it is currently impossible to determine the drawbacks of using the failure detector regarding that aspect.

Similar to sequencer algorithms, token-based algorithms do not allow a broadcast or multicast to processes that are not part of the group.

4.2 In-Network Methods

The following subsections outline the performance characteristics, benefits, and drawbacks of in-network methods.

4.2.1 Tree-Based Methods

Tree-based algorithms are not necessarily suitable for in-network use in datacenters. The algorithms they use cannot easily (if at all) be expressed in terms of the functions provided by a switch [5, 4, 15]. Instead, such algorithms are more often deployed in sensor networks, where they are more suitable [5, 4]. Additionally, the size of the buffers required increases linearly with the height of the tree [5]. At some point, this will be more difficult to scale, due to the limited memory of switch devices [15]. Once the buffers are no longer sufficient, reliability begins to suffer [5]. Tree-based algorithms in sensor networks make little to no guarantee regarding reliability [5, 4].

4.2.2 Deterministic Merging

Deterministic merging has similar ideas, compared to 1Pipe. Both use timestamps for ordering of messages [15, 1]. Deterministic merging was traditionally difficult to implement on switches, because of small per-port buffer sizes and the lack of ability to change the order of packets in the buffer, which might be necessary [15]. However, PIFO queues could be used to handle the ordering of packets, leaving only the buffer size as a problem. The authors are not aware of any research investigating this.

4.2.3 In-Network Multi-Sequencer

Eris ([16]) uses an in-network multi-sequencer as part of its design. It tries to overcome classical problems with global sequencers by only enforcing a partial ordering. However, this still means that there must be a separate sequence number register for every multicast group. On resource-constrained devices such as switches, this limit may be reached relatively quickly [16]. Of course, this method can still be slow when multicast groups are large.

The sequencer can be implemented on many different devices, including programmable switches. One of the drawbacks of implementing the algorithm on devices with less high-level programmable features, such as switches, is that delivery is only best effort (e.g. cannot re-send messages), although dropped messages can be *detected* [16].

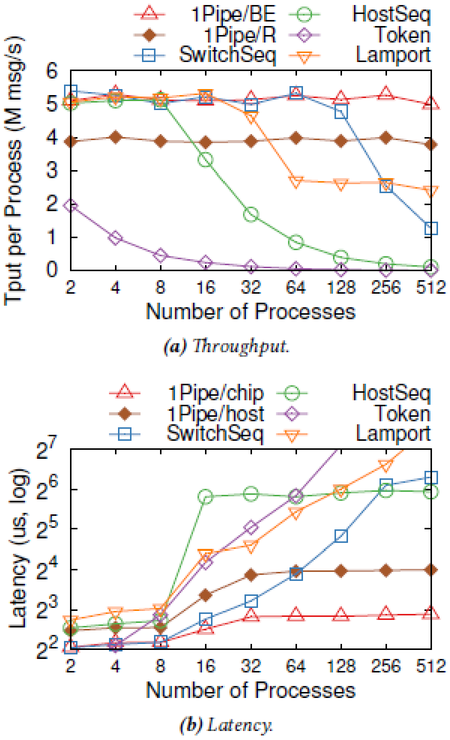
When deployed on an end-host (like in-process sequencing), the throughput is $1.61 * 10^6$ packets/second. On middleboxes, throughput of $6.19 * 10^6$ packets/second was observed. It is expected that performance on switches will be even better, but this was not tested [16].

4.2.4 1Pipe

The scalability of 1Pipe, in comparison with various other algorithms, was tested in [15]. The results are presented here in figure 3. From figure 3a, it can be seen that both best effort and reliable 1Pipe keep roughly the same throughput when the number of processes increases, while performance deteriorates with other algorithms. Other algorithms may first have similar performance compared to 1Pipe for a small number of destinations, but it deteriorates when the number of processes is increased.

In figure 3b, we can see the latency for various algorithms. Once again, 1Pipe performs best when the number of processes becomes sufficiently large. For most algorithms, latency tends to increase rather steeply. For 1Pipe, it increases slower, both at hosts and switches.

The overall takeaway is that 1Pipe tends to be more scalable than other algorithms. The main scalability problems come when scaling to larger networks. Latency issues can increase clock skew, which affects the performance of 1Pipe – though not the correctness. Additionally, there may be some overhead from beacon hops. However, the largest bottleneck is the centralized coordination of failure handling in reliable 1Pipe. This may cause significant performance overhead [15].



(a) *Throughput.*

(b) *Latency.*

Fig. 3. Comparison of 1Pipe with various other algorithms. 1Pipe/BE: best effort 1Pipe; 1Pipe/R: Reliable 1Pipe; SwitchSeq: In-network sequencer (global sequencing); HostSeq: Centralized Sequencer; Token: Token Based; Lamport: Lamport Timestamp Based. Measurements are for total order broadcast. Image source: [15]

5 Discussion

Based on the discussion of various algorithms in section 4, we can make some observations.

One of the major observations is that it seems that in-network scheduling methods for global message order can offer superior throughput and lower latency. Figure 3 in particular provides strong evidence for this. 1Pipe clearly performs better than all other algorithms in terms of both throughput and latency. The next best method seems to be in-network sequencing. Hence, in-network methods seem to out-perform other methods. This is also in accordance with literature, which says that in-process methods such as token based algorithms or global sequencing do not scale well [10, 7]. It would be interesting to see a comparison between 1Pipe and the multi-sequencing algorithm implemented for Eris, although preliminary results from their respective papers show that multi-sequencing might be able to achieve even better throughput than 1Pipe [15, 16].

In addition, we saw that 1Pipe, in contrary to many other algorithms, does not significantly deteriorate in performance as the number of process is increased. Hence, we can conclude that it is more scalable. This could possibly be attributed to the fact that 1Pipe does comparatively little computation inside the network: it only needs to track barrier timestamps, and perhaps some additional bookkeeping for reliable 1Pipe. This allows for less slowdown inside the network, and less performance deterioration when adding more processes.

Not all in-network methods can be applied in data-center context. In fact, many algorithms for in-network event order seem to be domain specific and operate at levels of detail unfit for programmable switches. Tree-based methods tend to have poor or incomplete handling of message loss, with converge cast algorithms essentially only avoiding message loss by sending a message an optimal amount of times [4]. In fact, the converge cast algorithm proposed in [4] would be impossible to implement on switches, because it requires looping, which is not usually supported by switch programming languages [3, 20]. Likewise, deterministic merging is also infeasible to implement on switches because the algorithm requires sufficiently large buffers for messages. Note that in 1Pipe, which is similar in idea compared to deterministic merging, this is avoided by forwarding all messages to recipients directly, and instead managing order and causality on the recipient using timestamps and barrier timestamps [15].

We also saw that the ordering and reliability guarantees offered by the different algorithms were widely different. A consistent and causal order, combined with reliable messaging, makes programming easier. This can be achieved by sequencers and token-based algorithms. 1Pipe is also able to guarantee all of this, contrary to other in-network algorithms. Additionally, it is able to do so in a scalable manner on relatively cheap and fast hardware (switches).

6 Conclusion

In this research, we examined various methods for global ordering of messages. We saw that in-network methods can achieve superior performance compared to direct coordination between applications. However, several classes of algorithms are either domain specific or unfit for use on switches, thus requiring devices with more expressive programmability. 1Pipe is an algorithm which offers the best of many worlds: it can run on relatively simple yet powerful hardware, is scalable, provides total and causal ordering, and is reliable. It shows how development in programmable switches can enable the development of powerful and scalable technologies in the data centers.

A number of technologies were not considered in this paper due to time and space constraints. They could potentially be included in a more elaborate review in future research. We will list some potential research directions here, although a more elaborate literature review could be performed to ensure no additional significant methods were missed by this research. In [9], an atomic multicast algorithm with a Paxos-like mechanism with a leader responsible for ordering messages is presented. In [17], Ring-Paxos is proposed, another algorithm for atomic multicast. Another algorithm, RamCast, was designed for RDMA-based atomic multicast in shared memory environments [14]. Finally, TEA is an RDMA-based algorithm for increasing storage capabilities of switches by using server DRAM, which could potentially be used for future interesting in-network ordering algorithms [11].

Acknowledgements

The authors wish to thank Bochra Boughzala for providing the initial papers, and reviewing the first draft of this paper. The authors also wish to thank Auke Roorda and Arjan Tilstra for reviewing the draft.

References

[1] M. K. Aguilera and R. E. Strom. Efficient atomic broadcast using deterministic merge. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’00, page 209–218, New York, NY, USA, 2000. Association for Computing Machinery.

[2] C. E. Bezerra, D. Cason, and F. Pedone. Ridge: High-throughput, low-latency atomic multicast. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pages 256–265, 2015.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. 44(3):87–95, jul 2014.

[4] G. Bu and M. Potop-Butucaru. Total order reliable convergecast in wban. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN ’17, New York, NY, USA, 2017. Association for Computing Machinery.

[5] S. Chakraborty, S. Chakraborty, S. Nandi, and S. Karmakar. A reliable and total order tree based broadcast in wireless sensor network. In *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*, pages 618–623, 2011.

[6] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, dec 2004.

[7] R. Ekwall, A. Schiper, and P. Urban. Token-based atomic broadcast using unreliable failure detectors. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pages 52–65, 2004.

[8] P. Fodor, D. Anicic, and S. Rudolph. Results on out-of-order event processing. In R. Rocha and J. Launchbury, editors, *Practical Aspects of Declarative Languages*, pages 220–234, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[9] A. Gotsman, A. Lefort, and G. V. Chockler. White-box atomic multicast. *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 176–187, 2019.

[10] M. Kaashoek and A. Tanenbaum. An evaluation of the amoeba group communication system. In *Proceedings of 16th International Conference on Distributed Computing Systems*, pages 436–447, 1996.

[11] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan. Tea: Enabling state-intensive network functions on programmable switches. SIGCOMM ’20, page 90–106, New York, NY, USA, 2020. Association for Computing Machinery.

[12] J.-S. Kim and C. Kim. A total ordering protocol using a dynamic token-passing scheme. *Distributed Syst. Eng.*, 4:87–95, 1997.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, jul 1978.

[14] L. H. Le, M. Eslahi-Kelorazi, P. Coelho, and F. Pedone. Ramcast: Rdma-based atomic multicast. Middleware ’21, page 172–184, New York, NY, USA, 2021. Association for Computing Machinery.

[15] B. Li, G. Zuo, W. Bai, and L. Zhang. 1pipe: Scalable total order communication in data center networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM ’21, page 78–92, New York, NY, USA, 2021. Association for Computing Machinery.

[16] J. Li, E. Michael, and D. R. K. Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP ’17, page 104–120, New York, NY, USA, 2017. Association for Computing Machinery.

[17] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone. Ring paxos: A high-throughput atomic broadcast protocol. In *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pages 527–536, 2010.

[18] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Commun. ACM*, 39(4):54–63, apr 1996.

[19] B. Rajagopalan and P. McKinley. A token-based protocol for reliable, ordered multicast communication. In *Proceedings of the Eighth Symposium on Reliable Distributed Systems*, pages 84–93, 1989.

[20] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM ’16, page 15–28, New York, NY, USA, 2016. Association for Computing Machinery.

[21] A. Sivaraman, S. Subramanian, A. Agrawal, S. Chole, S.-T. Chuang, T. Edsall, M. Alizadeh, S. Katti, N. McKeown, and H. Balakrishnan. Towards programmable packet scheduling. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, New York, NY, USA, 2015. Association for Computing Machinery.

[22] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable packet scheduling at line rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM ’16, page 44–57, New York, NY, USA, 2016. Association for Computing Machinery.

[23] The P4 Language Consortium. The p4-16 programming language. https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html#sec-vss-arch.

university of groningen

faculty of science and engineering

computing science