university of groningen

faculty of science and engineering

computing science

SC@RUG 2020 proceedings

17th SC@RUG 2019-2020

Rein Smedinga, Michael Biehl (editors)

rug.nl/research/**bernoulli**

SC@RUG 2020 proceedings

Rein Smedinga
Michael Biehl
editors

2020
Groningen

ISBN (e-pub): 978-94-034-2766-9
Publisher: Bibliotheek der R.U.
Title: 17th SC@RUG proceedings 2019-2020
Computing Science, University of Groningen
NUR-code: 980

About SC@RUG 2020

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 seventeenth time in the academic year 2019-2020. Students wrote a paper, participated in the review process and gave a presentation. Due to the corona virus there was no symposium this year.

The organizers Rein Smedinga and Michael Biehl would like to thank all colleagues who cooperated in this SC@RUG by suggesting sets of papers to be used by the students and by being expert reviewers during the review process. They also would like to thank Henk Klabbers for giving additional lectures workshops on presentation techniques and speech skills.

Organizational matters

SC@RUG 2020 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 and on scientific integrity and a workshop on reviewing were given by Michael Biehl

Henk Klabbers gave a lecture 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. Henk Klabbers, Rein Smedinga and teaching assistent Rick van Veen were present during these presentations.

Unfortunately the workshops on presentation skills (to be given by Henk Klabbers) and the actual conference itself were cancelled due to corona virus measures. So this year no organition of this conference by the students themselves, no session chairing and no final presentations.

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 on the 2 minute madness presentation.
Because there were no final presentations and students did not need to organise the conference itself or act as a chair during one of the sessions, we had to redefine the final grading as follows:
The draft paper accounted for 20%, the final paper for 40%, the 2 minute madness presentation for 20% and the review and re-review process both for 10%
For the grading of the presentations we used the assessments from the audience and calculated the average of these.

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).

In this edition of SC@RUG students were videotaped during their 2 minute madness presentation using the video recording facilities of the University. The recordings were published on *Nestor* for self reflection.

Website

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

Sponsoring

Since there was no final conference, there was no sponsoring this year.

[1]this year, all papers were accepted

Thanks

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

- Alen Arslanagic
- Anja Reuter
- Arash Yadegari Ghahaderijani
- Estefania Talavera Martinez
- Fadi Mohson
- Fatih Turkman
- Frank Blaauw
- Héctor Cadavid Rengifo
- Jie Tan
- Jiri Kosinka
- George Azzopardi
- Michael Biehl
- Michel Medema
- Vasilios Andrikopulos
- Simon Gazagnes

and all other staff members who provided topics and provided sets of papers.
Also, the organizers would like to thank the *Graduate school of Science* for making it possible to publish these proceedings and sponsoring the awards for best presentations and best paper for this conference.

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

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

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

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

A survey of Encryption Algorithms in IoT

Kaavyaa Stalin Thara, Pranav Gupta Vallala

Abstract—Protecting sensitive information is a significant problem in the Internet of Things (IoT) devices. There are many types of symmetric and asymmetric encryption algorithms with a different set of requirements to protect the data in IoT devices. This paper compares the symmetric algorithms such as Advanced Encryption Standard, Chacha20-Poly1305 and two asymmetric algorithms such as Rivest–Shamir–Adleman and Elliptic curve cryptography encryption algorithms. As per the survey, we provided the importance encryption methods in IoT device and the findings give insight which encryption methods are most beneficial to use in IoT devices for data protection.
The insights are obtained, analysing the requirements with efficiency and reviewing the scalability of encryption algorithms in IoT devices under each encryption methods. Requirements are estimated by implementation cost, encrypting time and the risk factor whereas the efficiency of each encryption algorithm is measured by the consumption of power when the size of a message is applied to encrypt/decrypt. We also discussed the security measures of each method in the IoT device. The conclusion of this research shows that all encryption methods have their own advantages and disadvantages. Based on the findings, Advanced Encryption Standard and Chacha20-Poly1305 symmetric encryption algorithm are cost-efficient, faster and useful methods to protect the information in small IoT devices. Whereas Rivest–Shamir–Adleman and Elliptic curve cryptography asymmetric encryption algorithms are most efficient method to handle more informations in the IoT devices. So, the encryption algorithm can be used depending on the requirements of the IoT device.

Index Terms—Symmetric methods, Advanced Encryption Standard, Chacha20-Poly1305, Asymmetric methods Rivest–Shamir–Adleman, Elliptic curve cryptography.

1 Introduction

Networking of physical objects which contains electronics embedded into them is called the Internet of Things (IoT). These objects communicate and sense interactions among each other or with an external environment. Advancements in Power, agriculture, medicine, smart homes and cities, are just some of the few examples where IoT is strongly established. Data exchange between these devices over the internet are rapidly increasing. In turn, this generates more security and privacy risks for the users of these devices, which is currently one of the biggest challenges of the IoT [15]. Cryptography techniques such as symmetric and asymmetric encryption algorithms are developed to handle the data loss, security issues and protect the device from hacker/attacker.

The main intention of this paper is to provide details about the usage of encryption algorithms in IoT devices. Many different types of encryption algorithms are available to protect the data, we only focus on the two different symmetric encryption algorithm such as Advanced Encryption Standard, Chacha20-Poly1305 and two asymmetric encryption algorithm such as RSA and Elliptic Curve Cryptography because these four encryption algorithms are widely used in different type of IoT devices.

These different type of encryption methods are explained in-depth and compared against each other with requirements, efficiency and scalability in IoT device. Whereas, we measured the requirements based on how much cost is required to implement and what is the risk factor while implementing in the IoT devices. Then we evaluated the efficiency based on how much power consumption is needed when it is protecting the sensitive data in the IoT devices. Finally, scalability is estimated by the security measures of each encryption algorithm.

The paper designed as follows: In section 2, we discussed the background information of symmetric and asymmetric encryption algorithms to protect the data. We explain selected symmetric encryption algorithms and briefly describe their advantages and disadvantages in section 3. Then in section 4, we explain the asymmetric encryption algorithms and briefly describe the implementation process with their strength and weaknesses. Based on the requirements, efficiency and scalability of IoT devices we discussed the comparison between each encryption algorithm and their benefits in the IoT devices in section 5. Then, we summarize our findings in section 6. Finally, in section 7, we added our ideas to implement in future.

- *Kaavyaa Stalin Thara, E-mail: s.t.k.stalin.thara@student.rug.nl.*
- *Pranav Guptha Vallala E-mail: p.g.vallala@student.rug.nl.*

Manuscript received 3 February 2020; accepted 10 February 2020; mailed on 23 March 2020.
For information on obtaining reprints of this article, please send e-mail to: s.t.k.stalin.thara@student.rug.nl p.g.vallala@student.rug.nl .

2 Background

The main aim of cryptography is to apply an encryption algorithm in the Internet of Things to secure the data communication between the devices. There are three different types of cryptography algorithms: symmetric, asymmetric and hash function.

The symmetric algorithm is known as a same secret key at both ends, for instance, the original message (plain text) is encrypted by using a key and the encrypted message called as a cipher text. Then the cipher text is decrypted by using the same secret key to show the original information. The process shown in Figure 1. The main drawback of the symmetric key encryption is that all the parties that are involved should exchange the same key that is used to encrypt the data. AES(Advanced Encryption Standard), DES(Data Encryption Standard), Blowfish and RC4 (Rivest Cipher) are the subtypes of symmetric encryption. In section 4 we discussed the sub-types of symmetric encryption algorithm in detail.

An asymmetric encryption algorithm is different from symmetric encryption because of the pair-key rule used in asymmetric encryption process. It represents that both the public and private key used to encrypt and decrypt pieces of information in the asymmetric algorithm. The Figure 2 shows the process of encryption of data by using asymmetric algorithm. Elliptic curve cryptography(ECC), Diffie-Hellman and Rivest–Shamir–Adleman(RSA) are types of asymmetric encryption algorithm that are used widely to create a digital signature to secure the information. Moreover, this algorithm will verify the message flow between the sender and receiver node by using the public key to secure the information. On the other hand, the user needs to keep the private key safely. If the user loses the private

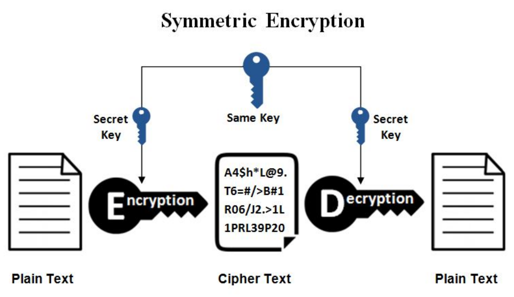


Fig. 1. Symmetric Encryption Process [1].

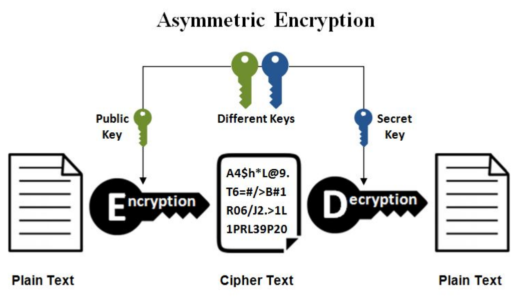


Fig. 2. Asymmetric Encryption Process[1].

key then the regeneration of the same private key is not possible, which leads to major problems. Firstly, the user will not be able to read the new information and will be unable to delete previous communications. Secondly, if the hacker identifies the private key then the hacker can read all communications. The additional shortcoming of the asymmetric encryption algorithm process is that it is slow while handling large datasets of encryption.

In addition to it, cryptography techniques are an essential, effective and efficient component to ensure the secure communication between the different entities by transferring unintelligible information and only the authorized recipient can be able to access the information [11]. Both symmetric and asymmetric encryption are key-based encryption algorithms to secure the information. We briefly discuss them in forthcoming section 3 and section 4.

3 SYMMETRIC METHODS

Cryptography means protecting private information against unauthorized access in situations where it is difficult to provide physical security [9] [18]. Symmetric encryption techniques are a type of cryptography techniques which provides an efficient method of securing communication between the IoT devices. Various sub types of symmetric encryption methods are there to encrypt and decrypt the information, but we have taken only two types of symmetric encryption algorithms which are Advanced Encryption Standard and Chacha20-Poly1305. These methods are designed for different types of requirements to encrypt and it varies from each other as discussed in section 3.1 and 3.2.

3.1 Advanced Encryption Standard

Advanced Encryption Standard is a encryption technology to provide a shield to any dataset that contains sensitive information in the IoT devices. This encryption method known as substitution and permutation network is a number of mathematical operations are carried out in block cipher algorithms [1]. Using the mathematical operations, original message are encrypted into set of numbers and alphabets. This process is known as encryption and the encrypted message known as cipher text.

In [10] the authors explain the implementation process of cost effective Advanced Encryption Standard algorithm. Development steps shown in Figure 3 and described as follows:

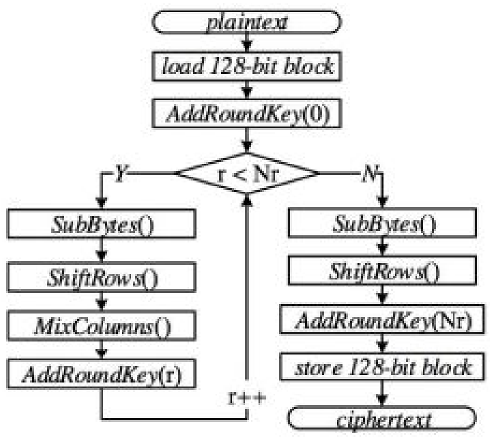
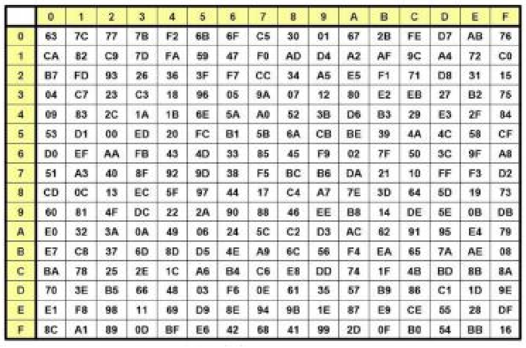


Fig. 3. AES Workflow[10]

1. The input message is known as plain text which will be stored in the AES 128-bit block and as per the size of input message the block size can able to change as 128,192 or 256-bits, this process are called as round key. A number of rounds are repeated in the AES, *Nr*, is represented by the length of the key, which can be 10, 12 or 14 for key lengths of 128, 192 or 256-bits. [10].

2. The condition ($r < Nr$), r represents the list of letters need to encrypt and Nr represents the number of key allocated for encrypting the entire message. This condition starts to work once the block size of the encryption process is fixed.

3. **SubBytes** are used to split each letter separately from the input message to convert into bytes and evaluate according to the lookup table as shown in Figure 4.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Fig. 4. Lookup Table(S box Table)[1]

4. **ShiftRows** each of these rows is shifted to the left by a set amount: their row number starting with zero and then top row is not shifted at all, the next row is shifted by one[2],as shown in Figure 5.

One major advantage of the Advanced Encryption Standard is that it produces a high level security to the data in the IoT designs while

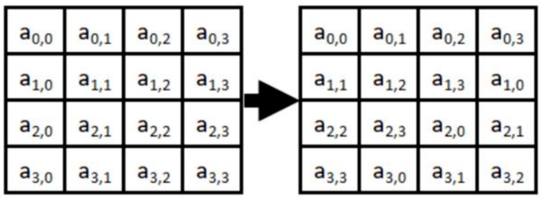

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Fig. 5. ShiftRows[1]

analysing to other symmetric algorithms. Moreover, the speed of the encryption and decryption method are comparatively high for the small key-size. On the other hand, first stage of encryption process is to convert the message to subBytes. The subBytes of each block can be read by converting it into binary digits. So, if hacker/attacker enters the first stage of encryption process then they can able to read the subBytes.

3.2 ChaCha20-Poly1305

Chacha20 is an encryption technique and Poly1305 is a cryptographic message authentication code.On a general purpouse 32-bit(or greater) CPU without dedicated instructions, Chacha20 is generally faster than AES. The reason for this is because of the mathematical operations such as addition, multiplication, rotation and XOR that are used to encrypt and decrypt the messages compared to binary digits in Advanced Encryption Standard(AES) that are used for encryption to secure the messages. In addition to that, the developer will not be needed to set up the lookup table for Chacha20-Poly1305 and it's easy to implement in IoT devices, whereas in Advanced Encryption Standard, the developer needs to set up the lookup table as shown in Figure 4 to provide more efficiency while encrypting the messages and the implementation process of Advanced Encryption Standard in IoT devices are challenging. For instance, Chacha20/Poly1305 has already been adopted and deployed by major companies such as Google (Chrome browser, Android mobile devices) and Apple (Apple HomeKit for IoT devices) [4]. The workflow of Chacha20-Poly1305 is described as follow and Figure is shown in 6.

1. Depending upon the input message, it will generate the size of the key. The generated key remains the same for both encryption/decryption and perform an XOR function by using the generated key also known as streamed key.

2. Poly1350 are used to validate the encrypted message.

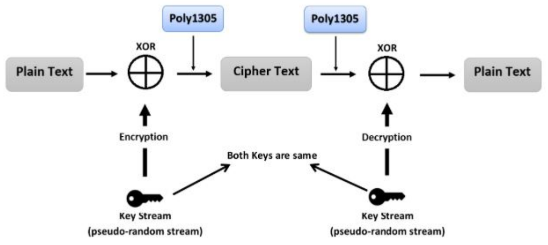


Fig. 6. Workflow of Chacha20-Poly1350[6]

4 ASYMMETRIC METHODS

There are many asymmetric encryption methods are used to authenticate, validate and secure the datas in IoT devices by generating the keys,For example asymmetric encryption algorithms are used in fingerprint detection, home security IoT and so on. We provided brief overview of two asymmetric methods such as Rivest–Shamir–Adleman(RSA) and Elliptic curve cryptography(ECC). This method is totally different from symmetric encryption as discussed in section 4.1 and 4.2

4.1 Rivest–Shamir–Adleman

Rivest–Shamir–Adleman(RSA) encryption algorithm is the basis for modern asymmetric encryption, which uses a pair of keys (public and private key) to encrypt information and prove the sender's identity [17]. RSA protects the sensitive data by applying complex mathematical operation such as factorization method. The message workflow of RSA encryption and decryption is shown in Figure 7.In RSA workflow, the sender encrypts the message using the public key and it will generate the digital signature to protect the message in the IoT device. Whereas, the receiver will be able to read the message by using both private and public key.

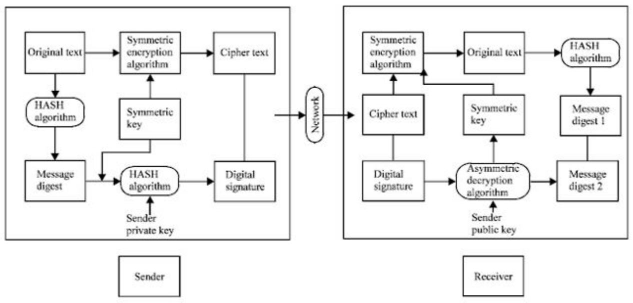


Fig. 7. Illustrates the encryption and decryption process of RSA algorithm in the Digital signature uses the public key to encrypt and the private key to decrypt in the encryption and decryption process [12]. The sender using HASH algorithm to calculate the hash value of the file M, then generate the digital signature C from using the key to encrypt digital abstract and then M C together and sent to the receiver meantime receiver receives the file M1 and digital signature C1, needs to verify that M and M1 are identical [12]

In [20] the authors implemented the RSA algorithm which requires the short key length to secure the information from wireless IoT device as shown in Figure 8.Their study states that:

1. Firstly, two 16 bit prime numbers p and q are used to generate 32 bit public and private keys [20].

2. When both a public and private key are generated, the public key (e, n) is distributed to the device requiring encryption and the plain text is encrypted and encrypted cipher text is sent where data are required and decrypted via private key (d) [20].

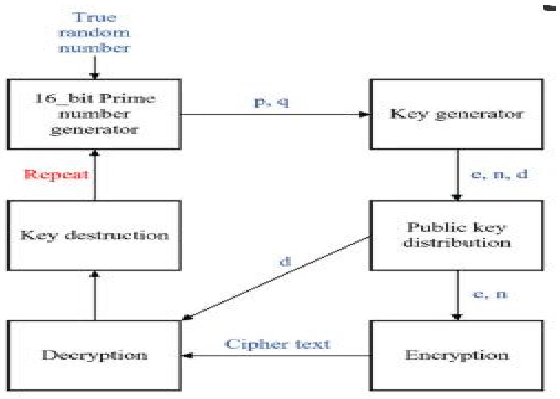


Fig. 8. RSA encryption by generating the random number with small key bits mechanism [20]

The keys will not store any information in the memory, so once the decryption process is completed the keys will be automatically deleted. Then the keys with a different set of a random number will be generated for new encryption. The major drawback is the implementation cost is higher for regenerating the smaller keys for encrypting the messages.

An advantage of RSA encryption is that it generates some random numbers to set as a private key. In addition to that, the random number key is a fixed private key for a single user. For every encryption of the message, the receiver receives a new private randomly generated key to access the message from the sender so that hackers or attackers are will not be able to find the private key to read the secret message.

Moreover, the drawback of RSA algorithm is that, it requires a key of 2048 bits or more to guarantee security and the encryption algorithm using such a large key size is not suitable for use in wireless communication devices,cell phones, IoT devices, or places that require fast data processing [20].

4.2 Elliptic curve cryptography

Elliptic Curve Cryptography is a more advanced method used for encrypting the information. This method encrypts large document which contains more than 400 words in a document within a few seconds.The key distribution algorithm is used to share a secret-key to the user, the encryption algorithm enables confidential communication and the digital signature algorithm is used to authenticate the signer and validate the integrity of the message [14] [5].

4.2.1 Speed-up the encrypt and decrypt process

Laiphrakpam Dolendro Singh and Khumanthem Manglem Singh (2015) implemented the high-speed text cryptography using Elliptic curve cryptography encryption algorithm. The algorithm is designed in such a way that it can be used to encrypt or decrypt any type of script with defined ASCII values [16]. Their study states that:

1. Over 409 words encrypted in 0.093seconds and decrypt the same message length in 0.14seconds with 21.017kB size.
2. Their method avoids the costly operation of mapping and the need to share the common lookup table between the sender and the receiver [16].

Their results show the speed of encrypting the message with lesser cost and low computational power. Also, they proved 192-bit key length can able to protect against naive attack.

The ECC algorithm outperforms RSA in a constrained environment in terms of memory requirements, energy consumption, key sizes, signature generation time, key generation and execution time, and decryption time while RSA performs better in verifying the signature and encrypting [13].

5 DISCUSSION

Both symmetric and asymmetric encryption methods have their strength and weakness.Also, all encryption methods varies each other such as, *how much execution time is taken for encrypting and decrypting the message in IoT device? how many key sizes need for encryption process? what are all the risk while encrypting?*. In this section, we will compare them on requirements, efficiency and scalability on IoT devices. We also discussed real-time example are implemented in different types of IoT devices using an encryption algorithm as shown in Table 1.

5.1 Requirements

Requirements for each encryption algorithms play a vital role during the implementation process in IoT devices. Requirements such as cost, risk factor and execution time are taken into account.

Furthermore,to compare efficiency between symmetric and asymmetric encryption methods, the length of words used for encryption and decryption of each method and the power consumption(speed) as explained further.

The most beneficial method, the Advanced Encryption Standard algorithm reduces the cost of building the security to protect the message between the edge devices. Moreover,For this algorithm it is hard to implement the protection structure but it provides better security to message after implementation of the AES algorithm in IoT device, no major risk detected in the AES algorithm. Execution time called as a number of rounds where the rounds depending upon the key length. For instances, 192 bits input key length need 10 rounds to encrypt the message.

The power consumption is dependent on the processing speed because of the execution time, so the number of computations that determines the processing speed becomes the index of the lightness [19].AES encryption speed up the process for small information length.

Chacha20-poly1350 is more efficient in encrypting large information in IoT devices, browers(Morella Firefox) and wearable devices. The execution time is input-independent since ChaCha20-Poly1305 does not contain variant time operation such as SS-box [8]. Chacha20-Poly1350 was three times faster than the Advance Encryption Standard(AES) on mobile devices. In addition to that, this algorithm has masking functionality to protect the signals from attackers/hackers.

In contrast to the popularity of the Advanced Encryption Standard(AES) encryption method, Chacha20-Poly1305 plays a prominent role to protect the sensitive data. In additon, Chacha20-Poly1305 encypts the message faster so the power consumption are low comparatively. we observed this difference by analyzing the several research papers. This algorithm works very fast to encrypt and decrypt the large sensitive message as already discussed in section 4.

Rivest, Shamir and Adleman(RSA) requires more enhanced memory requirement, memory usage to encrypt the messages. Meantime, RSA applies a large key size to generate a random number for safeguard the data. So, implementation cost is higher for every addition of key size. RSA is very vulnerable to attacks, if the generated key is weak, therefore care must be taken to ensure that two large random numbers are used to calculate the modulus. [3], [13].The execution time of the RSA asymmetric algorithm for encryption takes low time for small message length and more time for large message length compares to the Elliptic curve cryptography asymmetric algorithm.

Elliptic curve cryptography algorithm is a high standard asymmetric algorithm, which can be able to execute the algorithm effectively in IoT devices. Elliptic curve cryptography requires smaller pairs of a key to encrypt and decrypt the long message faster while comparing to RSA key size. In addition to that, the ECC algorithm needs less amount of time to execute the encryption and decryption process for both a short and large set of information. By applying the complex techniques such as scalar multiplication technique to the ECC algorithm, the security, power and timing attacks can be preventable.

5.2 Scalability

The scalability in the IoT devices play a significant role because there are a diverse number of wireless IoT devices which use different type of encryption methods depending upon the requirements and the efficiency needed for particular IoT system.

AES is a widely used encryption algorithm in IoT devices, due to its security measure and low cost for implementation. Moreover, it requires fewer resources and is also much faster than asymmetric ciphers [7].

Chacha20-Poly1305 is an alternative symmetric encryption algorithm of Advanced Encryption Standard (AES). This algorithm developed with both security and authentication to protect the sensitive information in IoT devices. Also, it is even more faster than Advanced Encryption Standard. Comparatively, in symmetric encryption algorithms, Chacha20-Poly1305 plays a prominent role in IoT devices

In asymmetric encryption algorithm Elliptic curve cryptography is an alternative of Rivest–Shamir–Adleman (RSA) method. Elliptic curve cryptography is the fastest asymmetric encryption which is used to encrypt large set of information in the IoT devices. Moreover, it can protect multiple communications between the devices. Also, the implementation costs are low compare to Rivest–Shamir–Adleman method. IoT device needs small key size with low implementation cost to secure the informations, whereas RSA has large key size and implementation costs are high so it cannot be used in IoT device. The researchers implemented the low key-size generated by RSA and tested in IoT device which gives more advantages but equally it has drawbacks for protecting the data for long period of time.

5.3 Real-Time Examples

The Table 1 shows the encryption algorithms used in the IoT devices and in Table 2 shows the different type encryption algorithms with key size. Encryption algorithm can be implemented depending upon the requirement of key size in the IoT device

Table 1. Real-Time Examples

Encryption Methods	IoT device
AES	Refrigerators and smart phones
Chacha20	Google Chrome, Apple's HomeKit,Mozilla Firefox
RSA	web Browsers
ECC	Smart Homes (IoT)

Table 2. Overall encryption Methods

Encryption Methods	key size
AES	128, 192 or 256 bits
AEDS-GCM	128, 192 or 256 bits
Chacha20	256bits
Chacha20-Poly1305	32 bytes
XChacha20-Poly1305	32 bytes
RSA	1024 or 2048 bits
ECC	256bits

6 CONCLUSION

Various encryption methods exist in the IoT devices with their strengths and weakness. This paper is limited to the comparison of two types in the symmetric method and asymmetric method. In addition, we discussed the requirements, efficiency and scalabilityin IoT device of each method.

Chacha20-Poly1305 is the best method in the symmetric encryption algorithm when we look at the requirements in section 5.1 and Table 2. This algorithm has the best encryption standards with masking functionalities to protect the information in IoT devices. Moreover, this method is widely used in many IoT wearable devices.

In asymmetric methods, Elliptic curve cryptography encryption is the best method because it needs fewer parameters to build the encryption algorithm in the IoT devices. Also, this algorithm encrypts the information faster with fewer memory requirements and key size. Chacha20-Poly1305 use the same key for both encryption and decryption whereas Elliptic curve cryptography method uses the different key(public key and private key) for encrypting and decrypting the message. In addition, Chacha20-Poly1305 has small key size which is the added advantage to the implementation in the IoT devices. The key size of each encryption algorithm as shown in Table 2

In conclusion, we argue that the encryption algorithm should be applied depending upon the requirements of IoT devices. The encryption algorithms we discussed have their advantages and disadvantages. So, the algorithms should be used accordingly. Advanced Encryption Standard process should be used to a smaller dataset flow in the IoT device whereas Chacha20-Poly1305 can be used for complex IoT devices.

Both symmetric and asymmetric encryption algorithms can be used in IoT devices to protect the data. Symmetric encryption types are recommended to use in small IoT devices or less communication between the IoT devices such as Apple home kit, smartphone and surveillance systems. These devices need to implement with fewer requirements to obtain better efficiency to protect the data with an inexpensive cost for personal usage. On the other hand, the execution time needs to be less for encrypt/decrypt, the more information in IoT device or more sensible interaction between the devices. So, for this scenario, asymmetric encryption algorithms are recommended. These findings should provide a clear overview of which encryption algorithm should use in a different type of devices.

We provide a concise and comparison of different well-known encryption methods used in IoT devices. We compared the encryption algorithms based on requirements to build better protection to the data, efficiency to speed up the encrypt/decrypt process and scalability of encryption algorithm in IoT devices. Moreover, we provided real-time examples of each encryptions algorithm and an overview of all existing encryption. In the section of future work, we will suggest a general idea to extend our research work which also increases the quality of comparison between symmetric and asymmetric encryption algorithms in IoT devices.

7 FUTURE WORK

For future work, we would suggest extending our research by analysing, validating the parameters used for estimating the encryption/decryption speed and evaluating the count of words in Advanced Encryption Standard, Chacha20-Poly1305 types of symmetric encryption methods and Rivest–Shamir–Adleman, Elliptic curve cryptography types of asymmetric encryption methods. In addition, each encryption method have their own key generation process. So analyzing each key generation technique,would be a valuable addition to our paper. Moreover, there are serval sub-types for every individual branch of symmetric and asymmetric encryption methods. Addressing these sub-types will uplift the standards of this paper.

ACKNOWLEDGEMENTS

The authors wish to thank Héctor Cadavid, Wouter Hertsenberg, Rishabh Sawhney for reviewing the paper.

REFERENCES

[1] A. M. Abdullah. Advanced encryption standard (aes) algorithm to encrypt and decrypt data. *ResearchGate*, July 2017.

[2] Compose Labs Inc. *The Advanced Encryption Standard (AES) Algorithm*, 2016.

[3] Doctrina.org. *How RSA Works With Examples*, 2012.

[4] EENEWS EUROPE AUTOMOTIVE. *Chacha20/Poly1305 authenticated encryption IP targets IoT*, June 2017.

[5] M. Hellman and J. Reyneri. Fast computation of discrete logarithms in gf(q). 1983.

[6] Java Interview Point. *Java ChaCha20 Poly1305 Encryption and Decryption Example*, April 2019.

[7] JSCAPE LLC. *What AES Encryption Is And How It's Used To Secure File Transfers*, May 2015.

[8] KDDI Research, Inc. *Security Analysis of ChaCha20-Poly1305 AEAD*, 2017.

[9] R. Kumar and A. Ani. Implementation of elliptical curve cryptography. *IJCSI International Journal of Computer Science Issues*, 8(2):1694–0814, July 2011.

[10] L. Li, J. Fang, J. Jiang, L. Gan, W. Zheng, and H. F. a nd Guanwen Yang. Sw-aes: Accelerating aes algorithm on the sunway taihulight. *IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications*, pages 1204–1211, Dec. 2017.

[11] M. F. Mushtaq, S. Jamel, A. H. Disina, Z. A. Pindar, and M. M. D. Nur Shafinaz Ahmad Shakir. A survey on the cryptographic encryption algorithms. *IJACSA International Journal of Advanced Computer Science and Applications*, 8(11):333–343, 2017.

[12] NaQi, W. Wei, J. Zhang, J. Z. Wei Wang, J. Li, P. Shen, X. Yin, X. Xiao, and J. Hu. Analysis and research of the rsa, algorithm. *Information Technology Journal*, 12:1818–824, July 2013.

[13] S. Nisha and M. Fari. Rsa public key cryptography algorithm –a review. *INTERNATIONAL JOURNAL OF SCIENTIFIC TECHNOLOGY RESEARCH*, 6:187–191, July 2017.

[14] K. Rabah. Theory and implementation of elliptic curve cryptography. *Journal of Applied Sciences*, 5(4):604–633, June 2005.

[15] D. A. F. Saraiva, V. R. Q. Leithardt, D. de Paula, A. S. Mendes, G. V. González, and P. Crocker. Comparison of symmetric key algorithms for iot devices. *MDPI*, oct 2019.

[16] L. D. Singh and K. M. Singh. Implementation of text encryption using elliptic curve cryptography. *Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)*, pages 73–82, 2015.

[17] Sophos Ltd. *Researchers discover weakness in IoT digital certificates*, 2019.

[18] W. Stalling. Cryptography and network security. June 2010.

[19] O. Toshihikos. Lightweight cryptography applicable to various iot devices. *NEC Technical Journal*, 2(1), 2017.

[20] H. Yu and Y. Kim. New rsa encryption mechanism using one-time encryption keys and unpredictable bio-signal for wireless communication devices. *Licensee MDPI*, 9, Feb. 2020.

An Overview of Community Detection Techniques in Graph Analysis

Alpheaus Feltham S4216768, Vinayak Prasad S4208110

Abstract—A set of network community detection algorithms that have been proposed in other papers and compare them for efficiency, effectiveness and ease of implementation. As part of our review, we discuss the function of each algorithm, its ease of implementation, and the relative effectiveness of each algorithm in various scenarios. Of the four algorithms discussed, the first two are an optimization algorithm and an agglomerative algorithm both using a measure of modularity to determine the interconnectedness of the network nodes. The second two are both divisive algorithms, the first using a measure of 'betweenness' to determine community structures, and the second using an edge clustering coefficient.

Index Terms—Graph Analysis, Group Detection, Community Detection, Graph Theory.

1 Introduction

Networks are an increasingly common element in modern life and society. There exist various natural systems which may be seen as networks, examples of which include: cellular structures and interactions, ecological networks such as food webs and biological networks that describe protein folding. There are numerous networks present within human social circles, describing structures such as collaboration, citation or relationship networks. Over the last 40-50 years, the rapidly increasing prevalence of the internet and telecommunications has led to a rise in the value of personal and group information. As such, detecting and analyzing community structures in networks is often a very useful source of data for modern research, as well as modern corporations and organizations.

Graph theory is the predominant means by which networks are analyzed, as it allows various algorithms to easily manipulate the networks, assign values where required, and use such values to make measurements or categorize various portions of the graph. Figure 1 shows a basic network containing 3 separate communities, each of which is densely connected, while the different communities themselves are sparsely connected to each other. An algorithm may interact with each of these individual nodes and the edges between them, assign data to the individual elements and analyze the structure of the graph based on adjacent vertices.

In this paper, we review and compare a number of different methods for identifying community detection in networks, primarily through graph theory based algorithms. We analyze the ease of implementation, the complexity, the effectiveness, and the efficiency of each method for different kinds and sizes of graphs. The first two algorithms use a variation of network modularity to isolate and identify communities. The second two rate a network on its "betweenness" a measure of the number of connections within groups compared to the number of connections between them.

This paper focuses on exploring and comparing a few of the algorithms implemented. Existing papers, such as [1][3][4][6] are not efficient as new methods are implemented and updated everyday. Therefore, we have chosen to analyze, compare and discuss 4 graph theory: Modularity-Based Optimization, Modularity-Based Agglomeration, Betweenness-Based Division and Edge Clustering-Based Division.

• *Alpheaus Feltham is a student at Rijksuniversiteit Groningen, E-mail: a.feltham@student.rug.nl.*
• *Vinayak Prasad is a student at Rijksuniversiteit Groningen, E-mail: v.prasad@student.rug.nl.*

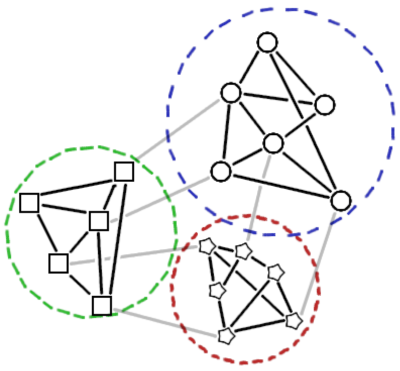


Fig. 1. A network consisting of 3 different, tightly knit communities, loosely connected with each other.

2 Background Information

In this paper, multiple implementations of community detection methods are compared. We are going to first provide an outline of what the challenges are faced when community detection had been implemented. Secondly, we are going to provide an introduction to Graph theory.

2.1 Challenges Encountered

Due to the many industries using different methods to detect communities, many implementations are used for producing valuable insights, each with various purposes and requirements. Many methods have different complexity and data, and interoperability is often a problem. Another challenge is to better define the conditions of applicability of different methods, and theoretical grounds to define when a network needs transformation to become suitable to be analyzed by a given method.

The evaluation of the quality of dynamic communities, both internally and externally, represents a challenge for future works in dynamic community detection. Methods directly adapted from the static case do not consider the specificity of dynamic communities, specifically, the difficulties of smoothness and community events. This question is of utmost importance, since, despite the methods already proposed, their performances on real networks besides those they have designed to figure on remains mostly unknown [2].

As we have seen, various methods exist to generate dynamic graphs with slowly evolving communities. They need different properties, like community events, stable edges, or overlapping communities. Active challenges are still open during this domain, among them The generation of link streams with community structures, and An assessment on the realism of communities generated with such benchmarks, compared with how empirical dynamic communities behave.

2.2 Graph Theory

A graph is an illustrated representation of a collection of objects, where some pairs of objects are connected by links. The interconnected objects provide points termed as vertices, and the links that connect the vertices are called edges.

There are two primary methods for detecting communities in graphs. The Agglomerative method we take an empty graph that consists of nodes with no edges. Then add "stronger" to "weaker"edges one-by-one to the graph. This strength and weight of each edge can be calculated in different ways. In the Divisive method, this occurs in reverse order. In the complete graph take off the edges iteratively. The edge with the greatest weight is removed. Then repeat at every step it recalculates the edge-weight calculation . The weight of the remaining edges change after an edge is removed. After a number of steps, we get clusters of densely connected nodes.

This is used in the Clustering algorithms, which detects communities easily. The graphs provide a t way of coping with abstract concepts like relationships and interactions.

3 Detecting Methods

In this section we present existing solutions for detecting communities using graph theory. The function and implementation of each algorithm is described, and its accuracy and complexity are discussed. Of the algorithms we will be investigating, the first is an optimization algorithm, it finds a value or function by which it can measure the community structure and attempts to maximize it. The second is an agglomerative algorithm, meaning it functions by recursively merging similar nodes or groupings to discover community structures. The final two will be divisive in nature, as they partition and divide the graph into smaller and smaller pieces by removing inter-community links [1].

3.1 Modularity-Based Optimization

The algorithm proposed by Blondel and Guillaume, finds high modularity in large networks in quick succession. As an optimization algorithm, it attempts to use a function to describe a community structure, and then attempts to maximize this function for each grouping of nodes. In this case, that measure is defined by modularity , which helps in identifying the structure of a given graph. Modularity is a scalar value between -1 and 1 that is a measure of the number of links connecting nodes within a community compared to the number of nodes connecting it to other communities. The function of the algorithm consists of two steps, taken iteratively and repeated multiple times, using the output of the previous run as the input for the next. This algorithm generally functions with a weighted network, where edges between nodes are given a weight value determined by some aspect of the network, for example in a phone network, it might be the number of communications made between two users [1].

In the first phase, the algorithm assigns each node in the network to a different community, meaning that in the first phase, there are as many communities as there are nodes. The modularity for each node is calculated, and then the node is compared to each of its direct neighbors. If there is a gain in modularity caused by removing the initial node from its community and by placing it in the community of its neighbours, the initial node is then placed into the community for which this gain is maximum. This is only done if this gain is positive. If there are no positive gains, then the initial node stays in its original community. This process is repeated sequentially for all nodes until no further improvement can be achieved.

In the second phase, a new network is built whose nodes are now the communities found from the previous step. The weights of the links between the new nodes are given new values. These are determined by the sum of the weight of the links between nodes in the corresponding two communities. Once this phase is completed, it is then possible to reapply the first phase of the algorithm to the derived weighted network and to iterate. Since meta-communities decreases with each pass, most of the computing time is used in the first pass and subsequent passes take less time to process. This process repeats until there are no more changes and a maximum of modularity is achieved [1]. A simple example of this process can be seen in figure 2.

The modular process followed by this algorithm is straightforward. As it has to iterate through each node at least once, and then repeats this process for a smaller and smaller network as more and more of the nodes are aggregated into larger communities, the computational complexity is approximately on the order of O(nlog(n)). This means that even on larger networks, this algorithm can function quite efficiently.

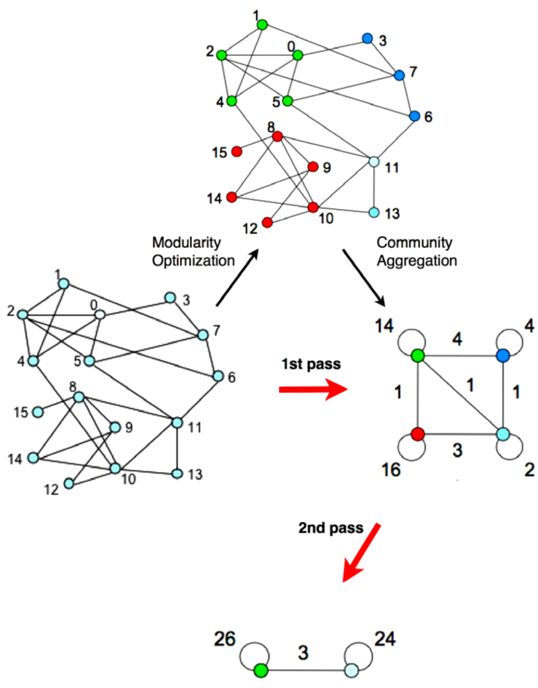


Fig. 2. A simple overview of the Modularity Optimization Algorithm, modified from [1].

3.2 Modularity-Based Agglomeration

The algorithm proposed by Clauset et al. [3] is similar to the algorithm proposed by Blondel et al. [1], which attempts to organize the network into communities using a measure of modularity. While the algorithm proposed by Blondel et al. ends up in simply merging various clusters and to test if there are any changes in modularity, Clauset et al. propose a method which simply each node and tests to work out if its modularity would increase if it is assigned to a specific community.

Essentially, this algorithm attempts to find combinations of adjacent nodes or groups that would increase the modularity measure of the community as a whole. It then repeatedly combines the two nodes or communities whose amalgamation produces the largest increase in modularity. Unlike the algorithm proposed by Blondel et al. however, this algorithm does not require a weighted network to function. This step by step process also allows the algorithm to make a hierarchical dendrogram of the community structure of the network.

The exact implementation proposed in this paper involves storing the modularity of each pair of communities with at least one link between them within a sparse matrix, and storing each row as a binary tree. A max-heap is used to store the largest element of each row of this matrix along with the labels of the corresponding community pair. The algorithm process involves populating the sparse matrix with the initial modularity value of each linked node pair, and finding the largest value to add to the max-heap. The largest value in the max heap is then taken and the two nodes or communities listed are joined into a community. This process is then repeated until only one community remains.

This algorithm is rather simple, its computational complexity is among the one of the top that we've reviewed, being approximately on the order O(mdlog n), with n being the number of nodes in the network, m the number of edges, and d the depth of the resultant dendogram. The complexity of implementation in this case has been reduced through an inspired solution of simply trying to find differences within the modularity of the network, instead of consistently keeping track of every node and adjacency.

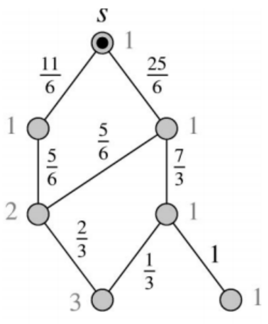


Fig. 3. An example of the results of the shortest path method, taken from [4].

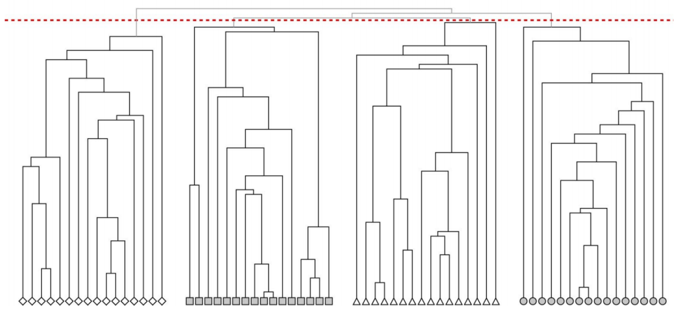


Fig. 4. A view of a hierarchical network dendrogram representation of various communities detected by the betweenness algorithm, taken from [4].

3.3 Betweenness-Based Division

The detection algorithm proposed by Newman and Girvan describes a detection method based on a 'betweenness' value. It uses this measure to determine the interconnectedness of various clusters of nodes in order to discover more densely connected groups. This method has two steps: the first step is to calculate the 'betweenness' of each edge connecting a node and its neighbors, the second is to use this value to slowly disassemble the whole network piece by piece, removing the edges that have the least 'betweenness' in order to determine the community hierarchy of the graph.

The paper proposes a few different options in order to accomplish the first step, determining the 'betweenness'. The first is an implementation of a basic shortest path algorithm that has been adapted to allow for path weighting in cases where there are more than one shortest path between different nodes. Essentially, this functions by selecting a 'source' node to serve as an origin point, and then assigning a weight to each node or vertex in the graph depending on how many shortest paths there are from the source to said vertex. The algorithm then begins calculating the betweenness values for each edge in the graph beginning with the outer extremities of the network, and working its way to the source node, assigning a 'betweenness' value to each edge based on a ratio of the weight of the two vertices it is connecting, and the values of the edges connecting to the vertex that is further away from the source node. An example of this can be seein in figure 3. This whole operation, calculating for a number of source nodes n and edges m functions in time O(mn). Since this then has to be repeated for every edge in the network as it is removed and the betweenness recalculated, the final operational order is generally O(m^2n), though it can become O(n^3) when implemented on sparse graphs [4].

The second and third methods proposed to determine the 'betweenness' values of each node in the paper are fairly similar to one another. Of the two methods described, the first emulates a resistor network in order to determine a 'betweenness' value for each edge, while the other emulates the time taken for an individual to walk between two points using random routes. As mentioned by the authors of the paper, the core principles of both of these methods are based in the same mathematics [4], and they are effectively equivalent regardless of specific implementation. Essentially, these alternate methods attempt to measure the 'betweenness' by simulating a flow rate, either of electrons or of people, between various points within the network. Both implementations use a matrix of values that are inverted to find the flow-rate equivalent between each selection of 'sources' and 'sinks', a process that can be quite intensive, and according to the authors functions approximately on the order of O(n^3) to O(n^4) depending on the graph in question [4].

Finally, once the betweenness measure is found for each edge in the network, the algorithm then sorts through the edges in the network and removes edges with the lowest 'betweenness' scores, gradually increasing the cutoff range as the network begins fragmenting into the most interconnected groups. This allows the network to create a hierarchical dendrogram of the network showing the community structure at different levels of 'betweenness' an example of which can be seen in figure 4 below.

Using the shortest-path 'betweenness' algorithm as suggested by Newman et al. [4] provides us with an algorithm of, at worst order O(n^3), which for small to moderately sized graphs and networks is functional enough, but at the time of publication (2004) this limited the size of the network that could be processed to about 10000 nodes[4]. While the exact number has likely increased with subsequent improvements in computer hardware, a complexity of O(n^3) can still significantly impact the efficiency of any system using this algorithm.

3.4 Edge Clustering-Based Division

The algorithm proposed by Radicchi et al. [6] is very similar to the previously proposed algorithm proposed by Newman and Girvan, and builds off of a similar division method in order to separate the different community structures within the network. The primary difference in this case is the means by which various edges within the graph are selected for removal. Where the Newman et al. used a measure of 'betweenness' to determine which edges to cull, the algorithm proposed by Radicchi et al. calculates an edge clustering coefficient for each link in the network.

The edge clustering coefficient is very similar to a node clustering coefficient, with the only major difference being that it is applied to graph edges rather than vertices. To determine the coefficient for each edge, the algorithm calculates the number of triangles to which the edge belongs, divided by the number of triangles to which it could belong. Specifically, Radicchi et al. describe the clustering coefficient (C) for the edge between vertices i and j ($C_{i,j}$) as

$$C_{\mathrm{i,j}} = \frac{z_{\mathrm{i,j}}}{min[(k_{\mathrm{i}}-1),(k_{\mathrm{j}}-1)]} \tag{1}$$

where $\mathrm{z}_{\mathrm{i,j}}$ is the number of triangles in the network built using the edge, and $\mathrm{min}[(\mathrm{k}_{\mathrm{i}} - 1),(\mathrm{k}_{\mathrm{j}} - 1)]$ is the maximum possible number of triangles that could be built using the edge [6]. Since within clusters of vertices in a graph there will be numerous triangles, especially as the interconnectedness of a community increases, this is a good measure of the structure of a community within the graph.

This measure is then used by the algorithm, in much the same way as the previous algorithm proposed by Newman et al. to divide the network into smaller and smaller community structures and produce a hierarchical layout of the communities within the network. A variant dendrogram chart produced by this algorithm can be seen in figure 5.

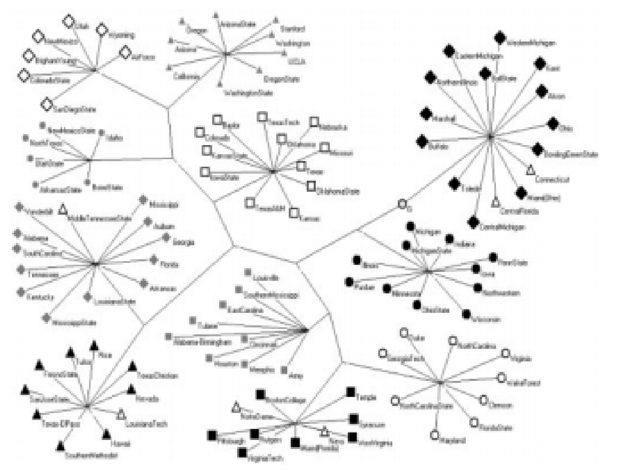


Fig. 5. A view of a network dendrogram representation of various communities detected by the Edge Clustering Detection algorithm, taken from [6].

This algorithm, as it must calculate this coefficient for each edge based only on each other adjacent edge has a computational complexity roughly on the order of $\mathrm{O}(\mathrm{n}^2)$, which is indicative of a fairly significant decrease in complexity and a relatively substantial increase in performance. As such, this algorithm is more efficient, while remaining approximately as effective as the algorithms proposed by Newman et al.

4 Analysis

4.1 Methodology

Each algorithm presented above was compared using 3 different measures; the first, computational complexity, is a measure of how intensive an algorithm is to process, as well as how the size of the input affects the processing time. It is generally measured in *O(n)* notation, with the function O representing the time-scaling factor or approximately how much the algorithm's computation is affected by its input n [5]. The second measure was accuracy, namely how successful was each algorithm in detecting groups given any data set? Did this accuracy change depending on the size or structure of the set? This is obviously an important point to ratify, as the accuracy of an algorithm can heavily influence its usefulness to a user. The final measure, ease of implementation, was a measure of the structural complexity of the algorithm, how difficult it is to actually code and implement on any device. This measure is essentially simply a look at the structure of the algorithm itself and the steps required to complete its calculations.

For the most part, each measure was provided within the papers proposing the algorithms themselves. Most of the papers included an overview of the computational complexity, and all provided an analysis of the accuracy of the proposed algorithms. The complexity of the algorithms that did not have a value provided was roughly calculated from the design of the algorithm itself. The ease of implementation measure was estimated by reading through the description of the algorithm's process provided in the paper, and simply checking to see how many different steps, exceptions and logical comparisons are being made by the algorithm. A simpler implementation would have less of any or all of these, while a complex algorithm may have multiple interconnected steps. This of course means that there is a modicum of bias inherent in determining the last measure, but for this reason, we have provided a general overview of each method in their own sections above so that a reader may determine for themselves if an algorithm is more difficult to implement than another.

For our analysis, each measure listed above was given a different weighting based on its importance to a potential user. The first measure, computational complexity, was provided a moderate weighting. This is because while it could heavily impact the use of an algorithm, unless the computational complexity is so inefficient that even small network inputs would take orders of magnitude of time most algorithms should still be entirely usable even if they are inefficient. The second measure, accuracy, is the most important, and is therefore weighted as such. Evidently, if the algorithm cannot properly detect groups or clustering within the network, it is not a useful algorithm for this task. However, for the most part, we expect that this will also likely be the measure that changes the least between the different methods we have analyzed. Finally, the third measure, ease of implementation, is given the least weight in our analysis. This last measure was not only fairly dependant on the individual implementing it, but we also expected it to act more as a means of providing distinction between algorithms that perform very similarly in the other two categories.

4.2 Discussion

Among the algorithms analyzed by this paper, the divisive algorithms proposed by Radicchi et al.[6] and Newman et al.[4] proved to be the most computationally intensive, generally providing solutions with a computational complexity of $\mathrm{O}(\mathrm{n}^2)$ or $\mathrm{O}(\mathrm{n}^3)$.As a result, these algorithms become untenable for larger networks, as their processes scale exponentially with the number of nodes. The agglomerative and optimization algorithms proposed by Blondel et al.[1] and Clauset et al.[3] fared much better however, with algorithms that operated with a computational complexity of approximately O(nlog(n)).

All algorithms were able to accurately detect various levels of grouping within numerous different kinds of networks. Each method was able to appropriately identify groups and communities and analyze the network structure of both known test networks, and networks built from real data. As such, accuracy was a moot benchmark for this investigation, with each algorithm performing similarly, though this was to be expected. In each case, implementation was generally feasible, though there was more variation among the proposed algorithms, none of the solutions were overly complex, and generally implemented only a few fairly basic steps and generally simple equations. As such, the implementation methods should be taken into consideration on a user-by-user basis, and will mostly hinge on personal preference.

5 Conclusion

The most important factor then, was the computational complexity of each algorithm, as when attempting to analyze modern networks, potentially with hundreds of thousands if not millions of nodes, processing time will be a major hurdle for less-optimized processes. Given this, the computational complexity of each algorithm should be the primary indicator for an optimal algorithm, at least when the intended use is the analysis of networks of massive scales.

As such, these algorithms would operate much more effectively on larger networks, such as the ones that might be of greater interest in the modern era. Given these findings, it is recommended to use algorithms such as those proposed by Clauset et al. or Blondel et al. in order to obtain the most efficient results.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[2] R. Cazabet and G. Rossetti. Challenges in community discovery on temporal networks, 07 2019.

[3] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[4] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[5] C. H. Papadimitriou. *Computational Complexity*, page 260–265. John Wiley and Sons Ltd., GBR, 2003.

[6] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the national academy of sciences*, 101(9):2658–2663, 2004.

High-Level Architecture of Serverless Edge Computing Networks and its Requirements

Mark Soelman, Jaap van der Vis

Abstract—With the continuous rise of Internet of Things (IoT) devices with limited resources, computationally expensive tasks are often performed by the device itself or moved to the cloud. For many of these devices and tasks the latency, network traffic, battery usage and resource utilization need to be minimised. Both edge and serverless computing are considered relatively new fields of research and the combination of these paradigms proves to be promising within the field of IoT. Edge computing is a promising alternative to processing on the same device or in the cloud, in which computational tasks are delegated to geographically near edge devices. Serverless computing allows for dynamic execution of stateless functions without preallocating resources in advance, often through third-party providers.

This paper contributes to these fields by researching various aspects of combining edge and serverless computing, by gathering the requirements of a serverless edge network. Based on these, an architecture consisting out of three main components is proposed. The first component comprises a data-oriented approach of deploying and executing functions, mainly focussing on traditional computational tasks, as well as batch and stream processing. These are requirements for typical IoT use cases. The second component examines the building blocks of an edge computing network, including roles and a orchestration scheduling function. Finally, the third component introduces a lightweight proxy for dynamically determining if a function is best executed in the edge or in the cloud.

Index Terms—Edge, serverless, fog, latency, network, topology, Function-as-a-Service, platform, cloud, IoT, Internet of Things, data-centred

1 Introduction

Since around 2010, the Internet of Things (IoT) has been gaining a lot of interest from academia as well as from the industry. We have arrived at the age in which some of its consequences are already visible in our everyday lives. Major cities use security cameras with face recognition to improve safety of their citizens, while citizens themselves use smart home devices to improve living comforts. Soon self-restocking refrigerators might become reality [16]. Autonomous vehicles are already a reality to some extent, in the future vehicles might be able to warn other vehicles or the relevant authorities for accidents or road damage [12]. Within logistics warehouses, internet-connected devices store and retrieve products [15]

The increasing interest and adoption of IoT posed new challenges. Most IoT devices are constrained by limited resources and are therefore unable to access, process or store information required for the device's core functionality. As a result, a lot of the responsibilities of IoT devices are moved to the cloud. However, as described in [9], some of these devices require low latency for their computations. Others may require low network traffic due to generating large volumes of data and due to bandwidth limitations. IoT devices may also require location awareness to access the closest servers, for example, to warn nearby cars of road damage. For devices with at least one of the aforementioned requirements, using the cloud is insufficient, and requires another approach. Edge computing aims to overcome these limitations of the cloud, by moving computation closer to these devices, i.e. to the edge between the cloud and the local device. Although some prefer to use the term fog computing, we stick to the term edge computing.

The use of edge computing opens up new possibilities for IoT devices but poses new challenges as well. While the resources of edge devices are less limited compared to IoT devices, efficient usage is required to maximise the benefits of using edge nodes. The traditional model of full-fledged virtual machines may offer more flexibility in their usage, but do this at the cost of higher resource usage and long provision delays. The serverless computing model aims to solve this by delegating the resource management of the edge nodes to a third-party provider. This provider is responsible for provisioning and efficient resource usage, which it can accomplish by adopting the Function-as-a-Service (FaaS) model [3]. The FaaS model allows users to perform computations via stateless functions, that the third-party provider runs in its environment. Because the functions are stateless, there is minimal need for resource allocation, thus decreasing the provision delays.

The serverless computing model is more efficient and has lower latency compared to the traditional cloud computing model of using stateful virtual machines as computational resources. Combining edge computing and the serverless computing model allows for low latency and efficient usage of the limited resources. Low latency is often valuable within the field of edge computing [2]. Efficient resource usage is also important in this context, as nodes in the network have very limited resources. Another advantage is that the stateless functions are only restricted by the framework provided by the provider. The FaaS model also allows users to perform computations agnostic of the architectural concerns of the servers, meaning that the user might not even be aware of which edge node executes the computation. However, this means that a serverless computing model requires a more sophisticated architecture.

This paper aims to answer the following research question: *What are the requirements of a serverless platform for edge computing, and how can such network be structured flexibly and dynamically?* The main contributions of this paper are as follows: (1) This work gathers the requirements for a serverless edge computing platform. (2) Next, this work discusses a serverless approach to orchestrate functions and data on available devices in the network in a flexible manner with the focus on data. (3) To facilitate orchestration, this paper elaborates on a scheduling algorithm to determine the optimal machine for execution dynamically. (4) Scenarios are discussed in which using the edge would not be beneficial, while illustrating how a lightweight proxy can determine if a function should be executed in the cloud or locally during runtime.

This paper is structured as follows. Section 2 gives background information on edge computing and serverless computing. Section 3 provides an overview of the requirements of a serverless edge platform. Section 4 discusses a data-oriented approach of serverless function orchestration. Section 5 discusses a scheduling algorithm for selecting the optimal machine for the execution. Section 6 elaborates on choosing between the cloud or the edge. Section 7 reflects on our work and

• *Mark Soelman of the Faculty of Science and Engineering, University of Groningen, m.soelman@student.rug.nl*
• *Jaap van der Vis of the Faculty of Science and Engineering, University of Groningen, j.s.van.der.vis@student.rug.nl*

discusses our findings. Section 8 mentions related work. Section 9 concludes our work and provides suggestions for future work.

2 Background Information

This section first describes the fundamentals of IoT. Then, it provides a definition of edge computing. Finally, the concepts of serverless computing are explained.

2.1 Internet of Things

As described by Atzori et al., the IoT is a paradigm for integrating a variety of objects with the internet to reach a common goal and enhance everyday lives [1]. These internet-connected objects often interact with other objects or with users. An important aspect of this interaction is that these devices aim to achieve autonomy. By integrating objects with both the internet and the environment, they can take over tasks that users would otherwise have to do themselves. While these devices were initially simple, now they are generating increasingly higher volumes of data and more often rely on machine learning algorithms. Because of this trend, the cloud is often used as the main platform for computations, discovering other devices and storing large amounts of data.

As mentioned earlier, this poses problems for a wide range of use cases. An example use case where the increasingly more complicated processing techniques could pose a problem is in autonomous vehicles. An IoT connected vehicle could warn other vehicles about potentially dangerous road conditions, e.g. wet roads that could cause aqua-planing or holes in the road [8]. However, an application like this requires processing power for assessing the situation and for whom this is relevant, something that IoT devices might not have. A solution would be to move the processing to the cloud, but this introduces a communication latency overhead and lacks locality, meaning warnings might arrive too late to prevent accidents. The cloud is a good platform for performing computations that require more resources, but the latency overhead and lack of location awareness are problems that require a different approach.

2.2 Edge Computing

Edge Computing is an approach for moving logic and data away from the cloud due to the aforementioned constraints [9]. Thanks to the proximity and the possible elimination of network bottlenecks, the resulting network contains very low latency and high bandwidth between *prosumers* and their servers. Low latency and high bandwidth give prosumers access to more demanding resources with limited hardware. In this paper, the term prosumer will be used to denote a device with minimal hardware that can either produce or consume data. These prosumers include, but are not limited to, IoT devices.

Although the focus of edge computing is to move away from the cloud, the goal is not to replace it. Both the cloud and the edge can be part of the same computing ecosystem as shown later in Section 6. Instead, the edge acts as an intermediary between the cloud and geographically local prosumers. It is valid for prosumers to use the edge for some purposes while using the cloud for others. Some examples of common edge devices are as follows: a Network Attached Storage (NAS) to provide fast and local access to data and services. A Network Video Recorder (NVR) to record and possibly analyze video streams without heavily using limited bandwidth.

Since the edge is conceptually similar to the cloud, its implications might not be obvious at first. Therefore, we briefly mention its most significant implied advantages and characteristics, as identified by [4, 19].

Proximity and Low Latency Thanks to the presence of edge devices in the vicinity of the prosumers and users, responsibilities such as storing or computing can be offloaded to these edge devices, without having a significant impact on the latency. Furthermore, a prosumer can easily find and communicate with other prosumers in the vicinity, since other nearby prosumers connect to the same edge device.

Location Awareness Applications can find the geographic location of prosumers, by locating the edge device that it is connected to without explicit location logging. During an emergency, instructions or warnings can be sent to on-site edge devices, that forward the messages to all connected prosumers.

Geographic Distribution When the cloud is moved closer to the prosumers, it is implied there is a large number of smaller distributed clusters, instead of only a small amount of large clusters. For instance, this property is useful when searching for a person using face recognition and within a large array of connected cameras, as described in [19]. The algorithm can be run only in the areas of interest on the data stored there, instead of on a single large cluster with much more data.

Network and Cloud Offloading Since IoT is being applied to increasingly more devices, the number of IoT devices is expected to grow to the billions according to [19]. Because of this rapid growth, network bandwidth could become a significant bottleneck.

Heterogeneity The role of the edge is not to eliminate the cloud. Its role is to take over only some of the cloud's functionality. Since the reasons behind edge networks can differ, so can the composition of edge networks differ. Some networks might be large and contain lots of servers, while others might only have a few. Heterogeneity may be present even within the same edge network. Some devices might only serve as storage, while others act as processors, GPU servers, load balancers, et cetera.

2.3 Serverless Computing

The serverless computing model is inherently connected to the FaaS model. In the more traditional model of cloud providers, Infrastructure-as-a-Service (IaaS), the management of virtual machines, operating systems, and load balancing is the responsibility of the end-user. The cloud provider only delivers the infrastructure. Next to IaaS exists Platform-as-a-Service (PaaS), in which the cloud provider abstracts all this away from the user. The user only provides their complete application, which is then hosted by the cloud provider [13]. The FaaS model is similar to the PaaS model, except the provider does not host complete applications, but only provides an environment on which *functions* can be executed [11]. The user only pays for the runtime of the function and does not have to pay for the idle time of a virtual machine.

A *function* consists of a small code module that is typically run on a container (typically docker) with a maximum runtime. The small code size increases the provisioning time, allowing for the rapid scaling of functions and efficient resource usage. The rapid scaling is also a significant benefit for the user, whose application can thus easily handle irregular loads without having to employ idle virtual machines in preparation.

The provider can also improve efficiency by choosing to allow infrastructures to go *cold*, which means to shut down containers that are not recently used, making the resources available again. Lloyd et al. describe four possible states of a FaaS cloud provider with regards to a function:

1. Provider cold: The provider has not come across this function yet and needs to orchestrate it from scratch;
2. VM cold: The provider knows of the function, but wants to deploy it on a new Virtual Machine;
3. Container cold: The function is present on the VM, needs to be deployed on a new container;
4. Warm: The function is located on the container, and can be executed when necessary.

The latter states showed a significantly better performance compared to the earlier [11].

3 Requirements of a Serverless Edge Platform

Baresi et al. identify several requirements of a serverless edge platform, which is the combination of edge computing and serverless computing [3]. In this section, we discuss the proposed requirements.

Low-latency Computations Latency-sensitive or data-intensive computations of client-side devices (e.g. IoT devices, mobile phones) should be moved to the edge in the form of functions. This is similar to the FaaS model in a cloud environment. The advantage of the edge is the lower latency of local edge nodes compared to the cloud [3]. Off-loading computations to external devices makes it possible to work with applications that require powerful computations on (potentially weak) client-side devices. A second advantage is the reduced resource usage on the client-side device.

Collaboration between Server Levels When an edge node does not have available resources for a function, it should either assign functions to other edge nodes or move the function to a higher server level, e.g. the cloud. Edge nodes function as self-organising entities (SEPs) and work together to facilitate the execution of functions. Each node, including the cloud nodes, communicate their capabilities and latency. Based on this information the node which can execute the function the fastest or most efficiently is selected, thus increasing the overall throughput [3].

Coordination via the Edge Location-aware edge nodes should offer a publish-subscribe service, through which nodes (both edge and cloud) can react to events of other nodes. This allows for coordination between different devices, edge nodes, and cloud nodes. A typical example is V2V coordination, where edge-to-edge coordination can be used for vehicles to communicate, while edge-to-cloud coordination can be used by emergency services to react to accidents within a larger geographic area [3].

Latency- & Network Traffic Optimisation To optimise latency an efficient communication protocol like WebSocket should be used instead of HTTP. The HTTP protocol comes with significant overhead compared to more modern communication protocols like WebSocket. The WebSocket also allows back-and-forth communication, since a connection is persistent [3].

A *Workflow-* and *Caching Service* can reduce the network traffic when multiple functions need to be executed sequentially or on the same data. Whenever function composition is needed, the *Workflow Service* can be used to route the intermediate results from one function to the other, rather than sending the intermediate results via the client to the next function [3]. In case multiple functions rely on the same data, the *Caching Service* can supply the data to the functions, preventing the client from having to send the same data multiple times [3].

Network Usage Optimisation Edge nodes can be used to preprocess and filter large volumes of data, the results of which can then be sent to cloud nodes. This preprocessing and filtering prevents large amounts of raw data being sent to the cloud nodes. Only data of interest is sent. [3].

Similarly to the previous requirement, this method attempts to optimise network usage. However, as discussed in the second requirement, some processing requests could be too large to handle for the current network. Therefore, collaboration between server levels is required in order to ensure stable latency. Thus, processing tasks are split between the edge nodes with potential off-loading to the cloud nodes [3].

Stateful Partitions While a big advantage of the serverless model lies in stateless functions, a temporary stateful partition can serve some use cases that cannot be efficiently handled by these stateless functions. Temporary stateful partitions are less restricted in resource usage compared to cloud functions. A *Stateful Compute Service* is responsible for the provisioning of temporary stateful partitions and the routing of requests [3]. A typical use case would be an online game where players can play against other players nearby. All the players connect to a temporary stateful partition on a nearby edge node. The communication requests are routed to the correct container by providing a session token that is unique to this container [3].

4 Data-oriented FaaS Orchestration

In Section 3 the requirements of a serverless edge platform were described. In this section, we discuss the Fog Function, which addresses several of these requirements.

For data-intensive batch and stream processing applications, Cheng et al. propose a data-centric programming model called *Fog Function*, supported by a context-driven orchestration mechanism [6]. In the coming sections, we further elaborate on the Fog Function and context-driven orchestration.

4.1 Fog Function

A Fog Function is a data-oriented function that has relaxed lifetime and resource constraints compared to FaaS functions. Instead of being event-driven, as is usually the case for FaaS, the orchestration of Fog Functions is data-driven [6]. Whenever one or more new data entities arrive to which a Fog Function is subscribed, the Fog Function is started and subscribes to the corresponding input streams.

Data-driven orchestration reduces network usage and latency by making it easier for multiple Fog Functions to use the same data entities. This can help migrate functions between edge nodes or to the cloud. It also makes it possible to save the intermediate or final results as separate data entities, which can be used by other Fog Functions [6]. As described in Section 3, the intermediate results are sent directly to the next function in the chain without passing to the client, thus reducing network usage.

4.2 Context-Driven Orchestration

Context-driven orchestration uses a management node. This management node consists of a Discovery and Orchestrator component and handles data entities and Fog Function orchestration. Each edge node has a Broker and a Worker. The Broker saves data (input, intermediate, and output) as entities, while the Discovery keeps track of all created entities and nodes. The Orchestrator is responsible for assigning actions (combinations of functions and input data) to Workers [6].

The orchestration and scheduling of actions depend on three contexts, namely the *Data context*, *System context*, and the *Usage context*, which are illustrated in figure 1. Contexts are not constant but change over time, which implies several responsibilities for the management node:

- *Usage context*: Keep track of the registered Fog Functions. Whenever a new function is added, the Orchestrator registers it and subscribes to the specified input data in the Discovery.
- *Data context*: The Discovery keeps track of all data entities saved by the Broker. When an entity is added to which the Orchestrator is subscribed, it assigns the corresponding Fog Function to an edge node.
- *System context*: Keep track of the edge nodes and their available resources. When an edge node is changed, added to, or removed from the system, a Fog Function might have to be reassigned to another node or the cloud. This can either be because the node is unable to finish the function due to lack of resources, or because using another edge node can improve latency [6].

The management node is a crucial part of the context-driven orchestration but is also a limiting factor. On a scale of hundreds of edge nodes, the combination of Fog Functions and context-driven orchestration can increase system efficiency by 95% and reduce latency by 30% in comparison to a normal edge function [6]. However, for larger systems, Discovery and Orchestration become a limiting factor and have to be decentralised [6].

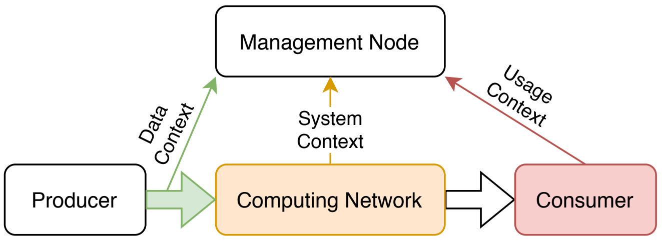


Fig. 1. Context Tracking, redrawn from [6].

5 Scheduling Function Execution

In Section 4, we described a high-level approach for orchestrating functions on the cloud with the focus on data. Now that we discussed *how* functions are executed in a serverless manner, an important question remains: *Where* should these functions be orchestrated on? To be more specific, on what edge device? This question is answered in two parts: Firstly, we elaborate on the network topology. Secondly, we discuss algorithms for choosing the edge device that perform the computations.

Section 2.2 highlighted the heterogeneous nature of the edge. As explained here, some edge devices serve different roles, e.g. computations or storage. However, even devices serving the same role could be fundamentally different. One device might have significantly more resources than the other. Another device could be closer to the prosumer than others with lower latency. Yet another device might be online for a couple of hours during the day to save on energy usage, while others might be online 24/7. Since the edge is heterogeneous and more dynamic than traditional cloud networks, this requires new network topology and selection algorithms, as discussed next. We created Figure 2 to illustrate an example edge network topology.

5.1 Network Topology

Since the edge acts as a dynamic, flexible substitution of the cloud, distributed responsibilities are required. Distributed responsibilities ensure that edge devices are allowed to fail or be turned off at any time. The building blocks of an edge network are described per responsibility. Every node in the network can have any combination of one or more responsibilities. The topology is based on the structures of Cheng et al. [6] and Cicconeti et al. [7].

Prosumer A prosumer or *client* is a device making use of the services provided by the edge network. A prosumer can either produce data for further processing by the cloud or edge or consume data provided from the cloud or edge. Prosumers contact the service discovery to find the router with the lowest latency, i.e. closest to its location. This can be seen in Figure 2. Function orchestrations are submitted to the closest router.

Server Servers in the edge network are defined as heterogeneous devices providing services to the prosumers. Examples of these are: processing server, GPU server, load balancing server, proxy and storage server. Since different servers have different capabilities, they publish their capabilities and notify their availability to the Service Discovery (see Figure 2).

Service Discovery Prosumers, servers, and routers notify the service discovery when they come online. The service discovery will then record their availability in its registry. Multiple service discovery instances may be present in the same network. All service discoveries exchange their registries of online services, so they all share the same information for fault-tolerance. Figure 2 shows this exchange of registries between service discoveries. A secondary task of the service discovery is to forward and update the registries to the routers, so these routers also know who is online within the entire network and the capabilities of these servers.

Router Each router maintains the same service registry, as forwarded by the service discovery. The router performs periodic heartbeats to all services, to keep track of their latency. Separating these latency checks is important, since network partitions may occur and the latency to each service may be different throughout the network. Prosumers submit their function execution requests to the nearest router since this router has the most accurate latency information for the nearest prosumers. This concept is further elaborated by Cicconetti et al. [7].

5.2 Selection Algorithms

Once a function request arrives at a router, it must select a server to forward it to. The ultimate goal is to have the lowest latency on average over all function requests. The following selection algorithms were discussed by Cicconetti et al., which we reiterate: Random-proportional selection, Least-impedance selection, and Round-robin selection [7].

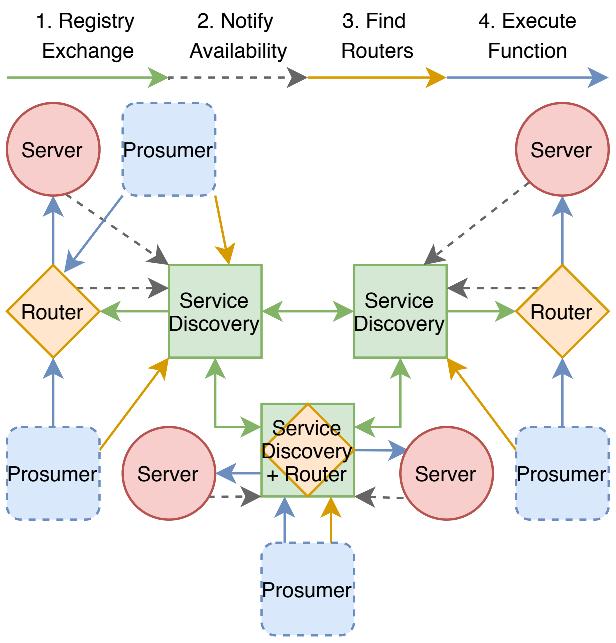


Fig. 2. Edge Network Topology

These selection algorithms are based on weights per server: the cost to execute the function in terms of time.

Smoothed Average Weight Updating A common and effective way of updating the weight is by applying the smoothed average. Formula 1 describes the new weight w of server s at t_1, based on the old weight and the new latency $\delta_d(t_1)$. Here, α is a constant smoothing factor.

$$w_d(t_1) = \begin{cases} \infty & \text{if unreachable} \\ \alpha w_s(t_0) + (1-\alpha)\delta_s(t_1) & \text{if reachable} \end{cases} \quad (1)$$

Random-Proportional Selection Each server is given the following chance of being selected: $1/w_s$. A random server is picked for the function execution, although servers with the lowest weight (average latency) have the highest chances of being selected. Because of the random factor, this algorithm also balances the load.

Least-Impedance Selection The router forwards the request to the server with the lowest weight. This is a simple and greedy algorithm. The main disadvantage of this approach is that multiple near-simultaneous requests are all forwarded to the same server, causing load spikes and possibly a significantly increased latency because of these spikes. For simple set-ups this might suffice, otherwise an algorithm with load balancing should be used.

Round-Robin Selection Based on the registry of services and weights, the router creates and maintains a list of appropriate destinations and a count c of how often they have been selected:

$$A = \{(s,c) | w_s \leq 2minw_s, c \in \mathbb{N}_0\}.$$

Every request, the server with the lowest count is selected, and the count is increased. If the weight becomes too large, it is removed from the list, and re-admitted with count 0 once the latency is back within the given range. This algorithm balances the load over only a subset of the servers.

6 Choosing between Cloud versus Edge

While introducing the concepts of edge computing in Section 2.2, a strong emphasis was put on the edge not being a substitute of the cloud, but rather a complement. For some cases, only relying on the edge is inadequate. Below, we mention some examples for which this is the case:

- There are no edge devices with the required (hardware/proprietary software) capabilities for the given task.
- There are edge devices capable of processing the given task, but they are offline or experiencing high load.
- A large data set required to process the given task is not stored in the edge, and transferring the data set is too expensive.

One solution could be to specify per function if it should be executed in the edge or cloud. This approach could be sufficient for small applications, but it does not come without disadvantages. First of all, it requires extensive performance testing for each function, to see what environment yields the lowest latency. This should not only be performed once but every single time a function is changed or a new function is written. Secondly, manual hard coding assumes the edge is static, while in fact, devices may be added or removed at any point [6]. While the latency may be lower for executing a function in the edge at some point, it may be higher (or infinite) at a later time, due to the changing resources at the edge. Lastly, the edge might be unreachable from the prosumer. To overcome these limitations, a dynamic decision-making approach is desired.

Since we assume the cloud and the edge are primarily being used for serverless function deployment, switching between the two requires no complicated or time-intensive migration of containers, for instance. The prosumer only needs to change the destination of where the function execution request is sent to. As no changes are required by the servers, our serverless edge architecture can be easily extended to support dynamic function forwarding. In the following, we discuss adding a proxy for forwarding requests and an algorithm to select the edge or cloud.

6.1 Edge or Cloud forwarding using a Proxy

A paper by Pinto et al. discussed the usage of a proxy between a client and multiple servers [18]. This proxy evaluates past executions of the same request type per environment. Based on the latency between requesting the function execution and the time taken to return the results, the proxy decides where the next function should be executed.

We similarly extend our architecture by applying this proxy pattern. Instead of deciding what *server* should execute the function, the proxy decides on the *environment*. In our architecture, this proxy is located in between the prosumer and both the edge and cloud environment. All function requests go through the proxy. Moreover, function forwarding should be its sole purpose. The proxy should be lightweight and does not act as a load balancer.

6.2 Forwarding Algorithm

As described by Pinto et al, there is an important trade-off between exploration and optimisation to consider for our forwarding algorithm [18]. Within the field of probability theory, this is a well-known problem since the 1930s and is referred to as the *Multi-Armed Bandit*. This trade-off is briefly discussed in the next paragraph. For further reading and more solutions to this problem, the reader is referred to [5, 20].

The Multi-Armed Bandit problem describes the trade-off between exploitation and exploration. Exploitation refers to picking the best out of all options, based on the information at hand, to minimise regret. In contract, exploration refers to making a sub-optimal decision to obtain more information, while hoping this unexplored option might be the (new) best. Within the context of our forwarding algorithm, exploitation means forwarding a request to the environment with the lowest average latency based on previous measurements. Exploration means forwarding a request to the alternative environment, of which we think will yield a higher latency. Exploration is important for our forwarding algorithm, since the environments, especially the edge, will constantly change in terms of computational capabilities.

$$a_t = P_{t-1}(a) + \sqrt{\frac{2log(t-1)}{N_{t-1}(a)}} \tag{2}$$

Pinto et al. compared multiple forwarding algorithms by running various experiments [18]. The UCB1 algorithm turned out to have the best results for choosing between execution on the edge and cloud. The UCB1 algorithm, which is a variation of Hoeffding's Inequality, is shown in Equation 2. a_t denotes the score of choosing action a at time t, which is incremented for each function execution request. Each action a represents an environment, i.e. the edge or cloud. The first term is a score for exploitation. $P_{t-1}(a)$ is the probability that action a will result in the best latency, based on the previous latency's for all options so far. The second term is a score for exploration. $N_{t-1}(a)$ is the number of times action a has been picked, thus the second term ensures that as time progresses, a sub-optimal environment is explored. While some requests might use a sub-optimal server, the overall performance is stable and changes in the environment are acted on accordingly by the proxy.

7 Discussion

The performance of the suggested solutions and prototypes referenced in this paper are not always properly compared to existing solutions in the industry. To make a correct comparison, we need to establish a performance baseline that represents reality correctly. Furthermore, all comparisons to this baseline should use the same dataset, to ensure consistency of the results.

This work discussed the usage of Fog Functions in Section 4. Cheng et al. developed a software framework FogFlow[1] with adapted settings to orchestrate Fog Functions. Cheng et al. analyse the performance of this framework in an edge environment based on various aspects, such as startup latency, migration latency and throughput [6]. While this is a representative analysis of the Fog Function presented in this paper, it lacks a baseline of existing serverless orchestration frameworks such as those used for the cloud. Therefore, to the best of our knowledge, a sound comparison of the performance of Fog Functions in the edge with serverless frameworks of the cloud is still nonexistent.

Pinto et al. performed a quantitative analysis on having a proxy to switch execution between a local network and a cloud network [18]. Since we integrated a similar component in our own architecture in Section 6, this analysis provides an indication of the performance of our proxy as well. However, Pinto et al. unfortunately do not establish a proper baseline in which all requests are executed on the cloud. Therefore, the average speed-up of including this proxy, as opposed to an architecture without it, is undetermined.

We can extend the proxy component discussed in Section 6 further to allow for horizontal communication between serverless edge platforms, which was discussed by Baresi et al. [3]. While horizontal communication can speed up requests, the same is true for forwarding them to the cloud. In both contexts, latency increases due to the communication with external networks. Furthermore, there is no guarantee that another (horizontal) serverless edge platform can process a request, if the local platform cannot process it. A benefit of this approach is that horizontal communication can reduce the network load of the cloud servers significantly by communicating only with geographically nearby platforms. However, we chose not to include this in our architecture, since it would only be applicable in a very limited number of use cases, and it adds a significant amount of complexity to the overall network structure. Baresi et al. also included a prototype to test the performance based on various metrics, but also did not include a proper baseline.

When the performance of a prototype was tested, this was done in an experimental environment instead of in a (controlled) real-world environment. Cheng et al. evaluated their prototype of the Fog Function-based system using experiments performed on virtual machines from Google Cloud [6]. Baresi and Mendonça and Cicconetti et al. did their performance evaluation on in-house machines [3, 7].

The validity of the results could have been improved by performing more experiments in a real-world environment. Controlled testing environments are important for making usable comparisons between different solutions. Typically it is much easier to create such a controlled environment in self-owned architectures like in-house servers, but some of the typical conditions of an edge network (e.g. locality of nodes) are difficult to simulate. Real-world experiments (possibly

[1]https://fogflow.readthedocs.io/en/latest/index.html

controlled) could give more information on the performance of the prototypes.

8 Related Work

Serverless computing and edge computing are both immature fields of research, especially when both paradigms are being combined. Nevertheless, frameworks for these paradigms are already being developed. Cloud providers offer their serverless infrastructure as a service, commonly referred to as FaaS. Google offers its *Cloud Functions* service[2], Azure has *Azure Functions*[3], and Amazon offers *AWS Lambda*[4]. Functions need to be written and deployed in a certain way, resulting in vendor lock-in. More importantly, these serverless platforms can only be used in the cloud, making them unsuitable for the edge.

Other well-maintained open-source frameworks are Apache OpenWhisk[5], Bitnami's Kubeless[6], OpenFaaS[7] and Knative[8]. The performance of all of these frameworks was extensively analysed for an edge environment by Palade et al. and concluded Kubeless outperforms all other frameworks, while Apache Openwhisk had the worst performance [17]. Mohanty et al. and Kritikos et al. both evaluated and compared the features of open source serverless frameworks, but did not take edge computing into account [10, 14]. Taherizadeh et al. performed an extensive analysis of the requirements of monitoring tools for devices in the edge, as well as evaluating the features of existing tools [21].

9 Conclusion and Future Work

Serverless edge computing combines relatively recent paradigms to create a computational network with characteristics that are interesting for many fields, especially the Internet of Things. Based on the 6 requirements we gathered, we researched state of the art literature to find the best practices for both paradigms. This included a fog function to realise context-driven serverless function and data orchestration, by keeping track of the usage context, data context, and the system context. Then, four main roles of the participants in an edge network were identified as prosumers, servers, service discoveries and routers. Out of the three selection algorithms, round-robin yielded the best results for minimising latency by maintaining a queue of active workers. Instead of choosing between edge or cloud, a proxy can be added to dynamically forward requests. This proxy can use the UCB1 algorithm to exploit the best decision while exploring the sub-optimal decision once in a while to check if they are still sub-optimal.

Future work could be to analyse whether there exist serverless frameworks that satisfy the requirements we identified. Moreover, existing frameworks could be extended by adding the architectural building blocks discussed in this work, such as the fog function, having a role-based network topology, improving selection algorithms or adding integration with a decision proxy. Other future work includes researching multiple levels of vertical communication between platforms, such as having an edge network per building, city, province, or country.

Acknowledgments

The authors of this work wish to thank all reviewers for providing helpful comments and suggestions for improvement. Specifically, for pointing out issues with references and paragraph structure, and for the suggestion of a figure.

References

[1] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010. doi: 10.1016/j.comnet.2010.05.010

[2] L. Baresi, D. Filgueira Mendonça, and M. Garriga. Empowering low-latency applications through a serverless edge computing architecture. In F. De Paoli, S. Schulte, and E. Broch Johnsen, eds., *Service-Oriented and Cloud Computing*, pp. 196–210. Springer International Publishing, Cham, 2017.

[3] L. Baresi and D. Filgueira Mendonça. Towards a serverless platform for edge computing. In *2019 IEEE International Conference on Fog Computing (ICFC)*, pp. 1–10, June 2019. doi: 10.1109/ICFC.2019.00008

[4] F. Bonomi and R. Milito. Fog computing and its role in the internet of things. *Proceedings of the MCC workshop on Mobile Cloud Computing*, 08 2012. doi: 10.1145/2342509.2342513

[5] S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012. doi: 10.1561/2200000024

[6] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada. Fog function: Serverless fog computing for data intensive iot services. In *2019 IEEE International Conference on Services Computing (SCC)*, pp. 28–35, July 2019. doi: 10.1109/SCC.2019.00018

[7] C. Cicconetti, M. Conti, and A. Passarella. An architectural framework for serverless edge computing: Design and emulation tools. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 48–55, Dec 2018. doi: 10.1109/CloudCom2018.2018.00024

[8] Y. Huo, W. Tu, Z. Sheng, and V. C. M. Leung. A survey of in-vehicle communications: Requirements, solutions and opportunities in iot. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 132–137, Dec 2015. doi: 10.1109/WF-IoT.2015.7389040

[9] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[10] K. Kritikos and P. Skrzypek. A review of serverless frameworks. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 161–168, Dec 2018. doi: 10.1109/UCC-Companion.2018.00051

[11] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara. Serverless computing: An investigation of factors influencing microservice performance. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 159–169, April 2018. doi: 10.1109/IC2E.2018.00039

[12] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark. Connected vehicles: Solutions and challenges. *IEEE Internet of Things Journal*, 1(4):289–299, Aug 2014. doi: 10.1109/JIOT.2014.2327587

[13] P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011.

[14] S. K. Mohanty, G. Premsankar, and M. di Francesco. An evaluation of open source serverless computing frameworks. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 115–120, Dec 2018. doi: 10.1109/CloudCom2018.2018.00033

[15] C. K. Nagendra Guptha, M. G. Bhaskar, and V. Meghasree. Design of iot architecture for order picking in a typical warehouse. In *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, pp. 50–53, Dec 2018. doi: 10.1109/CSITSS.2018.8768752

[16] G. S. Nayak, C. Puttamadappa, et al. Intelligent refrigerator with monitoring capability through internet. *Int. J. Comput. Appl*, 2:65–68, 2011.

[17] A. Palade, A. Kazmi, and S. Clarke. An evaluation of open source serverless computing frameworks support at the edge. In *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642-939X, pp. 206–211, July 2019. doi: 10.1109/SERVICES.2019.00057

[18] D. Pinto, J. P. Dias, and H. Sereno Ferreira. Dynamic allocation of serverless functions in iot environments. In *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 1–8, Oct 2018. doi: 10.1109/EUC.2018.00008

[19] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016. doi: 10.1109/JIOT.2016.2579198

[20] A. Slivkins. Introduction to multi-armed bandits, 2019.

[21] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software*, 136:19 – 38, 2018. doi: 10.1016/j.jss.2017.10.033

[2] `https://cloud.google.com/functions`
[3] `https://azure.microsoft.com/nl-nl/services/functions/`
[4] `https://aws.amazon.com/lambda/`
[5] `https://openwhisk.apache.org/`
[6] `https://kubeless.io/`
[7] `https://www.openfaas.com/`
[8] `https://knative.dev/`

Comparing Reference Architectures for IoT

H.F. Stegenga, A. Jakubovskis

Abstract—Over the last few years Internet of Things has been receiving more attention as it has multiple applications across different industries. As the popularity of internet of things increases, so does the demand for a common reference architecture between IoT devices. Multiple IoT reference architectures have been developed. However, the developer community has not agreed on a particular reference architecture, but rather on multiple reference architecture for specific purposes. As the previous research on IoT reference architectures is outdated, this paper focuses on summarizing four reference architectures - IoT ARM, RAMI 4.0, IIRA, Arrowhead - and gives an up-to-date comparison between them. In the comparison, characteristics such as target audience, focus, interoperability and security are analyzed and discussed. It has been found that the four reference architectures differ in their target audience, focus as well as how abstract or detailed the reference architectures are. All of the reference architectures reviewed in the paper have security and interoperability as their concerns, however, the approaches that ensure these qualities have been found to be very different.

Index Terms—Internet of Things, IoT, Architecture, Reference, Comparison

1 Introduction

The Internet of Things is often associated with many technologies which make use of multiple interconnected small electronic micro-controllers for various purposes. The decreasing costs of these devices, sensors and actuators allow for internet connected systems which can support us in many ways. A well known example of these devices, as used today, are smart energy meters. These meters monitor the electricity usage and automatically send the energy usage to the electricity company. They also provide functionality for keeping track of the energy usage per electricity group, so users can detect which parts of their house consumes the most energy [20].

Not only can these devices be useful as energy meters, but they can provide more functionality and services when they are connected with each other and the internet. Smart Homes are a great example of this. In smart homes many small electronic devices are connected together to automate, for example, the lights and the heating facilities. For instance, if connected to the internet, and thus your mobile phone, they can detect when you are on your way home and automatically start warming up the house. By using these technologies the energy costs can be reduced.

Many industries and companies already make use of IoT technologies [23]. For example, these technologies are being used to automate and optimize their production facilities. In these situations the IoT devices are used to monitor and subsequently optimize the production efficiency, often to reduce the amount wasted materials or to guard the quality of the product. Many companies are implementing these technologies, because even a small increase in efficiency can lead to significant money savings. An important aspect of many implementations of these devices is that they are connected to the internet, hence the name 'Internet of Things'.

This paper focuses on exploring and comparing a few of the more well known Internet of Things Reference Architectures. Existing papers, such as [15][7][22], are outdated. They mostly discuss IoT Reference Architectures on a high level and do not provide any details on how, or even if, these architectures define important qualities of IoT networks. We have chosen to analyze, discuss and compare based on four key areas - focus, target audience, interoperability and security. Focus and target audience key areas have been chosen as the audience for reference architectures can vary. Some are aimed at a more general audience than others, while others are more specific to certain industries. In addition, the interoperability and security key areas are chosen as they are of vital importance for IoT reference architectures, no matter the industry which is targeted. The following Reference Architectures will be analyzed, compared and discussed: IoT ARM, IIRA, RAMI 4.0 and Arrowhead.

• *Hindrik S., E-mail: h.f.stegenga@student.rug.nl*
• *Andris J., E-mail: a.jakubovskis@student.rug.nl*

2 Structure

Chapter 1 provides the introduction of the topic and introduces the research issue of the paper. Chapter 3 focuses on background information on the field of IoT and reference architectures in general. In Chapter 4, the four reference architectures are analyzed in detail. Chapter 5 compares the reference architectures on the four key areas. Lastly, in Chapter 6 the results are concluded.

3 Background information

In this chapter background information regarding IoT Reference Architectures will be provided. We will first provide an overview of what the challenges are within the field of the Internet of Things. Secondly, we will provide an introduction to reference architectures in general.

3.1 The challenges in the field

Due to the many companies and industries using IoT, there are many software implementations written for IoT systems, each with various purposes and use cases. This means the ecosystem is highly fragmented. Even though these days many IoT specific standards and protocols exist [18], many software systems use different networking protocols and communication mechanisms, and interoperability is often a problem.

More challenges in the field of the Internet of Things come from the problems related to managing and using many small devices. Most IoT devices are not very powerful, so they are not usually capable of performing complex calculations. Therefore, these calculations have to be done on another, more powerful, system, which means the device has to send the data. Managing a large amount of data coming from many small IoT devices and processing all this data can be a big challenge [21]. Another problem is that sometimes the IoT devices are battery powered, which means that if large amounts of data needs to be transmitted, the batteries are worn down much quicker. Next to these problems, latency can also be an issue in some situations where quick response are required.

Usability is another issue with existing IoT platforms. End-users often need to be able to program the devices to do exactly what they want. Therefore, expertise is often required to understand the IoT platform and how to properly set up these devices. However, the situation is improving for consumers, specifically for smart homes. Many products are available which offer seamless integration with, for example, a platform like *Apple HomeKit* [1]. Such platforms offer easy way to set-up the system where the end-user purchases compatible sensors and actuators, which can be connected to the system.

Another important issue in the Internet of Things industry is related to the security of the devices and related privacy concerns. Securing the devices that control the heating or control the automated garage door is crucial. To show an example, imagine that someone could hack these devices and turn up the heat to dangerous levels while you are sleeping, or could hack a wearable that controls your insulin pump. Therefore, it is of critical importance that these devices are secured, especially when they can be remotely controlled or have internet access. Privacy is another issue, since installing IoT devices in our environment means that a lot of data is collected. This data could be abused for tracking people or advertisement purposes, which may have adverse effects on the individual.

Another major problem is the fragmentation of the technology in general; many hardware and software solutions are available for various problems, but it is often unclear what the best solution might be for a specific problem. A lack of standardization also causes problems, making it harder and more error prone to connect IoT devices together. IoT Reference Architectures try to solve all above mentioned problems for specific target industries.

3.2 Reference Architectures

Reference architectures serve the purpose of providing a reusable and well established guide for developing software architectures. They provide the stakeholders of a software project with a generic and high level abstract view of all the potential components and features involved in their software project. This also includes non-technical guides and descriptions for non-technical members of the software project [16].

Reference architectures also provide a common framework and guide for software architectures in their domain, in our case; the Internet of Things. They primarily consist of a high level description of the most common software components and techniques used in the domain. For IoT these components include topics like communication, data collection and security, for example. Some of the reference architectures might even provide a standard specification on how to implement certain components.

Another important part of reference architectures is that they usually provide non-technical documentation as well. This includes terminology, design decision guides and best practices and general directions on how to fulfill the quality attributes of a project [16]. By using reference architecture as a guide, a software architecture can be created which is easy to understand for anybody in the field. Therefore, by using a reference architecture, this results in an IoT project will have common terminology, components and features, hence promoting an industry standard which is common between different projects.

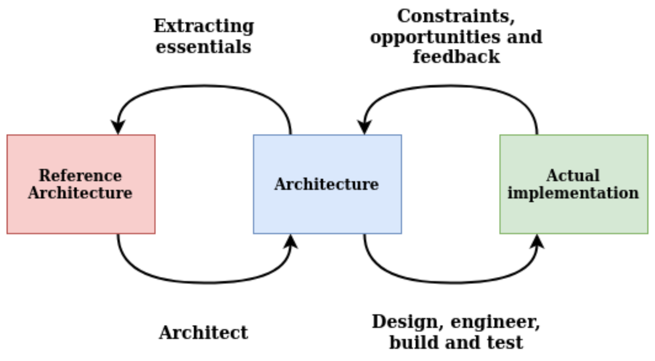


Fig. 1. The relationship between reference architectures, software architectures and the actual implementations. (Adapted from [16])

In Figure 1 the process of generating an architecture from a reference architecture is visualized. More information on how to design a software architecture based on a reference architecture can be found in [6].

Due to their high level abstraction, many reference architectures are aimed at specific target audiences or intended for specific purposes. Therefore, many reference architectures do not try to cover the entire IoT sector and software of it, but rather, a smaller subset of it. They aim to give a high-level standardized way of designing software and IoT infrastructure for their specific target audience and systems.

4 Analysis of Reference Architectures

In this chapter the reference architectures will be analyzed in detail.

4.1 IoT ARM

The first reference architecture to be analysed in this paper is IoT ARM or otherwise known as IoT Architectural Reference Model. IoT ARM has been derived from the IoT-A project. The origins of IoT-A project come from 2009 when new promising applications of IoT started to appear on crowd funding platforms which attracted attention of people [6]. These IoT projects covered wide range of applications in different industries. Therefore, it was challenging to come up with an reference architecture that would be suitable for all domains. Nevertheless, a common reference architecture was needed in order to ensure rapid development of solutions that could be applied across various application domains.

Thus, in 2009 a group of researchers from large industrial companies formed a group to lay the basis for a common reference architecture for the Internet of Things called Internet of Things Architecture project otherwise know as IoT-A [6]. The main aim of this project was to develop an architectural reference model for the interoperability of IoT systems. Other focus points were to outline principles and guidelines for the technical design of its protocols, interfaces, and algorithms [8]. With the introductions of IoT ARM several improvements came along. IoT ARM provided language that could be use among developers and architects. This facilitated the communication, therefore reducing misunderstandings as well as projecting, development time as language was linked to the architecture. Moreover, with the reference architecture people who were new in the field could better guide them selves as there was already a basis provided to develop their IoT solutions. Furthermore, it helped to identify the independent systems blocks which project managers could use to plan the road-map for projects.

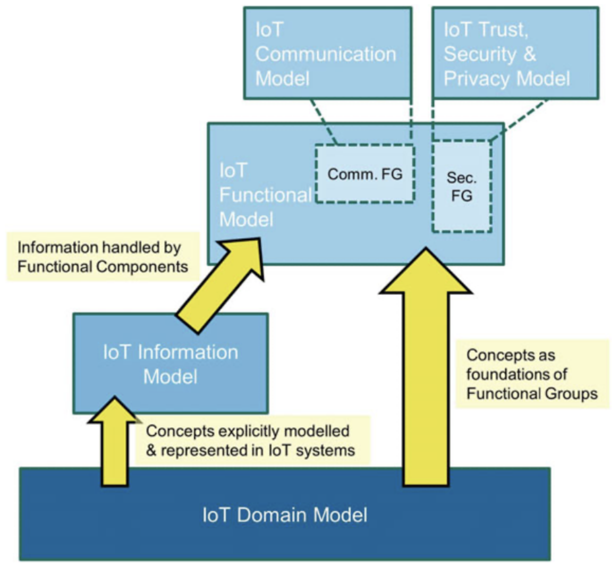

IoT Communication Model
IoT Trust, Security & Privacy Model
IoT Functional Model
Comm. FG
Sec. FG
Information handled by Functional Components
IoT Information Model
Concepts as foundations of Functional Groups
Concepts explicitly modelled & represented in IoT systems
IoT Domain Model

Fig. 2. Interaction of all sub-models within IoT ARM reference model [6]

IoT ARM consists of five sub-models. Figure 2 shows interaction of all sub-models within IoT ARM reference model. The yellow arrows indicate dependency between the modules. IoT Domain Model acts as base model which establishes main concepts in the system such as attributes and dependencies on a conceptual level. It uses high level abstraction in order to be applicable for different industry sectors. IoT Information Model defines structure of information within an IoT system. IoT Functional Model contains groups of functionalities that allow for interaction with attributes and control the information associated with attributes. Functional model also contains two sub-modules which are common for all types of IoT systems - IoT Trust, Security & Privacy Model and IoT Communication Model. IoT Trust, Security & Privacy Model enforces security in the system. Lastly, IoT Communication Model is responsible for ensuring communication in heterogeneous environment.

4.2 RAMI 4.0

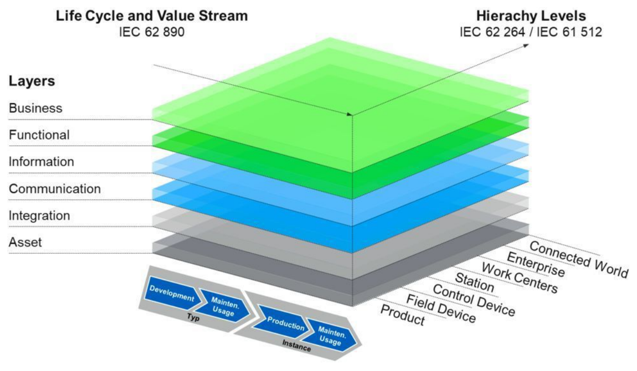


Fig. 3. Three-dimensional model of RAMI 4.0 [9]

RAMI 4.0, or Reference Architecture Model Industrie 4.0, is a IoT Reference Architecture which is aimed at industrial manufacturing companies, primarily in Germany, which want to automate (part of) their production lines. It is centered around the concept of "Industrie 4.0", which refers to the automation and optimization of the whole process of manufacturing products, using IoT devices. The Industrie 4.0 movement makes use of many existing standards and RAMI 4.0 is at the core of it [14]. Unlike most reference architectures, it is driven by integration with business aspects and specification compliance. It is developed and worked on by Zvei.org.

RAMI 4.0 is a service oriented reference architecture which consists of a three-dimensional cube that describes the most important aspects of Industrie 4.0 [19] [24]. A depiction of this can be seen in Figure 3. There are three main axis in this model:

- **The "Hierarchy Levels" axis:**
  The hierarchy levels are defined by those as specified in [11], which is the international standard series for enterprise IT and control systems [9]. This has been extended to include "product" and IoT under the term "Connected World".

- **The "Life Cycle & Value Stream" axis:**
  The bottom horizontal axis describes the life cycle of facilities and products, and is based on [12] and [10]. On top of the IEC standards, it makes a distinction between types and instances. A type can be thought of as a in-development product, and an instance is when the product is completed and actual production has started.

- **The "Layers" axis:**
  The layers on the vertical axis represent the decomposition of a machine into separate layers which represent certain properties of it. The layers map to the different aspects of a given machine, so, for example, the communication layer describes the communication with other devices and people.

RAMI 4.0 aims to integrate different user perspective by providing a common understanding of the Industrie 4.0 technologies. It provides a platform for creating industry wide specifications, standards and their use-cases which are created by committees. Their aim is to develop and integrate these standards in many different manufacturing industries to modernize the technology used by these industries [24].

4.3 IIRA

IIRA is a reference architecture designed for the Industrial Internet of Things. The main purpose of IIRA is to ensure a standard in the industry that promotes interoperability between devices [13]. The IIRA reference architecture document is mainly targeted for architects that are already familiar with architectural concepts, patterns, frameworks and reference architectures. Nevertheless, the document can also be used by project, business and IT managers in order to get insights on how plan the development process of IoT systems.

IIRA is a reference architecture with high abstraction level that does not have a focus on the actual implementation. IIRA has been developed by analyzing and summarizing patterns, features and characteristics from different use cases of the Industrial Internet Consortium, otherwise known as IIC [13]. Each stakeholder in a system has their own concerns and by combining these concerns the whole life cycle of the product is covered. IIRA regards to 4 viewpoints: business, usage, functional and implementation [13].

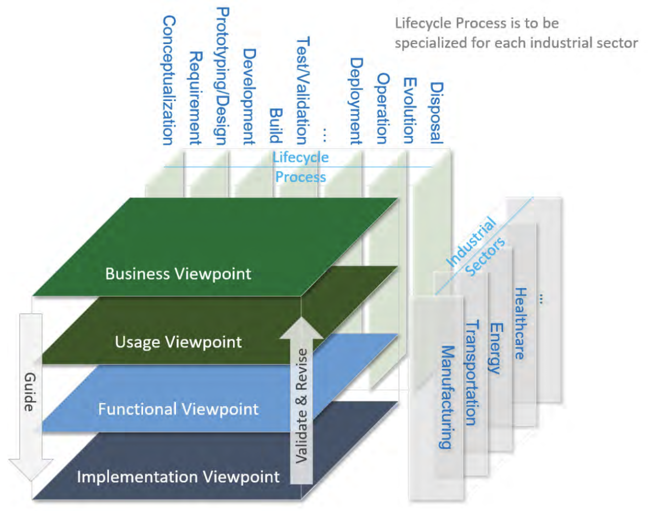


Fig. 4. IIRA viewpoints, system life-cycle process and applications scope [13]

Figure 4 shows the relationship between IIRA viewpoints, application scope in terms of industry sectors and system life-cycle processes which are industry specific. As can be seen in this figure, by going trough the viewpoints starting from business viewpoint in direction towards implementation viewpoint, the decisions get more detailed. The decisions made on business viewpoint are more highly-level abstracted. The decisions on more highly abstracted viewpoints effect the decisions on lowly-abstracted viewpoints.

4.4 Arrowhead

The Arrowhead Framework project is created by the Artemis group, which is an association for embedded systems within Europe [5]. Arrowhead is a reference architecture which specifically focuses on the communication within and between networks of systems. It is centered around the concept of 'local clouds', which are systems of systems [4] as depicted in Figure 5. Their primary goal two-fold: to enable interoperability between IoT components and to allow for easily building systems of systems from individual components [3].

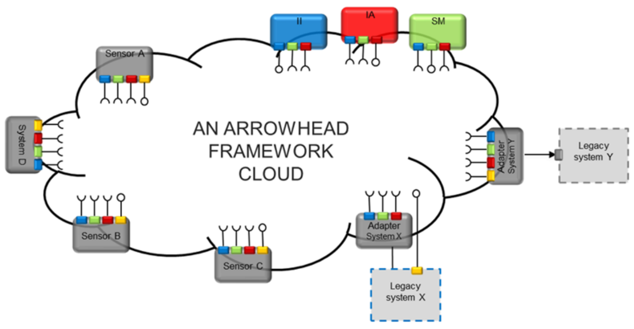


Fig. 5. The local cloud as defined in Arrowhead [4]

As mentioned previously, the concept of a local cloud is fundamental in Arrowhead. The idea is that local cloud is a closed group of industrial devices, which at the very least provide a set of basic networking services, as can be seen in Figure 6. These services include service registration, service discovery, authentication & authorization and orchestration of mentioned devices. Next to these required networking services, devices can expose a multitude of other services. These are called Application Systems/Services and may, for instance, include functionality related to event handling and Quality-Of-Service (QoS) management.

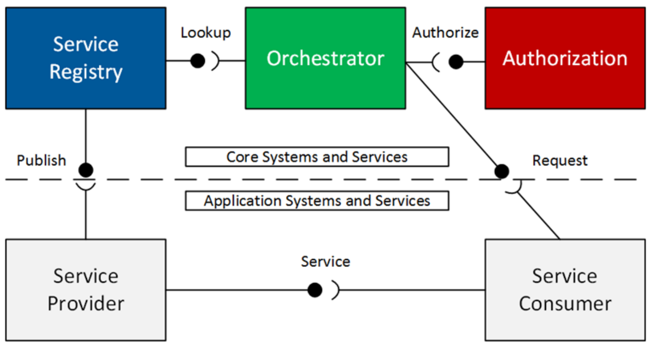


Fig. 6. Application services and systems in Arrowhead [4]

The group behind the Arrowhead reference architecture also provides a set of source code implementations for Java and C++, which can be used to easily create Arrowhead compliant code bases. They also provide a so called cookbook, which essentially is a reference for the Arrowhead reference architecture. There are also a set of libraries available along with documentation, which make it easier to setup a compliant software code base [2].

The project is still under development, and while the core foundations are in place, a lot of implementation work still needs to be finished before industry adaption can really begin. Last year a initiative has been started which focuses on developing tools for allowing third parties to develop and standardize their IoT software using the Arrowhead reference architecture [2].

5 Comparison of Reference Architectures

The reference architectures mentioned in this paper will be compared on four key areas: focus, target audience, interoperability and security. In the 'Focus' and 'Target audience' key areas we will be focusing on what the target audiences and focus of each reference architecture is, and in which way these correspond and differ between the different reference architectures. For the 'Interoperability' key area we will discuss and compare how the reference architectures allow for devices to interoperate with each other and to which extend the reference architectures specify how the communication has to be handled. In the 'Security' key area we will compare the different architecture based on to which extend the reference architectures have specifications for handling security related aspects.

5.1 General focus and target audience

The main purpose of IoT ARM is to introduce a common architectural standard that would promote interoperability between IoT devices, provide principles and guidelines for the technical design, address scalability requirements while at the same time providing examples of real world use cases and benefits of using the reference architecture [6]. The target audience of IoT ARM is general. It includes end users who want to deploy their IoT devices, system architects who are concerned about the design of their IoT systems and IT manages who are concerned with planning the development of IoT systems.

The main purpose of IIRA is to provide guidance and assistance with development, documentating, organizing and deploying IoT systems [13]. The target audience of IIRA is primarily IoT system architects. However, the document can be used by anyone familiar with general architecture concepts i.e. business, IT managers or people who want to better understand to develop IoT systems.

RAMI 4.0 focuses on the concept of Industrie 4.0. It aims to provide a reference architecture for building Industrie 4.0 factories using IoT devices. RAMI 4.0 unifies all the different specifications and standards as defined by the Industrie 4.0 movement, by providing a reference architecture for implementations using IoT Devices [9]. RAMI 4.0 is not only aimed at software architects, but also provides documentation for management and other business related personnel. This is covered implicitly, since RAMI 4.0 defines many parts of itself, i.e. communication and security, using DIN and other specifications.

Arrowhead focuses on the concept of System of Systems, and the interoperability between the systems. The main target audience is companies which want to automate (part of) their production processes [4]. They aim to provide a reference architecture for implementing and connecting multiple IoT devices into a connected system of systems. These systems together provide some kind of functionality. Arrowhead provides documentation and reference implementations of APIs for developers and people interested in using the framework [2].

5.2 Interoperability

IoT ARM does not guarantee interoperability between two architectures. However, it is possible to achieve interoperability between IoT systems with use of IoT ARM. In order to do it, design decisions have to be made that favor interoperability. IoT ARM also allows to achieve interoperability after the system has already be made by building bridges that deliver the main functionalities of another systems or by integrating a subsystem within another system [6]. Moreover, in order to enforce interoperability patterns can be used such as applying design techniques that facilitate change, reserving development environments or building variation points into the software.

RAMI 4.0 specifies interoperability as part of it's specification. This is specified in the form of a service oriented architecture. More specifically, communication between devices is specified in the communication layer of RAMI 4.0. In this layer the exact details of the communication between all devices, which can provide services to other devices, are specified. The communication between a specific device and other devices is handled using a so called Administration Shell. This is a small software layer that allows for inter-operation within the network, and interfaces with the physical device [17]. For RAMI 4.0 this is a bit more complicated than most other reference architectures, since the device can be a machine in a factory. This means that in these cases the Administration Shell acts as a software abstraction of the device, since the machine does not allow for native communication with other devices within the Industry 4.0 based network. The actual communication within the network itself is dependent on the specific implementation, but they are guaranteed to be specified in one of the Industry 4.0 specifications [14].

Arrowhead heavily focuses on interoperability between IoT devices. The whole architecture is centered around Systems of Systems and Services running on it. The architecture provides API standards and specifies exact communication protocols for making guarantees regarding communication between the devices. Next to these communication protocols, it also specifies the concept of Services. Services are pieces of software which run on the devices and provide some kind of service to the System of Systems. There are three core services which are mandatory for interoperability in a System of Systems: Service Registration, Authorization and Orchestration. Together, these three services provide the ability for any of the devices to inter-operate [4]. Next to this, the Arrowhead reference architecture also specifies more complex features for communication. Examples of this are Quality of Service, Broker and Event Handling features [2].

Interoperability is one of the key drivers of IIRA. By creating systems according to IIRA reference architecture, the systems will be interoperable with each other. However, IIRA does not specifically mention how is interoperability within IIRA achieved in their reference architecture specification. This is due to the fact that IIRA in general provides a more high level architecture description and does not provide details about implementation of a system that is aligned with IIRA.

5.3 Security

IoT ARM reference architecture enforces security by IoT Trust, Security & Privacy Model. The security model is made from three layers - Service Security layer, the Communication Security layer and the Application Security layer. These layers contain the following components - Authorization, Identity Management, Trust and Reputation, Authentication, and key exchange and management [6]. Moreover, tactics that promote security can be used such as avoiding over-the-air device management, securing communication infrastructure, physically protecting peripheral devices or considering peripheral devices as available to malicious users in the attacker model.

The Arrowhead reference architecture has two major core services related to providing secure systems. These are the Authorisation and Authentication services and they are mandatory on any Arrowhead implementation. As their names imply, the responsibility of these services is to provide authorisation and authentication services for the rest of the network. This means that a new IoT device can only interact with the network if it has correctly authenticated and is authorised by any of the security-service-providing systems in the network. Arrowhead specifies security features as additionally exposed services on the IoT network. There are special service specifications available for handling this. The most important ones related to security are the Gatekeeper and Gateway v4.0 specifications. Essentially, they are intended to be used for firewall purposes and therefore interactions with other networks, and as such not necessarily part of every Arrowhead implementation. Extensive documentation for these services and everything security for Arrowhead can be found at [2].

RAMI 4.0 tries to provide an entirely different view on security. On the physical level it abstracts secure connections using the aforementioned Administration Shell. This shell provides not only a standardized interface, but also functionality for setting up secure connections with other devices. This means that any interactions with physical devices, like manufacturing machines, goes through the Administration Shell, which increases security [19]. The communication standards themselves are based on existing specifications, as can be found at [14] and [9]. The non-physical part of RAMI 4.0 which influences security is handled by the 'Information' and 'Communication' layers. These layers cover which data and whom may access a given piece data. Together with the business layer, which provides the business logic, security can be implemented using RAMI 4.0.

In order to facilitate security, IIRA uses the so called service network which enables connectivity between services in platform and enterprise tiers of the system. The overlay of private network over the public internet allows for enterprise level security. Moreover, the use of adapters within the architecture not only allows for different type of data model compatibility but also serves as a mean to isolate separate system modules and bridge security domains [13].

	IoT ARM
Audience	General - consists of end users who want to deploy their IoT systems, system architects, IT project managers.
Focus	Introducing a common architectural standard that promotes interoperability, provides principles and guidelines for the technical design, addresses scalability requirements and provides examples of real world use cases.
Interoperability	IoT ARM provides design decisions that promote interoperability. However, it is the system architect that can choose if they want to follow the provided suggestions.
Security	Design decisions and their effect on the system are provided. Therefore, system architect can achieve security by following design decisions that promote it.

Table 1. Summarization of the IoT ARM reference architecture

	IIRA
Audience	IoT systems architects and project managers
Focus	To provide guidance and assistance in the development, documentation, communication and deployment of IIoT systems
Interoperability	Interoperability is a key driver of IIRA, however it does not specifically state how interoperability is achieved. This is due to the fact of IIRA being a high level reference architecture.
Security	IIRA uses a so called service network which enables connectivity between services in platform and enterprise tiers of the system. The overlay of private network over the public internet allows for enterprise level security. Additionally, the adapter pattern can be used to encapsulate data in a subsystem.

Table 2. Summarization of the IIRA reference architecture

	RAMI 4.0
Audience	Manufacturing companies interested in Industry 4.0
Focus	Providing a reference architecture for implementing systems based on the Industry 4.0 standards
Interoperability	RAMI 4.0 is a service oriented architecture where each device has a so-called Administration Shell. This shell provides a software abstraction which abstracts the functionality of the device, and provides services to other connected devices.
Security	Defined on the physical level using the Administration Shell, which provides secure connections to other devices based on existing communication standards. Defined on the non-physical level by specifying the interactions between the business, information and communication layers.

Table 3. Summarization of the RAMI 4.0 reference architecture

	Arrowhead
Audience	Companies looking to automate their production process
Focus	Providing a reference architecture for creating interoperable and connected systems, called System of Systems.
Interoperability	Defined by clearly specified protocols and API standards. Uses a Services model with a service registry, which allows devices to easily be added to an existing connected network and provide services to it.
Security	Arrowhead requires a System of Systems to have a Authorization service available in the network. This service is responsible for accepting or rejecting devices from the network by providing an authentication service, and also provides authorization service for rights management. Finally, external communication is handled by optional Gateway and Gatekeeper services.

Table 4. Summarization of the Arrowhead reference architecture

6 Conclusion

As IoT devices are more commonly used, the need for reference architecture increases. In this paper four IoT reference architectures - IoT ARM, RAMI 4.0, IIRA and Arrowhead - have been analyzed and compared. In order to perform analysis and comparison between these architectures, a literature review was performed on the most recent versions of the reference architecture specifications, as the previous research on this topic is quite outdated. Based on the literature review, four focus areas have been chosen on which the reference architectures have been analyzed and compared - audience, focus, interoperability and security.

The results of the research can be found in Section 5, and tables 1, 2, 3 and 4. contain the summary of the comparison between the architectures. It was found that the four reference architectures differ in their target audience, focus as well as how abstract or detailed the reference architectures are.

IoT ARM is a reference architecture that is aimed towards more general audience and provides design decisions for designing the desired system based on the quality attributes of the system. IIRA is applicable towards different industries and the reference architecture specification is abstracted on higher level compared to other reference architecture specifications. RAMI 4.0 is aimed towards service oriented industry and provides a guide for the digitization of manufacturing. Arrowhead is applicable across multiple industries and provides a more interoperability focused solution with a reference implementation.

As the previous research done on this topic was outdated, this papers adds to existing literature an up to date comparison between four of IoT reference architectures. Our research opens the possibility for more detailed future research to be done on this topic with regards to topics such as interoperability between the reference architectures.

REFERENCES

[1] Apple, Inc. *De Woning app configureren en gebruiken*, 2020. https://support.apple.com/nl-nl/HT204893.

[2] Arrowhead. *Documentation for Arrowhead*. https://forge.soa4d.org/docman/?group_id=58.

[3] Arrowhead. *Why & How*. https://www.arrowhead.eu/arrowheadframework/why-how.

[4] ArrowHead.eu. *Architecture*. https://www.arrowhead.eu/arrowheadframework/this-is-it/architecture/.

[5] Artemis IA. *Artemis Industry Association*. https://artemis-ia.eu/.

[6] A. Bassi, R. van Kranenburg, M. Bauer, S. Lange, M. Fiedler, S. Meissner, and T. Kramp. *Enabling Things to Talk*. SpringerOpen, 2008.

[7] E. Cavalcante, M. P. Alves, T. Batista, F. C. Delicato, and P. F. Pires. *An Analysis of Reference Architectures for the Internet of Things*. 2015.

[8] "Cordis Europe". *Internet of Things Architecture | IoT-A Project | FP7 | CORDIS | European Commission*. https://cordis.europa.eu/project/id/257521.

[9] Deutches Institut Fur Normung E.V. *DIN SPEC 91345*, April 2016.

[10] IEC. *Batch Control*, 1997.

[11] IEC. *IEC 62264 Enterprise-control system integration*, 2013.

[12] IEC. *Life-cycle management for systems and products used in industrial-process measurement, control and automation*, 2013.

[13] "Industrical Internet Consortium". *The Industrial Internet of Things Volume G1: Reference Architecture, Version 1.9, June 19, 2019*, 2019.

[14] Industry 4.0. *Industry 4.0 Standards*. http://i40.semantic-interoperability.org/.

[15] S. Krþo, B. Pokriü, E. Belgrade, and F. Carrez. Designing iot architecture(s). 2014.

[16] G. Muller. *A Reference Architecture Primer*. Gaudí Muller, 2008.

[17] Platform-i40.de. *RAMI 4.0*. https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.pdf?__blob=publicationFile&v=7.

[18] Postscapes. *IoT Standards and Protocols*, 2020. https://www.postscapes.com/internet-of-things-protocols/.

[19] K. Schweichhart. Rami 4.0: An introduction. https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf.

[20] Q. Sun, H. Li, Z. Ma, C. Wang, J. Campillio, Q. Zhang, F. Wallin, and J. Guo. *A Comprehensive Review of Smart Energy Meters in Intelligent Energy Networks*. IEEE, 2016.

[21] L. Wang and R. Ranjan. *Processing Distributed Internet of Things Data in Clouds*. IEEE, 2015.

[22] M. Weyrich and C. Ebert. *Reference Architectures for the Internet of Things*. 2016.

[23] L. Xu, W. He, and S. Li. *Internet of Things in Industries: A Survey*. IEEE, 2014.

[24] zvei.org. *The Reference Architectural Model Industrie 4.0*. https://www.zvei.org/fileadmin/user_upload/Themen/Industrie_4.0/Das_Referenzarchitekturmodell_RAMI_4.0_und_die_Industrie_4.0-Komponente/pdf/ZVEI-Industrie-40-RAMI-40-English.pdf.

User Profiling in Smartphones from Applications

Satyanarayan Nayak, Swastik, and Baskaran, Siddharth

Abstract—In this technological era, where computation is at our fingertips, smartphones are one such device that has revolutionized the human lifestyle rapidly. When encountered with such rapid changes, a question arises as to how aware the end-users are about technology. User profiling is a term commonly used to describe a process of identifying data about a user. User profiling has proved to be extremely useful in enhancing services, but on the downside, user privacy and ethical standpoint are questionable.
In this paper, we first describe the concept of user profiling, data collection techniques and types of data collected. Secondly, we will discuss with statistical facts to understand how the advertisement libraries are exploited by the data brokers to gain access to user sensitive data. Followed by the state of the art machine learning tools adopted by data brokers to re-identify anonymous data back to its original owners to infer user attributes from the anonymous data that would otherwise be considered as garbage. We will then discuss two techniques application containerization and AdSplit that is adopted to safeguard the end-user from user profiling.

Index Terms—user profiling, privacy, smartphones, advertising libraries, NLP, Machine learning.

1 Introduction

Data is the new oil [1], which has become a common refrain in this technological era. Advertisement companies collaborate with data brokers to execute targeted marketing. *Data brokers* are individuals or organisations that collect sensitive user data and sell them as assets to their consumers. In this competitive industry, there is a considerable amount of pressure on the advertising agencies to improve their targeted advertisement to the right set of users. Mobile devices contain rich sensitive information about user attributes which users may not be comfortable in sharing with advertising networks [3]. Apple's app store has around 1.8 million applications and Google's Android market has around 2.1 million applications as of Q1 2019 [3], and the numbers are growing rapidly. In this competitive setting, publishing applications for free and monetizing their applications through advertisements is a common strategy adopted by the companies. The free mobile application market alone is evaluated to be a 50 to 60 billion dollar industry[2] and 90% of iOS App Store apps are free, as are 95% of Google Play Store apps[3]. Key monetization strategies of free applications are in-app purchases, in-app advertisements, email marketing, and collection and selling of data[4].

Smartphones have become an integral part of our lifestyle, and contain rich information about the end-user. This information is stored to provide the user with a rich and personalised user experience based on their usage. This data is also collected by data brokers and are used to categorise a user based on their interests, traits, app usage, and so on. The collected data helps service providers to improve their application and its services. From the viewpoint of advertisement and service providers, they provide users targeted advertisements, personalized recommendations based on user's profiles. From the app developer and designer viewpoint, it is useful to know user's habits, preferences or interests so that apps provide better recommendations and adaptations to improve the user experience [8]. In the case of opportunistic advertising libraries, a privacy loss is possible if the library accesses private user information without the user's consent [3]. Due to the nature of the mobile OS, advertising libraries are given equal privileges as its host application, advertising libraries exploit this to collect the user-data and transmit it externally to the data brokers. The data collected violates the ethical boundaries but cannot be deemed as a violation in the terms and conditions as the advertising libraries and the application are treated as a single entity. The collected data is used for *user profiling*, a context-aware process of analyzing data to know more about the owner of the data. The outcomes of the analysis are stored and sold as assets by the data brokers. Some end-user use applications that promise anonymity from external trackers, but the anonymous data is also not safe from user profiling. Data brokers maintain a database of user data (user profiling), and based on the application footprints of a user, a state of the art machine learning algorithm like KNN, SVM or random forest is used to predict or classify the anonymous user and categorize the data into their respective user-profiles.

We will discuss the contribution of advertisement libraries towards user profiling by studying the statistical analysis done in [3] using *Pluto*, a modular framework for estimating in-app and out-app targeted data exposure for a given app by simulating the data extraction techniques used by advertisement libraries. Followed by the statistical study done in [1], on how anonymous data is retraced back to its owner and used for user profiling using machine learning techniques.

Some techniques can be adopted by mobile devices to safeguard their user's privacy from external trackers. *Application containerization* is a technique that will run each sensitive application in an isolated environment (container) of its own to ensure it stays hidden from other applications. *AdSplit* is another such technique, that proposes to run the advertisement library as a separate process rather than as a part of the host application to ensure that the advertisement libraries do not receive permissions of the host application, in-turn safeguarding the protected folders of the host application.

This paper is organized as follows: We begin with understanding the basic concept of user profiling in Section 2, followed by the main techniques on how data is collected by applications on mobile devices in 3 and the different types of data that is collected and classified by the data brokers in 4. In Section 5 the severity of user data that is exposed through advertisement libraries is discussed with statistical facts and in Section 6, the adoption of machine learning algorithms to trace the anonymous user data back to the original user is discussed. In 7 we see two promising techniques, application containerization and AdSplit that attempt to safeguard the user from user profiling.

2 User Profiling in mobile devices

User profiling from smartphones is a context-aware process of analyzing application data, exploring the correlation of applications

- *Satyanarayan Nayak, Swastik, E-mail: s.nayak.1@student.rug.nl.*
- *Baskaran, Siddharth, E-mail: s.baskaran.1@student.rug.nl.*

Manuscript submitted on 23 March 2020.
For information on obtaining reprints of this article, please send e-mail to:s.nayak.1@student.rug.nl / s.baskaran.1@student.rug.nl.

[1]https://www.forbes.com/sites/forbestechcouncil/2019/11/15/data-is-the-new-oil-and-thats-a-good-thing/
[2]https://rubygarage.org/blog/how-do-free-apps-make-money
[3]https://www.businessofapps.com/data/app-statistics/
[4]https://thinkmobiles.com/blog/how-do-free-apps-make-money/

with the user's personal information, and extracting key features to describe the characteristics and/or behavior of the user [9]. Inferring demographics, personality, interests, preferences, and habits are some of the key focus of user profiling [9]. The individual and/or organization that collect user data and offer them as assets for purchase are called data brokers.

Improving user's mobile experience is the key agenda of user profiling. It helps providers to improve mobile devices, applications, and services. The insights derived from user profiling benefits different actors based on their point of views. An advertisement and personalized service agent provide targeted advertisements, personalized recommendations, and other services for a seamless user experience that is profitable and appreciated by the user. The developer also utilizes the current community trends, user interests and habits to provide their users with a quality user experience. By understanding the user's needs, manufacturers tweak their smartphone design and features to make sound decisions to deliver their users with appropriate features, enabling for a close community-driven growth that has been positively accepted by end-users. The pre-installed applications are targeted based on the interests and needs of a community of users, which leads to quality user experience and reduction of unappreciated bloatware. Information collected for profiling is classified as user-based information and application-based information.

3 Data Collection techniques in smartphones

As Android is very open in design, any app that is installed on a user's phone can easily check for the presence of other apps installed on a phone. Analysis done by `HideMyApp`[2] shows that Android exposes a lot of metadata as it is built to be easy for developers, which in turn allows apps to be easily indexed and fingerprinted. A survey of over 2900 popular apps in the Google Play Store shows that around 57% of these apps explicitly query for the list of installed apps. Android classifies certain apps as potentially harmful which collect information about other apps without users consent. Developers can surpass this by simply providing a privacy policy that describes how the app collects, uses, and shares user data. As per the analysis done by HMA[2], only 7.7% of the apps clearly declared that they collect the list of installed apps in their privacy policies.

3.0.1 *Fingerprintability of Android Apps*

Based on the analysis conducted by HMA[2] an app can easily check if a specific app is already installed on the device. This is done by invoking two methods that Android offers `getInstalledApplications()` and `getInstalledPackages()` (hereafter abbreviated as `getIA()` and `getIP()`, respectively); they return the entire list of installed apps. An app granted with the `READ_EXTERNAL_STORAGE` permission, a very common permission can inspect for unique folders and files in a phone's external storage. An app can also fetch the list of all package names on the phone. This can be done by obtaining the set of UIDs in the `/proc/uid_stat` folder and using the `getNameForUid()` API call to map a UID to a package name. An app can also fetch the list of package names using the command pm list packages and obtain the path to the APK file of a particular app using the command `pm path [package name]`. Moreover, the adb privilege which is used by developers for debugging enables an app to retrieve the APK files of other apps using the command `pull [APK path]` then the app can use other API methods such as `getPackageArchiveInfo()` to extract identifying information from the APK files. The analysis done by HMA[2] shows that Android's open design exposes a significant amount of information that facilitates apps to fetch information by using fingerprinting attacks. App developers themselves cannot obfuscate most of the aforementioned information for the purpose of hiding sensitive apps.

4 Types of data collected for profiling

Data collected by the data brokers are domain-specific, for example, data related to the user's installed application installation list will speak about the category of applications the user is interested in. We can differentiate the data into two categories namely user-based and application-based information.

4.1 User based information for profiling

The data collected that is specific to the user's behavioral footprint like interests, traits. and activities are used to study the user's psychological state. Users with common interests are targeted by the vendors to boost the quality of their services and profit margins.

4.1.1 *Demographic attributes*

The statistical study of the size, distribution, and structure of the population is termed as demography. Age, gender, race, religion, life cycle stage, nationality, occupation, income, education, and marital status are some of the trivial demographic attributes that naturally belong to an individual [9]. Targeted marketing to the right audience drives a successful business. Key benefits of demographic analysis includes[5] ,

- Understanding end-users and what their needs and expectations are from the service provider.
- Develop new products to target a community of ideal users.
- Reduce financial risks and increase sales and profit margin.
- Identify new opportunities in the market.

4.1.2 *Personal interests, traits, and psychological status*

Object, events or process in which users tend to focus a lot are referred to as personal interests [9]. Extraversion, neuroticism, agreeableness, conscientiousness, and openness are referred to as the big fives that understand an individual's personality traits. Informative cues for understanding an individual's psychology can be obtained through smartphones as they play an important role in the day to day activities of the individual. For example, a user who has installed a significant amount of gaming applications would be interested in relevant gaming-focused advertisements. Collection and study of personality traits, interest and psychological status of an individual will then be used to profile and predict the behavior of the individual at a psychological and/or emotional level.

4.1.3 *Lifestyle*

Smartphones track a user's daily interactions and can fine-tune itself to the individual's needs. Fitness trackers and smartphones together monitor the physiological status of an individual some of which include sleep cycle, cardiac rhythm, calories, and daily activities. For example, by analyzing the location and time of the user over a while, smartphones can infer the user's daily activities and can automatically set reminders. At different stages of life, the needs of an individual changes, a study of individuals with similar age groups can reveal some interesting cues. For example, during adulthood a successful individual might be considering buying a new house, targeted news about houses can be recommended accordingly.

4.2 Application based information for profiling

Applications collect a vast amount of data, and some of the data collection policies are questionable. Android applications can access the list of installed apps without any permissions [1]. iOS does not provide a public API that can fetch the list of installed apps, but the app can make frequent scans of the currently running application and over a while construct a list of installed applications [1]. The below subsections will brief the different types of data collected by data brokers.

4.2.1 *Application usage*

Application usage directly translates to the user's behavior, making it one of the prime sources of information in user profiling. App usage report includes the user's interaction with apps, such as when an app is launched or killed, up-time of the application, frequency of application access [9]. App usage cues like duration and frequency is subjective

[5]https://smallbusiness.chron.com/examples-demographics-65678.html

to the user's unique usage pattern, this uniqueness makes it possible to create a unique profile for different users' and infer users' based on their characteristics. Event-driven collection and time sampling schemes are two ways to collect user app information. The event-driven collection will accumulate app usage data when a triggering event occurs, such as application startup, notifications, request to an external network, application shutdown. Time sampling scheme collects app usage data at regular time intervals. For example, the app usage records collected by Lausanne data collection campaign contained start, close, foreground and view event information on Nokia platforms over regular intervals [5][9]. A limitation of this collection scheme is that crucial app usage data might be lost if the sampling interval is set to a small value, an app running in the background, offline applications that do have access to an external network.

4.2.2 *Installed application list*

A list of the installed application provides insights into the user's smartphone usage, the category of applications they are interested in, for example, games, productivity, and management. For example, someone who likes playing games will install games on their smartphone and someone who regularly exercises is likely to install fitness applications.

Profiling based on application lists can be misleading as it is biased by the usage of the applications. Statistical analysis in [6], states that only 10% of the apps were used 80% of the time, and less than 10 applications are used most of the time, suggesting that some applications that are downloaded may not be used frequently. Capturing user's daily application usage along with the installed application list can improve profiling accuracy.

4.2.3 *Application metadata*

App metadata is analyzed together with other types of application-based information. The analysis is focused mainly on the description and category of the application. App description, category, reviews, ratings, and downloads can be classified as app metadata [9]. Based on the category and description of the application collected from the app store, the user's intent of downloading the app is determined. App metadata is generally extracted from the app store, with an exception to the private data-sets that cannot be readily accessed.

4.2.4 *Application installation behavior*

Installing, uninstalling and updating provides insights into the user's interaction with a particular app or a category. Since installing and uninstalling an application is done manually, it provides information on understanding the user's behavior towards a category of applications. Updating an application shows the interaction of a user towards an app, users tend to update their frequently used application but the inference made based on this data can be biased as users tend to switch off their automatic updates and forget to update their applications.

4.2.5 *Application interaction with linguistics user inputs*

Voice assistants have been integrated to elevate the user's mobile experience. Although these advancements are well appreciated, they are accompanied by their downside. Google assistant is one such app where the user can interact with their mobile phone using linguistics. But the application stores user's interactions and transmits this to potential advertising agents that use this information to provide targeted advertisements to the user[6]. Google does provide functionalities to secure privacy, but it has become hard to trust the applications as users are mostly unaware of the extent to which the app interactions are monitored and saved for analysis.

[6]https://www.inc.com/jason-aten/google-is-absolutely-listening-to-your-conversations-it-just-confirms-why-people-dont-trust-big-tech.html

5 User profiling through Advertisement libraries

Research on security and privacy have consistently demonstrated the exploit of advertisement libraries to extract valuable and sensitive data. Smartphones contain rich information about users that promote advertising networks to gather targeted data, sensitive information that a user might not be comfortable in sharing is made available through advertisement libraries [3]. Advertisement libraries on Android have the potential for greater data collection using unprotected APIs to learn other applications information and using protected APIs via permissions inherited from the host application to gain sensitive unauthorized information about the user [3]. We can confidently categorize advertisement libraries as gray areas where the collection of user data is questionable and borderline violates privacy terms and conditions. Risk assessment tools that reveal the leak of sensitive data in an application is hard to upkeep, as they employ static or dynamic analysis of apps and/or libraries. For example, every time an advertisement library is updated, or a new advertisement library is released, risk assessment analysis must be performed again. And some advertisement libraries load code dynamically allowing them to change their logic on the fly [3]. *In-app* and *out-app* are two types of advertisement library attack channels as shown in Figure 1.

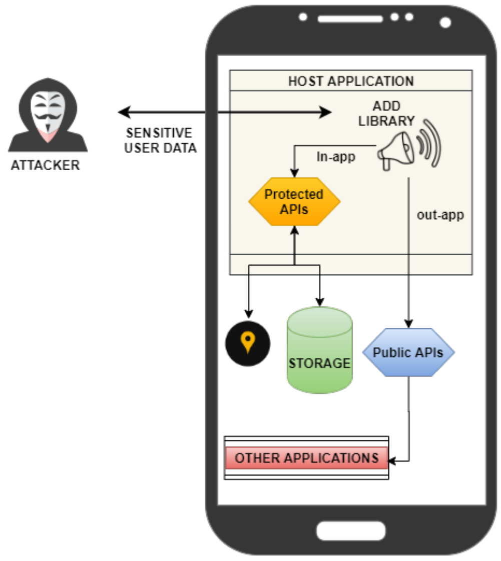


Fig. 1. The illustration shows the two classes of attacks that are performed by advertisement libraries on the mobile platform. In-app is an attack performed over the permission-protected APIs and Out-app is an attack performed over the public APIs.

5.0.1 *Pluto*

A modular framework for estimating in-app and out-app targeted data exposure for a given app. Pluto has the privileges as an advertisement library and it attempts to simulate the actions that an advertisement library would perform in-order to estimate the data exposure from a given app [3]. Pluto is geared to analyze the severity of in-app attacks by utilizing dynamic analysis and natural language processing, and machine learning technique is adopted to gauge the out-app attacks.

5.0.2 *In-app attacks*

In-app attacks are performed through protected APIs present within the host application. Application on an Android will be assigned with a unique UID that is used by the operating system to differentiate between apps during their lifetime on the device. The UIDs during execution are isolated from each other to ensure that one app cannot access another app's resources. The host application is given access to the local storage, GPS, and other sensitive privileges based on the UID that can be accessed via protected APIs. Any external access to these protected APIs is not possible, but the advertisement libraries' embedded within an application will be assigned with the same UID and privileges as the host application both in terms of Linux discretionary access control (DAC) permissions and Android permissions. Advertisement libraries can access the permission-protected APIs on the device and the operating system cannot restrict advertisement libraries privileges as the libraries embedded within an application cannot be treated as a separate entity/process.

DAC allows the advertisement libraries to access the UID-protected local persistent files that the host app generates and that is commonly used to provide personalized services to their user, even when the user is offline. The embedded advertisement libraries transmit this sensitive personalized information about the user to the data brokers. For example, My Ovulation calculator which provides women a platform to track ovulation and plan pregnancy, advertisement libraries can parse local files to learn whether its users suffer from headaches, whether she is currently pregnant, and if so, the current trimester of her pregnancy. User inputs are also tracked by some aggressive advertisement libraries to capture sensitive information like zip code, age, name, and gender, this invasive attack is termed as user input eavesdropping [3].

5.0.3 *In-app analysis*

Dynamic analysis and Natural language processing are adopted to expose vulnerabilities of the app from in-app access. Pluto uses a *monkey tool* to simulates user interactions like a pseudo-random stream of clicks, touches and system-level events, to trigger an event-driven collection of data. The natural language processing is used to find hidden patterns in well structure data that can provide some cues of the presence of user profiling. For example, a software engineer who follows best practices would name their variables used for user profiling as user_profile, uProfile, userProfile, up. Pluto utilizes the Wordnet's English schematic dictionary to derive sets of synonyms for each point to provide better mining precision [3]. Since Pluto emulates the functionalities of the advertisement library we will use them interchangeably in the further sections.

Experimentation: The data analyzed by Pluto is categorized into 2 distinct types, L1 inspection (L1-I) that is used to test the in-app attacks, L2 inspection (L2-I) that is used to test in-app attacks with user input eavesdropping. The goal of the experiment is to first determine the precision and recall of Pluto in determining user attribute (gender) and interest (workout) for both L1 and L2 data sets. Further, inspection is performed with Pluto's MMiners scrapes through the applications manifest to fetch permissions to allow L1 and L2 data sets to gain accessed to unauthorized user data [3].

Findings: From figure 2 we can distinctively see that an aggressive advertisement library with user input eavesdropping improves the accuracy of predicting user attributes and interests significantly. One reason for this observation is, eavesdropping enables the libraries to access layout files that contain information about user attributes and user interests [3].

From Figure 3, the presence of MMiners allows Pluto to predict the address of the user with 90%+ accuracy. By gaining access to the otherwise locked resources like the application bundles [7] that contains readily available personalized user-specific data, the advertisement library is able effortlessly read the user-specific information.

5.0.4 *Out-app attacks*

Out-app attacks are performed through the public APIs to collect a list of installed applications and device information. The Android operating system treats these APIs to be harmless and are left unprotected, this implies that the advertisement libraries can readily access these APIs. For example, the list of installed applications on

[7] App Bundle is a new upload format that includes all your app's compiled code and resources, but defers APK generation and signing to Google Play

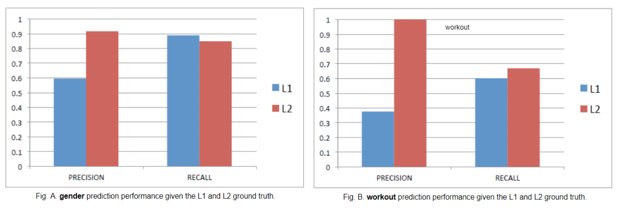


Fig. 2. The result shows the precision and accuracy oh In-app attacks without MMiners to determine the age of the user and if the user has a correlation with the category workout [3].

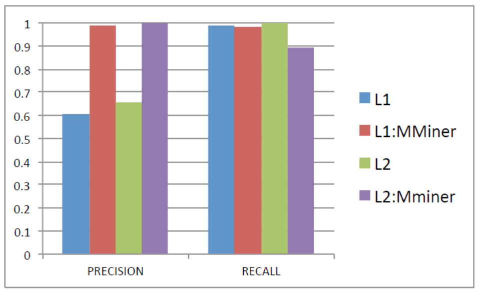


Fig. 3. The results shows the precision and recall of determining the address, and the capabilities of an aggressive ad library that utilizes the MMiners to extract permissions from manifests [3].

the user's mobile is sufficient to determine the user's interest and their smartphone usage. To illustrate the benefits reaped by these harmless public APIs, the twitter app graph program asserted its plans to profile users by collecting app bundles to provide a more personal experience. Guardian newspaper reported that Twitter's revenue in the third quarter of 2014 alone was reported to be $320 million, with 85% of the revenue generated was from mobile advertisements [4].

To understand the severity of public API abuse, in a study performed in [3], 2535 distinct apps were decompiled into smali code[8] for analysis and were looked up for invocation of getIP and getAP (discussed in Section 3) in each app, 27.5% of the apps tested turned positive for the invocation of these public APIs.

5.0.5 *out-app analysis*

Machine learning techniques on a corpus of app bundles are performed to understand the exposure of smartphones to out-app attacks through unprotected getIA and getIP public APIs. Pluto maintains a *co-installation pattern (CIP)* data set, that contains the lists of applications that are installed together, the data set is used by Pluto to simulate the run-time environment that would be available to the advertisement libraries.

Experimentation: Data collected by Pluto through public APIs are analyzed in two stages. First, the co-installation pattern is used to discover the association between two apps, i.e. if app A is present on the mobile device then app B can be found on that device with x% confidence, this technique is commonly referred to as association rule and is commonly adopted in market basket analysis[9]. For example, "Facebook messenger" is an application that can be treated as an extension of "Facebook" indicating the existence of high confidence. If the user has installed only "Facebook" on his mobile device, the vendor can recommend "Facebook messenger" application to the user through targeted advertisements due to the existence of a higher

[8] The smali format is a human-readable representation of the application's bytecode

[9] Market Basket Analysis is one of the key techniques used by large retailers to allows retailers to identify relationships between the items that people buy.

degree of association between the two applications.

Second, supervised machine learning techniques that take CIP estimated app bundles paired with a list of user-targeted data predict whether an app bundle is indicative of a user's attribute [3]. The features of the collected data are not equally important, like a date of birth field from the data has higher prominence in determining the age of the user in regards to other features like marriage date. Also as the number of features increases, it negatively impacts the accuracy of predictions due to the curse of dimensionality [10]. Pluto applies statistical techniques to balance the distribution and dimensionality reduction as a pre-process stage on the collected data before mining for user attributes. Dimensionality reduction is applied on the data set and the features of the data are weighted based on their significance towards predicting the user attributes, for example, if Pluto was configured to classify the individuals with allergy from a data set, assuming a boolean field "has allergy" exists in the captured data set, this feature will ideally receive the maximum weights in comparison to the other features [3]. KNN[11], SVM[12], and Random forest[13] classification algorithms are adopted to predict the user attributes from the CIP estimated app bundles.

Findings: Figure 4 shows the precision and recall of predicting the user attributes age, marital status, and sex over different classifier algorithms namely Random forest, SVM, and KNN. Random forest and KNN perform better after dimensionality reduction, but the SVM performs poorly. One possible reason is that SVM can handle higher dimension data, the model complexity of the SVM is determined by the number of support vectors instead of dimensions, reducing the dimensions crucial support vector data is lost [3].

Classifier	Age		Marital Status		Sex	
	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
Random Forest	88.6	88.6	95.0	93.8	93.8	92.9
SVM	44.8	35.4	66.9	50.5	80.9	70.1
KNN	85.7	83.6	92.5	91.2	91.6	89.9

P = Weighted Precision, R = Weighted Recall

Fig A. Accuracy of predictions after dimensionality reduced

Classifier	Age		Marital Status		Sex	
	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
Random Forest	64.1	66.3	89.8	83.6	91.5	89.6
SVM	65.5	63.6	89.0	82.1	87.4	83.1
KNN	62.7	60.0	86.3	77.7	83.4	74.8

P = Weighted Precision, R = Weighted Recall

Fig A. Accuracy of predictions without dimensionality reduced

Fig. 4. Out-app attack: the table shows the accuracy of different machine learning algorithms that were used to determine the original user for a given anonymous test data [3].

6 User profiling of anonymous user data

Anonymously shared user-data from different applications is analyzed by leveraging machine learning techniques to re-identify the user. Applications installed by the user reveals the user's interests, behaviors, and habits. In a study performed in [1], any 4 applications installed by the user are enough to re-identify the user with an accuracy of 95% and just 2 applications are enough to achive an accuracy of 75%. An advertisement company that has profiled a user for K installed applications, the company can identify the anonymous data that originated from that user. A good machine learning experimentation is possible only in the presence of unbiased and accurate sample data, to ensure the quality of the sample data the unicity of applications and Markov Chain Monte Carlo is adopted.

6.0.1 *Unicity as a measure of re-identification*

Let $\mathbb{A}$ denote the universe of all applications, where each application is represented by a unique identifier in $\mathbb{A}$. A data set D $\subseteq 2^{\mathbb{A}}$ is the ensemble of all apps. $|D|$ denotes the number of individuals in D. D_u represents a non-empty subset of A for all apps of an individual u in D. The set of K-apps over A is denoted as $\mathbb{A}^K$ [1].

Let supp(x, D) denote the support of x $\in A^K$ in data-set D, i.e., the number of records in D which contains x. The unicity or uniqueness of K-apps in D is denoted as, $H_1 = \frac{|\{x:x\in\mathbb{A}^K \wedge supp(x,D)=1\}|}{|\{x:x\in\mathbb{A}^K \wedge supp(x,D)=1\}|}$. It is the relative frequency of K-apps which are contained by only a single record. In generally, a relative abundance distribution (RAD) is a relative frequency of histogram $H = (H_1, H_2, ..., H_n)$ of K-apps concerning a data-set D [1].

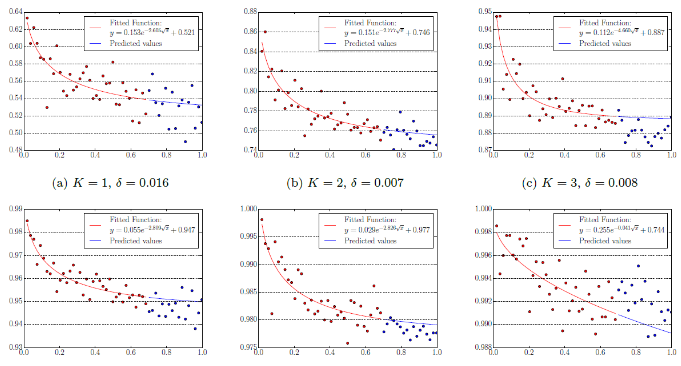
Unicity is the measure of re-identifiability and is used to measure the privacy of the user. It is the probability that an adversary who knows K applications installed on a user's device, can single out the record of the user in their database D. Calculation of unicity is computation extensive and can be approximated with sampling using, $\hat{H}_1 = \frac{|\{x:x\in\mathbb{A}^K \wedge supp(x,D)=1\}|}{|V|}$, where V is the sample set, and $|V|$ is the sample size [1].

6.0.2 *Uniform sampling K apps*

A Sampling of K-apps for experimentation is performed by selecting K-apps uniformly at random in data-set D. However this technique provides a biased estimation of unicity H_1, k-apps that occurs frequently in more records of D become more likely to be selected which makes the unicity measured biased towards popular k-apps [1]. Unbiased sampling is achieved with the use of Markov Chain Monte Carlo method, the proposed method is explained in-depth in [1] Section 3.2.

6.0.3 *Experimentation and outcomes*

The data-set of 37000 users is split into 70% training and 30% testing data. The model used for training is an exponential model describing an exponential decay of unicity, represented as $f(X) = a.exp(-b\sqrt{x}) + c$. Figure 5 shows the different exponential functions used during experimentation and the accuracy of the model to identify or predict the user. A significant gain of 75% is observed when $K = 2$ in comparison to $K = 1$ that has an accuracy of 53%. $K = 4$ yields a satisfactory accuracy of 95%, $K > 4$ does not significantly improve the accuracy of the model.



(a) $K = 1$, $\delta = 0.016$ (b) $K = 2$, $\delta = 0.007$ (c) $K = 3$, $\delta = 0.008$

(d) $K = 4$, $\delta = 0.005$ (e) $K = 5$, $\delta = 0.001$ (f) $K = 6$, $\delta = 0.002$

Fig. 5. Unicity generalization for different values of K, trained all with maximum 37000 users. The learnt models (i.e., f(x)) are present in the legend [1].

7 Techniques to safeguard from user profiling

The section will briefly describe two techniques *application containerization* and *AdSplit* techniques that can be adopted to counter user pro-

[10] $https://en.wikipedia.org/wiki/Curse\_of\_dimensionality$
[11] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[12] https://en.wikipedia.org/wiki/Support-vector_machine
[13] https://en.wikipedia.org/wiki/Random_forest

filing. The techniques are built around the concept of restriction or blocking data collection activities to secure the user's privacy.

7.1 Application containerization

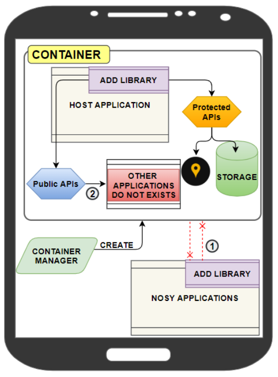


Fig. 6. Application containerization; (1) The nosy application is unable to access the application encapsulated by the container by the means of any public APIs; (2) The public APIs within the container will not expose any other applications, since each container is an isolated environment that only runs one application .

The application containerization technique spins up an on-demand isolated environment aka *container* for an application to run. The APK file of the application is only executed within the container during runtime, each container app has a generic package name and obfuscated app components. As a result, nosy apps[14] cannot fingerprint a sensitive app by using the information about its container app [2]. Figure 6 shows a pictorial representation of the application containerization technique, the container manager is responsible for the orchestration of containers and is the only entity that can be accessed externally. Although this technique promises the privacy of the user from user profiling, performance overhead is inevitable that is introduced during container orchestration [2]. The advertisement libraries that are an integral part of the host application itself can still access the protected APIs of that host applications, which makes this technique secure public APIs better in comparison to the protected APIs.

7.2 AdSplit

AdSplit attempts to restore the privacy of the mobile devices from advertisement libraries by running them as a separate application on the device. This ensures that the host application and the advertisement libraries are assigned with different UID's, eliminating the need for applications to request permissions on behalf of their advertising libraries and thus restricting access of advertisement libraries to the protected resources [7]. This technique requires design changes to the mobile operating system as a provision should be made, as advertisement libraries should be managed as a service provided by the OS itself rather than an externally controlled process. The advertisement libraries have access to public APIs, making this technique secure protected APIs better in comparison to the public APIs.

8 Conclusion

As with the rapid advancement of smartphone technology application usage, installed application list, application metadata, application installation behavior, and application interaction with linguistics user inputs are prominent categories to which the user-data collected by application can be classified into. The sensitive user information conveyed by the application is categorized into demographic attributes, personal interest, personal traits, psychological status, and lifestyle.

Data brokers collect user sensitive data through public and protected APIs. The installed application list is one of the most commonly sought our data that can be obtained by accessing the public APIs. The access protected data contained in the mobile devices are accessed predominantly through the advertisement libraries. The privilege to access these protected resources is gained through the exploitation of the flawed permission assignment protocols adopted in mobile devices.

Machine learning techniques have significantly boosted user profiling, by allowing the data brokers to mine for user attributes that are otherwise not apparent. Anonymous data which was generally deemed garbage, can now be processed through machine learning classifiers to re-identify original owner of the data.

Application containerization and AdSplit are two promising techniques that work towards securing user privacy. The application containerization skews towards securing the public APIs but allows the advertisement libraries to access the protected resource of only the host application, whereas AdSplit skews towards securing the protected APIs while allowing some public API access.

The paper should give the reader a good understanding of user profiling and the underlying techniques that are used to collect and profile the users-data. For an in-depth understanding of the statistical experimentation discussion in this paper, it is recommended that the reader should refer to the referenced research papers.

9 Future Works

Data collection techniques are not limited to the ones depicted in this paper, other techniques should be explored. For example, some unethical application can install backdoor, spyware, and trojan that can give unauthorized access to user sensitive data. Application containerization secures mainly public APIs whereas the AdSplit secures the protected APIs, a mixture of both techniques can be adopted that can provide security from both in-app and out-app attacks.

Acknowledgements

The authors wish to thank Prof. R. Smedinga, Prof. M. Biehl for their valuable guidance, Fadi Hohson for the reviews and the authors aforementioned in the references.

References

[1] J. P. Achara, G. Acs, and C. Castelluccia. On the unicity of smartphone applications. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 27–36, 2015.

[2] E. L. J. S. K. H. J.-P. H. Anh Pham, Italo Dacosta. Hidemyapp: Hiding the presence of sensitive apps on android. *28th USENIX Security Symposium*, pages 711–728, 2019.

[3] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter. Free for all! assessing user data exposure to advertising libraries on android. 2016.

[4] S. Dredge. Twitter scanning users' other apps to help deliver 'tailored content'. *The Guardian, November*.

[5] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. *Proc. ICPS, Berlin*, 68, 2010.

[6] V. Rivron, M. I. Khan, S. Charneau, and I. Chrisment. Exploring smartphone application usage logs with declared sociological information. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 266–273. IEEE, 2016.

[7] S. Shekhar, M. Dietz, and D. S. Wallach. Adsplit: Separating smartphone advertising from applications. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 553–567, 2012.

[8] D. M. Sumitkumar Kanoje, Sheetal Girase. User profiling trends, techniques and applications. *International Journal of Advance Foundation and Research in Computer (IJAFRC)*, 2014.

[9] S. Zhao, S. Li, J. Ramos, Z. Luo, Z. Jiang, A. K. Dey, and G. Pan. User profiling from their use of smartphone applications: A survey. *Pervasive and Mobile Computing*, page 101052, 2019.

[14] Application that try to access information about other applications.

Data Science Pipeline Containerization

Andrea De Lucia, Evi Xhelo

Abstract— In the last decade, data has become the most valuable resource for businesses worldwide because it can be used to establish insights about their customers using data analysis algorithms. Data science is the field concerned with the retrieval, processing, analysis and the interpretation of data, and most importantly, big data. Computer scientists are developing new means to facilitate the process of handling big data. One such approach is by helping data scientists build their own Machine Learning pipelines. Such pipelines require an ever growing set of tools and dependencies, creating what it is often called a "Dependency hell". This can be avoided using *containers*, such as Docker, that offer a lightweight approach to reusing and sharing libraries and environments. Containers can also work in the cloud, which lifts the restrictions imposed by working on a singular machine. However, in this scenario, it is important that the orchestration of containers is managed, which can lead to additional complexity. This paper aims to depict how containerization can be incorporated into data science pipelines, how the orchestration is solved and why this is relevant. We will show three different methods on how to establish pipeline containerization and depict a comparison between the three. In the end we will evaluate the methods based on four criteria to help the reader to choose the most fitted platform for their needs.

Index Terms—Containers, Kubernetes, Cloud computing, ML Workflow, Pipeline, Data Science, Kubeflow, Cloudflow, OpenWhisk

1 Introduction

Today there is an ever-growing quantity of data available as well as a demand to extract insight from it in the shortest time possible. Data scientists are able to identify relevant questions, collect pertinent data from a multitude of data sources, organize the information, translate results into solutions, and communicate the findings [14] in a way which can create value to the end-users[1]. Data scientists create value through data-driven analysis. The process of data-driven analysis is composed of several steps that require data scientists to implement different algorithms which perform the specific operations for each and every step [8], as shown in Figure 1.

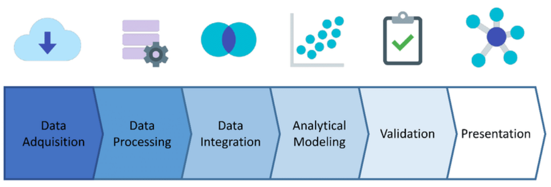


Fig. 1. Steps in the data-driven approach, retrieved from [8]

A collection of these steps is called a pipeline, and it is often used in Artificial Intelligence (AI), with its most common application being an Machine Learning (ML) workflow. The input for an ML workflow is raw data and the output is a machine learning model that can be used for the analysis, argumentation and prediction of data trends.

As of 2012, about 2.5 exabytes of data are created each day, with that number doubling every 40 months [1]. This tendency, also known as big data, has established a need for scalable ML workflows that can handle vast amounts of data and produce intelligible results. However, arranging the scalability of an ML workflow using the resources at hand can create new intricacies for data scientists. Moreover, having to re-run the pipelines multiple times update the hyperparameters of the algorithms is also prone to causing overhead challenges.

• *Andrea De Lucia is with RUG, E-mail: a.de.lucia@student.rug.nl.*
• *Evi Xhelo is with RUG, E-mail: e.xhelo@student.rug.nl.*

[1]Consumers: people, companies and public administrations. Data scientists create added value to serve consumers by finding their pattern and behaviour.

Another issue data scientists face is when the ML workflows have to be transitioned from the development environment, in which they were implemented, to the production environment, where they can start creating value by providing predictions to other software systems [20]. Given how ML workflows are subject to changes in the code, model, and data, it can be hard to predict their behaviour [5].

One of the available means of tackling these issues is by making use of containers and containerizing all the components of ML pipeline. Containerization guarantees that no unexpected behaviour will occur when the ML pipeline is transitioned into the production environment because every step of the pipeline runs in an environment (container) optimized for its purposes. We present a brief list of the added benefits of containerizing an application in Figure 2 [10].

- platform independence
- resource efficiency and density
- effective isolation and resource sharing
- speed: start, create, replicate or destroy containers
- horizontal scaling

Fig. 2. Advantages of using containers for development, retrieved from [10].

For managing running containers, orchestration platforms are used. These platforms have the purpose of eliminating many of the manual processes involved in deploying and scaling containerized applications. However, the tools that we will be discussing in this paper can be used by data scientists to establish their ML workflows without having to define the containers and their orchestration manually. These tools allow data scientists to focus solely on the implementation of machine learning algorithms.

Therefore, in this paper, we will present three different methods that can be used to establish the containerization of pipelines. Firstly, we will introduce some useful background information in Section 2 followed by a presentation on each one of the three tools in Section 3. Finally, we will define some objective criteria in Section 4, concluding with the final evaluation in Section 5 and then in Section

6 our conclusion.

2 Background

As paramount as the development of algorithms is to a data scientist, tuning their parameters, running them on the production environment and being able to showcase the results, is also important. Currently, an increasing number of software engineers (and data scientists) are migrating their work on the cloud, since it ensures more resources. Several intricacies can arise when trying to deploy a finished ML workflow to the cloud, which can prompt new challenges that need to be arranged such as uploading the data, taking care of the access and security, balancing just enough resources for the current job. The platforms that will be discussed in this paper aim to alleviate these challenges by handling the low-level tasks of containerizing the ML applications and orchestrating the containers. We will firstly introduce the concepts that have made them possible, from containers, all the way to their orchestrating tools.

2.1 Containers

A container is a standard unit of software that packages up code and all its dependencies so that the application can run quickly, and reliably, regardless of the computing environment [7]. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in the userspace. Containers take up less space than Virtual Machines (VMs) since container images typically use tens of MBs in size as opposed to VMs which can consume up to several GBs in size. A general comparison between the two different software can be noted in Figure 3.

Containers have become a relevant tool in the world of computing because they ensure compatibility across platforms. It is guaranteed that a containerized application will run in the deployment environments the same way as it would run in the development environment [3]. Containers can be used to establish *microservices* [11]. Essentially, every service is a (containerized) component which can be deployed, scaled, and tested independently from the other services, while also being interconnected to the other components [17]. Docker is one of the most widely-used containerization software in the industry [19].

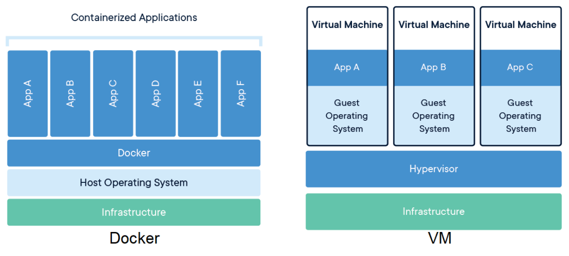


Fig. 3. Comparison between how containerized application (Docker example) and a VM, retrieved from [7].

2.2 Container orchestration

The containerization of applications often results in a *microservice* based architecture where the components are split across many containers. The next step would be to run them in a production environment and to manage their life-cycle by using an orchestration platform [16]. *Kubernetes* is the most popular container orchestration platform and was created by Google to work in any environment of the developer's choice [21]. We will use Kubernets as reference throughout the remainder of the paper.

Kubernetes is used for automating the deployment, management and scaling of a containerized application [19]. An overview of its complex architecture can is depicted in Figure 4. The building block of Kubernetes consists of three components: a *pod*, which is comprised of one or more Docker containers, a *node*, which runs and schedules the pods and a *cluster*, which is a collection of nodes. Kubernetes uses clusters to manage an application that will be deployed into production mode [19].

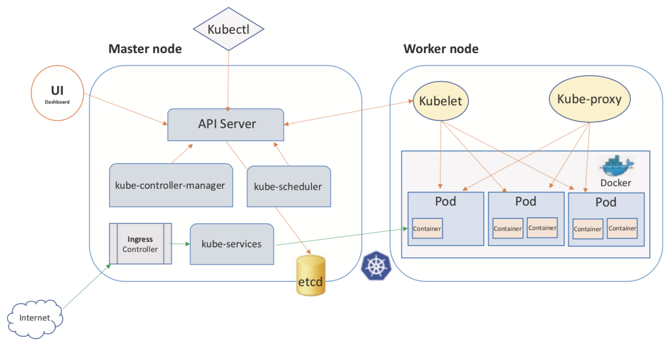


Fig. 4. Architecture model for Kubernetes, retrieved from [16]

2.3 ML-Workflow

An ML workflow defines the necessary steps that data scientists normally take when they are building prediction (or data) models. A prediction model is an algorithm that is used to understand the retrieved data and to possibly generate predictions about the future. The basic workflow for an ML project consists of several steps, including (but not limited to): problem formulation, data collection, data processing, data analysis, model construction, model validation and deployment [22][8].

Each one of the workflow steps requires special collections of packages, dependencies, frameworks or environment variables to run on. Deploying all the steps can impose cross-platform challenges and dependency issues which will result in new, time-consuming tasks for data scientists. Containerisation can alleviate these challenges by deploying each step as a microservice that will perform its specific goal consistently, in any platform that is deployed [3].

2.4 Portable and scalable ML pipelines

Containerizing an ML workflow and orchestrating the application using Kubernetes solves the portability issues that data scientists have to face when running their applications in a production environment. However, the constant increase on demands for ML workflows and big data processing workloads poses scalability requirements on the orchestration of their respective running applications.

Several platforms have been developed that give data scientists the opportunity to operate on Kubernetes-orchestrated applications without having to handle low-level operations on Kubernetes. More specifically, these platforms offer *pipelines* that will mimic the behaviour of an ML workflow by implementing the steps that data scientists have to establish themselves. In the following section, we will discuss three of such platforms, their features and how they can be used to establish ML pipelines.

3 Analysis

In this section we will introduce some of the most promising projects for the containerization of pipelines. More specifically, we will focus on three platforms: *Kubeflow*, *Cloudflow* and *OpenWhisk*, and discuss how they can be used to implement the pipelines. The criteria for the selection were:

(i) open-source software

(ii) compatibility with containers

(iii) compatibility with Kubernetes

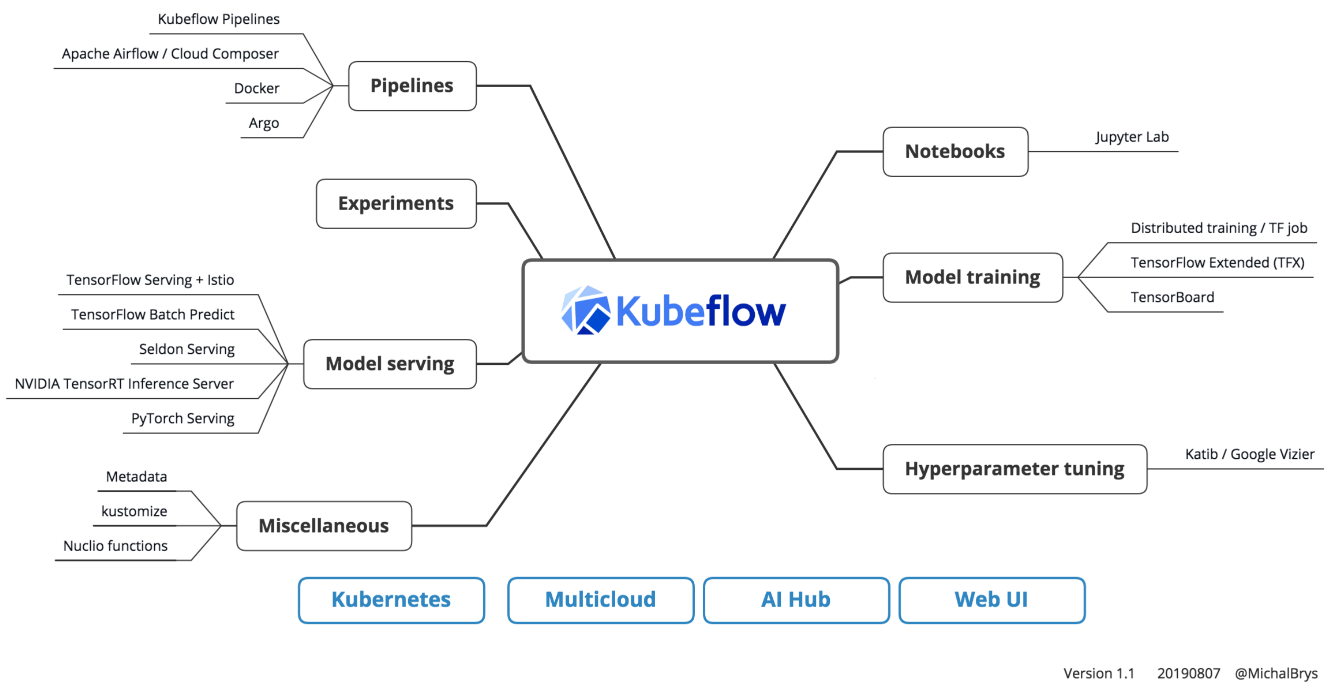


Fig. 5. Overview of Kubeflow components, retrieved from [2]

Although there are several alternative platforms that offer similar functionalities to the chosen Kubeflow and Cloudflow - *Jenkins Pipeline* [18], *MLflow* [6] or *Cubonacci* [4] which allows users to create their own pipelines within their system - these are built to work directly with containers and offer less complexity. The third chosen platform, OpenWhisk, offers an overview of the *serverless* architecture, as opposed the commonly seen containerization -where entire application or huge chunks of it are being hosted in a single container- and was created by Apache Software Foundation, a well-known name in the computing industry.

3.1 Kubeflow

Kubeflow is a platform built on top of Kubernetes and created by Google which can be used for developing and deploying an ML system. By providing a platform that creates ML workflows, Kubeflow 'skips' over the initial containerization steps, leaving data scientists with the sole task of programming their algorithms. Therefore, the purpose of using Kubeflow is to ensure that the scaling and deployment of ML models be as simple as possible by letting Kubernetes handle low-level tasks which include: *(i)* providing easy, repeatable, portable deployments on a diverse infrastructure (from the development environment to the production one) *(ii)* deploying and managing loosely-coupled microservices and *(iii)* scaling based on demand

Components of Kubeflow Kubeflow offers its services by making use of several components, which are software packages that provide specific functionalities. An overview of them can be seen in the Figure 5. We briefly introduce the key components in this section.

(a) *CENTRAL DASHBOARD* The central dashboard provides quick access to the components that are being used for the deployment.

(b) *JUPYTER NOTEBOOKS* Any Kubeflow deployment offers support for creating and managing Jupyter Notebooks.

(c) *FRAMEWORKS FOR TRAINING* Kubeflow offers five different frameworks that can be used for training ML models (Chainer Training, MPI Training, MXNet Training, PyTorch Training and TensorFlow Training).

(d) *HYPERPARAMETER TUNING* Hyperparameters are fixed parameters (they are not learned, so they do not change with training) which express important properties of the model. They are set to a given model before its regular training process is initiated. The system that Kubeflow uses for this purpose is Katib.

(e) *PIPELINES* This component is a platform that can be used specifically for the deployment of ML workflows. The goals of this component are threefold because it ensures orchestration of the pipelines, the possibility to experiment indefinitely and reusability of components.

(f) *TOOLS FOR SERVING* After an ML model has been trained, it should be made available to serve prediction requests. This component of Kubeflow handles the task of serving the ML models. Some of the available tools are KFServing, Seldon Serving, NVIDIA TensorRT Inference Server and TensorFlow Serving.

(g) *MULTI-TENANCY IN KUBEFLOW* The goal of this component is to ensure that KubeFlow supports multi-user isolation. Different users have the opportunity to isolate and protect their own resources when they have to share the same pool of resources across different teams and users.

Kubeflow pipelines Kubeflow ***pipelines*** are ready-for-use ML workflows that include all the components that data scientists would have to construct separately by themselves, such as data preprocessing, model training and more. Pipelines are built from self-contained sets of code called ***pipeline components***. A Kubeflow pipeline consists of a set of *input parameters* and a set of *tasks*. A task is an instance of a pipeline component which performs a step in the pipeline's workflow. The input parameters of the pipeline can be modified to *(a)* experiment with different sets of hyperparameters or *(b)* reuse a pipeline's workflow to train a new model.

Data scientists can either define their own pipeline components or alternatively, use the ones that are available on the AI Hub[2]. Pipeline components are composed of a set of *input parameters*, a set of *outputs* and the location of a *container image*. The container image includes the said component's executable code and a definition of the environment that the code runs in [9]. The input of a component can *(a)* depend on the input of the pipeline itself or *(b)* on the output of other components within the pipeline. The dependencies established between the pipeline components are used by Kubeflow Pipelines SDK to define the workflow of the pipeline as a graph.

When a pipeline is running, the system launches one or more Kubernetes *pods*, which correspond to the steps of the ML workflow. The pods start Docker containers, and the containers, in turn, start the programs. After users have established the pipelines according to their preferences, they can be uploaded using Kubeflow Pipelines UI or the Kubeflow Pipelines SDK.

3.2 Cloudflow

Cloudflow is an open source project to help develop, orchestrate, and operate distributed streaming applications on Kubernetes [12]. As shown in Figure 6, Cloudflow allows users to easily break down their streaming application into smaller composable components and wire them together with schema-based contracts. It is produced by Lightbend, the same company behind Akka and Play.

Cloudflow introduces the concepts of *stream* and *blueprint*. The former means that each component is a stream that can take data as input or produce it as output. This allows the user to focus on the business logic and have the system take care of the underlying deployment. Lightbend recommends using the `OODA` loop (Observe, Orient, Decide, Act) [13] for the infrastructure, because the GUI that comes with the system makes it relatively simple to monitor, alter, and/or analyse the application.

This platform is recommended for users that are already familiar with the Akka software since it also provides integration to Kafka, Spark, Flink and other run-time implementations for Streamlets. The Streamlets API allows the user to manage, scale and configure streaming applications at run-time. The GUI of the software provides insight into the user and server side, allows to link the Streamlets directly and to set their properties for scaling (although the GUI is available with a Lightbend subscription).

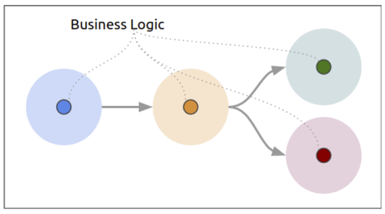


Fig. 6. Cloudflow model, retrieved from [12]

Cloudflow follows the *'let it crash'* principle and can recover from most failure scenarios, except those that are deemed catastrophic, where the data used for recovery (snapshots) may have been lost. This approach also follows the general policy of Kubernetes, where processes are ephemeral and can be restarted, replaced, or scaled up/down at any time [12]. In case of failure there are some guarantees:

(i) *At-most-once* Data may have been processed but will never be processed twice.
You may lose data, but your system will never process the same data twice.

[2]AI Hub is a platform created by Google where users have access to AI-related content including models, algorithms, research papers, components, ect.

(ii) *At-least-once* Data that has been processed may be replayed and processed again.
You may process data more than once, but each data will be processed at least once.

(iii) *At-Exactly-once* Data is processed once and only once.
This is the most sought after option but is next to impossible to achieve in distributed systems. It means that data will be processed once only in spite of any failure.

(iv) *Effectively-Exactly-Once* This means that producing the same record more than once is the same as producing it only once. Your system will process all the data and in case of processing more than one, it will make no difference

3.3 OpenWhisk

Apache OpenWhisk is an open-source, distributed *serverless* platform that executes functions in response to events at any scale [15]. As seen in Figure 7, in response to the events that OpenWhisk receives as input, it is possible to define *triggers* that, based on user-defined rules, will execute *actions*. Actions can be defined in the most widely used programming languages. It is also possible for users to create their low-level code to define them. The system will save a `JSON` file concerning the result of the action, which can be retrieved later.

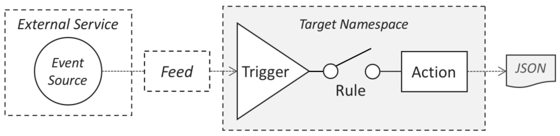


Fig. 7. OpenWhisk programming model, retrieved from [15]

As noted in Figure 8, OpenWhisk offers a multi-tier architecture, where the input is received from a front-end RESTful API which is completely HTTP based [15].

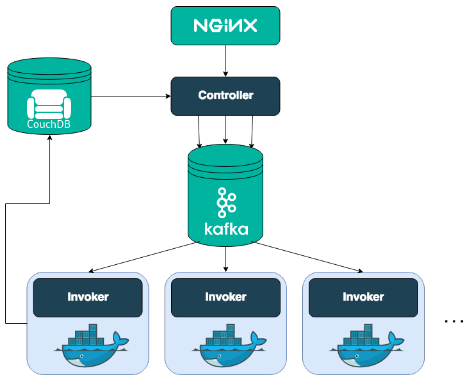


Fig. 8. OpenWhisk architecture, retrieved from [15]

We present a list of the main *components* of the OpenWhisk architecture.

(a) The *NGNIX server* translates the request towards the controller, providing the user's info.

(b) The *controller* is responsible for checking the user permission and afterwards translating the request in a real invocation of the action required.

(c) The *DB* contains both the user and object permissions, which means, for example, who can invoke what, but also how much hardware can be used for a task. This is the part where the code of all available actions is stored.

(d) The controller, knowing the status of all the executioners, acts as a *load-balancer*.

(e) To guarantee the delivery of the message in case of errors or under stress, OpenWhisk uses *Kafka* to deliver the message from the controller to the executioners called Invoker.

(f) The *Invoker*, which is the heart of the system, then executes the action inside a docker container.

4 HEURISTICS

In this section we will show how the selected platforms differ from one another, from their goals all the way to how they manage their container orchestration. Therefore, we have come up with four comparison criterions.

4.1 Platform goals

As listed in Section 3, the platforms we have chosen are quite different from one another. This can be further explained from their different goals.

(i) **Kubeflow**: The goal is to simplify the creation, deployment and re-usability of ML workflows on different environments by making use of Kubernetes.

(ii) **Cloudflow**: The goal is to ease the development, orchestration, and operation of distributed streaming applications on Kubernetes.

(iii) **OpenWhisk**: The goal is to provide an open-source, distributed, serverless platform that executes functions ($f(x)$) using multiple deployment options: Kubernetes and OpenShift, Mesos and Compose. The endorsed option is Kubernetes.

4.2 Orchestration

To achieve the aforementioned goals, each platform makes use of Kubernetes as its container orchestration in different ways:

(i) **Kubeflow** abstracts away from the orchestration details. The user is only required to define the pipeline components and their dependencies, creating what is called a pipeline specification. The specification is compiled in the Pipelines SDK and is then uploaded via the Kubeflow Pipelines UI. The user will then be able to view the pipeline graph generated from the specification and can schedule experimental runs based on the pipeline.

(ii) **Cloudflow** takes care of the links between various containers using a Blueprint. This Blueprint contains all the Streamlets and how they are connected. Cloudflow will then deploy each one of them and then connect them with the related input/output links. Blueprint is simpler than a full `YAML` file.

(iii) **OpenWhisk** does not only provide an easier orchestration, but also a different approach. After you deploy your actions, triggers and rules, OpenWhisk deploys them on Kubernetes using Helm (or its alternatives) and takes care of load-balancing, access managing and scaling (up/down) because the actions will be deployed only when needed.

4.3 Criteria

In this section we list a series of criteria that will be used to evaluate the different approach to containerization. The criteria are:

(a) Complexity: Is the system easy to use? Does the system come with overhead or savings for developments?

(b) Clarity: Is the application created with the system easy to understand, debug and reproduce?

(c) Implementation: Does the system make the development easier?

(d) Scope: What is the range of applications of the system? Can it be used with multiple programming languages/services?

5 EVALUATION AND DISCUSSION

In this section we will present the evaluation of each platform against the criteria explained before in Section 4.

5.1 Kubeflow

Kubeflow is a platform created by Google with the intention of automating the deployment and portability of machine learning pipelines. Kubeflow allows users to focus on the development of the algorithms which will be automatically ready for production mode.

(a) Complexity: Kubeflow has a learning curve, especially for users that are not entirely familiar with the concepts of containerization and container orchestration. The documentation online can help improve the initial workflow of the users.

(b) Clarity: Kubeflow is a very recent platform and various of its components are still in beta version. However, the software is extensively documented and provides example applications for new users. It should not take more than one week work to get fully accustomed to the functionalities of the current version of Kubeflow.

(c) Implementation: Kubeflow abstracts away from the low-level tasks of containerizing every step of the pipeline and managing their orchestration via Kubernetes. Instead, users are only expected to implement their machine-learning algorithms.

(d) Scope: The scope of Kubeflow lies mainly within the AI and Data Science community because the pipeline steps are representative of an ML workflow.

5.2 Cloudflow

Cloudflow is closely-related to Akka. Therefore, users that are already familiar with that software, should have a relatively easy working experience with Cloudflow as well.

(a) Complexity: Cloudflow is simple to use for users that have the subscription which offers a GUI, without the difficulty to scale up but it is still manageable.

(b) Clarity: Once setup, Cloudflow is relatively easy to maintain and understand. Everything is saved inside the blueprints and the actions.

(c) Implementation: Users that have working experience with the Akka or Lightbend technology will find it very easy to start working with Cloudflow.

(d) Scope: The range of application is quite narrow. It made of mostly for Lightbend software and some other popular software such as Kafka,Spark, Flink.

5.3 OpenWhisk

OpenWhisk uses the *serverless* approach, therefore, allows for the creation of functions to respond to whatever the user needs. It is event-based and has a set of rules to link which action should start after a trigger. This kind of approach is the most promising one for reducing the cost of working in the cloud because there are neither orchestrators, nor containers constantly running.

(a) Complexity: OpenWhisk offers a simple but newer approach. The user defines a function that will be executed on the cloud after certain triggers and rules. This paradigm, being new, has a non-trivial learning curve.

(b) Clarity: Once the system is built it can be rather simple to understand and work with.

(c) Implementation: It is relatively easy to implement OpenWhisk because it offers compatibility with a lot of software. The users are only required to define the actions, the triggers and the rules that fire the former.

(d) Scope: OpenWhisk has a really wide range of applications and it is compatible with: *.Net, Go, Java, JavaScript, PHP, Python, Ruby, Swift*. The community has also released the engine that lets Ballerina and Rust work with OpenWhisk.

5.4 Discussion

From the previous sections, we discovered the advantages of the three possible solutions, Kubeflow, Cloudflow and OpenWhisk. These three are the biggest projects in the direction of pipeline containerization, but this is not a comprehensive list. Furthermore, there is no definitive 'best' software, but only software that best fits to the user's needs. The most promising approach for cost reducing is the *serverless* one, but it is very new, and it has yet to be explored. In Table 5.4 we summarize the best systems for each criterion.

(a) Complexity: We have Cloudflow and OpenWhisk. The former is for the users already familiar with Akka, it's so for them is pretty easy. The latter instead is a simple but powerful approach, so both are considered the best for this criterion.

(b) Clarity: All of them are very clear, especially Cloudflow and Kubeflow are very tailored for their userbase.

(c) Implementation: The best for data scientist is Kubeflow not that the other two are bad, just that Kubeflow does not require deep knowledge of containerization and is tailored for them.

(d) Scope: the winner is OpenWhisk. It offers a wide range of application and it is compatible with a lot of programming languages.

	Kubeflow	Cloudflow	OpenWhisk
Complexity		x	x
Clarity	x	x	
Implementation	x		
Scope			x

Table 1. Table with the best system per category.

6 Conclusion

As described previously in Section 5, there is no absolute 'winner' since users have to decide on which platform fits their requirements best. On paper, it appears that Cloudflow is the most suitable option for the Akka users, especially those that are equipped the with Lightbend subscription. Although the GUI is not available in the open-source project that we analysed, the concept is, nonetheless, good.

OpenWhisk is the platform that offers the biggest compatibility and scope, but it is a new project and it is hard to evaluate how well will it perform. There are not a lot of publications available about the *serverless* principle, however, from the first tests that have been published, it appears that it can reduce the costs substantially.

Kubeflow is specifically dedicated towards machine learning models, and therefore, it is very specialised. This makes it the best option for data scientists that wish to implement and reuse their own ML workflows.

6.1 Future work

As future work it would be interesting to lead a study about a user group that implements the same project over the different platforms. In this way we can compare our evaluation with real data and depending on their preferred criteria, also re-evaluate the platforms. Another direction for this topic would be to extend the research with more platforms and more objective criteria, providing even more detailed suggestions for the best platforms.

Acknowledgements

The authors wish to thank dr. F.J. (Frank) Blaauw for his support.

References

[1] E. B. Andrew McAfee. Big data: The management revolution. *Harvard Business Review*, 2012.
[2] M. Brys. Kubeflow — a machine learning toolkit for kubernetes, 2019.
[3] J. Cook. *Docker for Data Science: Building Scalable and Extensible Data Infrastructure Around the Jupyter Notebook Server*. Apress, USA, 1st edition, 2017.
[4] Cubonacci. Machine learning lifecycle management. `https://www.cubonacci.com/platform-overview`.
[5] C. W. Danilo Sato, Arif Wider. Continuous delivery for machine learning, 2019.
[6] I. Databricks. An open source platform for the machine learning lifecycle. `https://mlflow.org/`.
[7] Docker. What is a container? a standardized unit of software. `https://www.docker.com/resources/what-container`.
[8] E. García del Valle, G. Lagunes García, L. Prieto Santamaría, M. Zanin, E. Menasalvas, and A. González. Disease networks and their contribution to disease understanding and drug repurposing. a survey of the state of the art, 09 2018.
[9] Google. Understanding kubeflow pipelines and components. `https://cloud.google.com/ai-hub/docs/kubeflow-pipeline`, 2020.
[10] Kumina. Top 7 benefits of using containers. `https://blog.kumina.nl/2017/04/the-benefits-of-containers-and-container-technology/`.
[11] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert. Microservices. *IEEE Software*, 2018.
[12] Lightbend. Cloudflow. `https://cloudflow.io/docs/current/index.html`, 2016.
[13] Lightbend. "how to build streaming data pipelines with akka streams, flink, and spark using cloudflow". `https://www.youtube.com/watch?v=MaXCx0fy0xU`, 2019.
[14] U. B. S. of Information. What is data science? `https://datascience.berkeley.edu/about/what-is-data-science/`.
[15] A. OpenWhisk. Apache openwhisk. `https://openwhisk.apache.org/`, 2016.
[16] C. Orchestration, S. Buchanan, J. Rangama, and N. Bellavance. *Introducing Azure Kubernetes Service*. Springer, 2019.
[17] C. Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2015.
[18] J. project. Jenkins pipeline and docker. `https://jenkins.io/doc/book/pipeline/docker/`.
[19] P. Riti. *Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes*. Springer, 2018.
[20] C. Samiullah. How to deploy machine learning models, a guide, 2019.
[21] S. J. Vaughan-Nichols. Kubernetes leads container orchestration. `https://www.zdnet.com/article/kubernetes-leads-container-orchestration/`, 2018.
[22] M. Wang, Y. Cui, G. Wang, S. Xiao, and J. Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 09 2017.

Continuous Security Testing: A Case Study on the Challenges of Integrating Dynamic Security Testing Tools in CI/CD

Remco v. Buijtenen and Thorsten Rangnau

Abstract—Continuous Integration (CI) and Continuous Delivery (CD) have become a well known practice in DevOps to ensure fast delivery of new features. This is achieved by automatically testing and releasing new software versions multiple times per day[18]. However, classical security management techniques cannot keep up with this quick Software Development Life Cycle (SDLC). Nonetheless, guaranteeing high security quality of software systems has become increasingly important [7]. The new trend of DevSecOps aims to integrate security techniques into existing DevOps practices. Especially, the automation of security testing is an important field. Although plenty of literature discusses security testing and CI/CD practices, only few deal with both topics together. Additionally, those studies cover only static code analysis but miss to address dynamic testing methods. We integrate three automated testing techniques into a CI/CD pipeline and describe the challenges and pitfalls we encounter. This will support DevOps teams that want to integrate dynamic security testing into their CI/CD pipelines.

Index Terms—DevSecOps, Dynamic Security Web Testing, Continuous Security, Continuous Integration

1 Introduction

In the past decade, a great shift occurred in software development from creating *Software as a Product* (SAAP), that is executed as a single instance on customers' machines, towards providing *Software as a Service* (SaaS) where many users share instances that run on cloud infrastructure [14]. This enabled software practitioners with the ability to continuously improve their product quality by providing frequent updates [15]. In order to manage these improvements efficiently, classical development (Dev) and operation (Ops) tasks were combined which resulted in a development concept termed DevOps [6, 15]. This concept is based on collaboration between the two former fields in all development stages and achieved by solving problems together, automating processes, and agree on mutual metrics to use when evaluating a system. This created the four pillars that guide teamwork in DevOps: culture, automation, measurement and sharing (CAMS) [17, 9]. This agile development method enables software practitioners to test and deploy software versions in a much more frequent pace and hence respond to customers' demands rapidly. A prime example of this is Amazon, where a new version was released more than once per second [18].

While fast releases are considered to be beneficial to the quality of a product, they may also increase pressure on developers to finish their tasks more quickly. Studies such as Kraemer [10] revealed that tight schedules or high work load can lead to the accidental introduction of security vulnerabilities into software systems. Kraemer also states that the reason for the presence of vulnerabilities is a lack of security knowledge in DevOps teams. This affects the quality of security tests and hence diminishes the security of a system. In addition to this, cybercrime is increasing in recent years. For instance, the number of stolen or compromised records has increased by 133% from 2017 to 2018 [16]. Furthermore, security and privacy regulations such as the General Data Protection Regulation (GDPR) have been implemented in the EU in order to enforce security standards and punish companies harshly if these regulations are violated (e.g. [5]). All of these aspects show that the security concerns have become increasingly important.

This increased focus on security introduced a new field called DevSecOps, which attempts to integrate security (Sec) practices into DevOps [15]. Traditionally, security experts were organized into separate silos and security concerns were addressed after the actual design and development stages [16]. Similar to the inception of DevOps, DevSecOps attempts to promote collaboration between development, operations and security teams. DevSecOps establishes a proactive approach to limit the attack surface of the application [18] and entails considering security from the very beginning of the project [15]. However, the integration of security practices into modern software engineering creates several problems. Firstly, traditional security methods are not applicable because they cannot keep up with the agility and speed of DevOps. Secondly, very little is known about DevSecOps so far, as only few studies were conducted on this topic [15]. Especially the lack of knowledge of when and where to use (existing) tools in automation is a considerable problem that prevents software practitioners from integrating security into their DevOps activities such as continuous integration and continuous deployment (CI/CD) [16].

Until now, research has identified the principles, priorities and practices in DevSecOps. It appears that the automation principle is equally significant in DevSecOps as it is in DevOps. One key practice is continuous testing of security. This enables security teams to keep up with DevOps and establishes fast, scalable and effective security tests [15]. However, most literature focuses on automatic security testing through static scans of source code (e.g. [11]). Although important, static tests cannot detect all security vulnerabilities in a system. In fact, static analysis is only able to find those vulnerabilities that can be derived directly from source code. These vulnerabilities are only a small subset of the ten most common vulnerabilities in web applications [1], such as *Components with Known Vulnerabilities*. Dynamic security testing on the other hand, where a system is attacked in a similar way as actual hackers would, is able to cover a much broader range of vulnerabilities. Literature such as [7, 8] describes how to execute these dynamic tests in a consistent, reproducible way. However, little is known about how to integrate this into the CI/CD pipelines commonly used in DevOps.

With this paper we aim to answer the question: *What problems can one encounter when integrating existing security testing tools into CI/CD* in order to bridge the gap between dynamic security testing tools and CI/CD pipelines. With this, we aim to provide insights that help to properly integrate security into the automated testing part of the Software Development Life Cycle (SDLC). We conduct a case study where we apply three different testing techniques in CI/CD. This will enable us to identify pitfalls, challenges and shortcomings DevOps teams may encounter while automating security tests. We perform Web Application Security Scanning (WAST) using Zed Attack Proxy

• *Remco v. Buijtenen is a Computing Science student at the University of Groningen, E-mail: r.m.van.buijtenen@student.rug.nl.*
• *Thorsten Rangnau is a Computing Science student at the University of Groningen, E-mail: t.rangnau@student.rug.nl.*

(ZAP)[1] in Section 4, Security API Scanning (SAS) with JMeter [2] in Section 4, and Behaviour Driven Security Testing (BDST) using SeleniumBase automation framework [3] in Section 4.

The remainder of this paper is structured as follows: Section 2 provides an overview on automated security testing techniques. An overview of automated testing in CI/CD is provided in Section 3. The setup of the case studies is described in Section 4, whereas Section 5 depicts our results, and in Section 6 we discuss our findings. We conclude in Section 7 and provide an overview of future work.

2 Dynamic Application Security Testing

Every software expert agrees that a software system needs to be tested before it is released. It is therefore recommended to test web or cloud applications for security flaws at the service, infrastructure, and platform level [19]. As this study focuses only on testing the service layer, those testing methods are explained in more detail. Security assessments of this layer can be divided into static and dynamic security testing [7]. Static testing or whitebox testing, describes inspection of the source code in order to find security flaws within the code such as using a libraries that are known to be vulnerable. In contrast to this, dynamic application security testing (DAST), focuses on testing a running application by testing how it response to malicious requests. This is also considered as blackbox testing because the perspective of the tests is the same as a user or hacker has on the production system and mainly focuses on scanning ports or services of the running application. In general, every attack scenario of DAST consists of a request that is sent to the system [8]. This request may contain input data that has to be processed by the application. The application answers with a response after processing the request. The challenge in security testing is to send the correct (attack) requests and to identify the information within the response that indicates the presence of a vulnerability.

2.1 DAST Techniques

In order to test security in CI/CD pipelines, those tests need to be automated and repeatable with consistent results. Therefore, this section discusses three DAST techniques that can be automated. Those are: Web Application Security Testing (WAST), Security API Scanning (SAS), and Behaviour Driven Security Testing (BDST)[8].

Web Application Security Testing (WAST)

This testing technique is an automated web security test that attacks a web application through its user interface [8]. It includes three steps performed by the WAST component: passive scan, active scan, and result reporting. The passive or spider scan explores the whole application in order to determine all URLs and resources available. The active scan performs then malicious requests against every resource. It also evaluates every response of the application in order to determine possible security issues on the targeted URL. After the active scan has been completed, the results are aggregated into a report. Usually, the scope of the scans and the attack scenarios can be configured. In addition, the security issues can be categorized into different risk levels. Figure 1 illustrates the WAST technique.

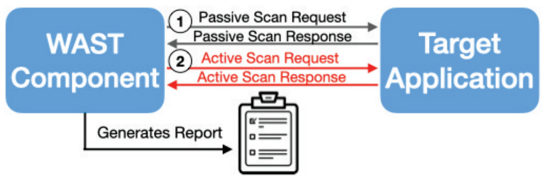


Fig. 1. Overview of WAST security testing technique: The passive scan determines all available components and the active scan attacks them.

[1]https://www.zaproxy.org/
[2]https://jmeter.apache.org

Security API Scanning (SAS)

The most important DAST technique is Security API Scanning (SAS). The WAST technique scans the entire web application but may not detect flaws of the underlying web service [8]. Therefore it is highly recommended to test the web service through its APIs with SAS. This technique allows testing of every endpoint in great detail and can cover multiple security relevant use cases such as authentication, input validation, or error handling. In SAS a parameterized request is sent to the API of the web service that is under attack. This input data can vary from adding credentials for authentication to malicious payloads such as SQL injection (SQLi). The request is made by a request component. It sends the request through a proxy component that intercepts traffic between the request component and target application. This proxy component evaluates all the intercepted traffic for any security issues. After the test is performed, the proxy component reports the result of the evaluation. SAS testing is especially useful when generated fuzz data is used as input. Fuzz data can be a list of the most common passwords, a bulk of random data in order trigger unexpected behavior of the system, or malicious input for SQLi. Figure 2 provides an overview of the SAS testing technique.

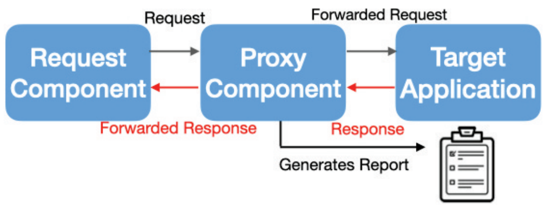


Fig. 2. Overview of SAS testing technique: Request component sends request through proxy component to target application. Proxy component evaluates traffic and generates report.

Behaviour Driven Security Testing (BDST)

Behaviour Driven Development(BDD) is an extension of Test Driven Development (TDD) and follows the idea of integrating business insights into testing [12]. BDD uses a natural languages approach in order to define behaviour and expected outcome of test cases. Behaviour Driven Security Testing (BDST) applies the idea of BDD to the domain of security testing for the added benefit that non-security experts can understand the security tests, further improving the collaboration between security experts and DevOps teams [7]. Additionally, BDST provides a dynamic security documentation of the whole software system due to the GWT (Given, When, Then) format of the test specifications. In UI tesing, BDD frameworks are used to automate standard UI tests that mimic the user behavior [12]. This approach can be used to automate the execution of attack scenario's from the hacker's perspective. Because this technique is executed against the system as a whole, this enables the identification of vulnerabilities that target multiple entrypoints to the system. BDST combines several security testing techniques such as SAS or WAST in order to mimic attack scenarios by a hacker, as well as to find security issues during normal system usage [8]. An example of a BDST setup is shown in Figure 3.

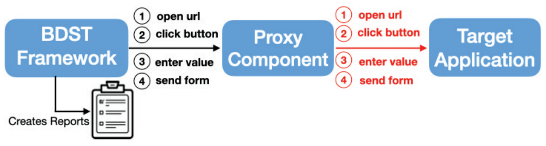


Fig. 3. Overview of BDST testing technique: The BDST framework sends behavior driven requests to the target application through the proxy component which then scans for security flaws.

2.2 Automation of DAST

As was mentioned in Section 2.1, security testing techniques must be automated before they can be integrated into a CI/CD pipeline. This section describes a number of tools that can be automated and executed in CI/CD. We will define requirements for selecting security testing tools that can be integrated into CI/CD. Each tool must have, as a minimum, a command line interface (CLI) that can be used to control testing activities such as triggering attacks or to configure testing components. More beneficial are tools that can be directly addressed through code via an API using HTTP requests or client libraries. User Interfaces (UIs) are not required except for creating test cases during development.

Automation of WAST

In order to automate WAST, the *OWASP ZAP* web security scanner can be used. It is a versatile open source tool that can be configured for multiple types of security tests. *ZAP* is a standalone application that is accessible via GUI, CLI, REST API and various client libraries. *ZAP* comes with pre-installed known attack scenario's that it will perform during its active scan. This is complemented by a spider scan that will attempt to automatically discover entrypoints in a web application that these attacks can be performed against. The result is a high level of automation with little configuration. However, these basic scans are rather limited because customized HTTP POST requests are not supported by *ZAP* in this configuration [8].

Automation of SAS

Besides the execution of predefined attack scenario's, ZAP can also be configured to act as a proxy between a testing tool and the target application. In this proxy configuration ZAP will not make its own requests so an additional tool is needed to perform the actual attacks such as *JMeter*. JMeter is a command-line tool that can perform parameterized requests against an API. Using JMeter's GUI, one can easily generate the XML files that define a test case. JMeter cannot detect vulnerabilities and therefore ZAP is inserted as a proxy between JMeter the target application. If required, JMeter can be configured to include malicious payloads for example Fuzz Testing.

Automation of BDST

ZAP can be used in combination with a BDD framework in order to perform high level use cases such as signing up, uploading files or filling in multi-stage forms. Such a framework is *SeleniumBase* [3], a wrapper for *Selenium* [8], which mimics user behavior to automate security testing for the aforementioned use case. A convenient way for developers to define the test cases for BDST is the *Selenium* IDE *Katalon*. It allows the recording of user activities on the web application such as clicking a button and converts them into an executable Python file. *SeleniumBase* executes these tests through the *Pytest* framework [2]. Similar to SAS, ZAP is inserted as a proxy between *SeleniumBase* and the target application.

3 Testing in CI/CD pipelines

Continuous integration and continuous deployment pipelines are software engineering processes used in DevOps in order to improve the efficiency of projects [4]. Such pipelines can have varying degrees of complexity, ranging from automated execution of unit- and integration-tests to full automated approval and deployment to a production server. Freely available technologies like GitLab CI[3] support managing these processes. A pipeline consists of multiple stages where each stage consists of a sequence of processes that manage building, testing and deployment of a system. A failing process stops the entire pipeline and hence deployment of the application is blocked automatically until the problem is resolved.

The extensiveness of a CI/CD pipeline can be divided into three categories, each building on top of the previous: 1) continuous integration, 2) continuous delivery and 3) continuous deployment [4]. The

[3]https://docs.gitlab.com/ee/ci

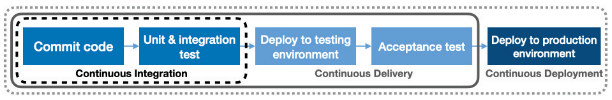


Fig. 4. The different stages of a CI/CD pipeline including continuous integration, continuous delivery and continuous deployment [13]

steps included in these categories can be seen in Figure 4. Furthermore, a pipeline can be configured to have varying degrees of complexity for different branches. For regular development, it is undesirable to execute slow tests because longer pipelines inevitably introduce delays in the the development process. However, for branches that are updated much less frequently such as those that are deployed to a testing or production server it is desirable to execute these slower but much more extensive tests in order to get better and more accurate test coverage.

3.1 Stages of a typical CI/CD pipeline

In agile software development, a new feature goes through a number of stages in a short time-span such as development, acceptance testing and deployment. Those stages are Continuous Integration, Delivery, and Deployment.

Continuous Integration

At the beginning, there is the development of a new feature where a developer writes the code and test cases. Tests are first executed locally until the feature is considered to be ready for testing [16]. At this point, the code is committed and pushed to a remote repository and a merge request will be created. This process will trigger the CI/CD pipeline for the application as a whole, where the new feature is combined with work from other developers. When all test cases have passed the DevOps team can decide to advance to the testing stage.

Continuous Delivery and Continuous Deployment

Continuous Delivery builds on top of continuous integration by automating the deployment of the work done in a sprint to a testing server [4]. This additional step is triggered whenever a new feature is merged into the testing branch which will in turn run all tests from the continuous integration step to ensure that the merge did not break any existing work. In addition to this, more extensive test cases can be executed in order to cover much of the acceptance testing that would otherwise have to be done manually. This shifts the workload for management from doing actual testing to verifying that the automated tests did their job correctly. Assuming that tests are thorough and well-defined, one can be assured that a feature works correctly as long as the tests pass. Assuming, of course, that the tests are thorough and well-defined.

Another extension of continuous delivery is continuous deployment, where an approved version of the testing server will be deployed to a production environment [4]. In this step the configuration of the application is updated to make it ready for production by disabling features used only for development and updating application secrets. Because this updated configuration has not been tested before, it may contain additional vulnerabilities. Therefore it is necessary to run all previously executed tests one last time. When the pipeline passes, the application is deployed to the production server where new features are made available to the user.

4 Methodology

In this section we discuss the integration of the three automated test techniques described in Section 2 into CI/CD pipelines. This enables us to not only evaluate the effectiveness of these techniques but also to identify challenges of applying DAST techniques to CI/CD. The findings can be helpful for software practitioners to extend their existing testing techniques and thus help to improve security quality of agile web development.

In addition to the tools that help to automate security testing, we used several other technologies in order to provide a test environment that can be executed on any CI platform. In order to build, execute and share the required applications involved in the particular test cases, we used the virtualization software Docker[4]. Docker enables us to containerize every application and run it e.g. locally or remotely in the CI environment. Further, the Docker Compose tool allows to create and combine multiple containers which is important to connect the different testing tools with the test application. For the CI environment we decided to use GitLab CI. The usage of this cloud platform is not only free but also provides CI/CD pipelines as a service[5].

Finding a way to test the integration of automated DAST in CI/CD is challenging because it requires an adequately sized web application in order to make statements about the different security tests. Since it is out of scope of this study to develop such an application with specific vulnerabilities, OWASP WebGoat was chosen as a target application. WebGoat is a deliberately insecure application that was designed for educational reasons on the one hand and for testing security tools on the other hand. We decided to use this open source project because its 89.100 lines of code represent a reasonably sized web application. In addition, the WebGoat community already provides a WebGoat docker image which can be pulled directly from the public repository. Furthermore, the documentation of WebGoat includes information about its vulnerabilities such as SQLi, and how to exploit them.

A CI pipeline has been implemented for all three testing approaches in order to demonstrate that this approach works for a variety of testing tools. Each pipeline has been divided into three stages: build, test, and deploy.

Building of docker images is executed in the *build* stage of the pipeline. Each testing approach requires two docker images to be built, resulting in a total of 6 images being built. In order to speed up this process, each build is executed as a separate parallel job. A build job consists of three steps: First, a login to a remote docker repository is required. Secondly, the docker images are built and and tagged using the commit hash of the current branch. This allows us to push images multiple branches to the same repository without them interfering with each other. Finally, the images are pushed to the remote repository so they can be used in the next stage.

The second stage is testing, where the images from the previous stage are pulled from the remote docker repository and security tests are then executed. This stage uses a docker-compose in docker image[6], allowing us to run the test setup exactly as one would do on a local machine. Because the images must be pulled from a remote repository, a custom python script is used to update the *build* section of the compose file with an *image* entry pointing to the remote repository. Results are written into a volume that is shared with the CI pipeline. The contents of this directory are exported as GitLab CI artifacts.

Tools such as PyTest and JMeter require a running application to test. Since docker-compose's container dependency feature does not wait for a web server to be ready, this had to be implemented manually. For this the *wait-for-it.sh*[7] was used to delay starting the testing process until all dependencies are ready to respond to requests.

ZAP and WebGoat are web services that keep running until they are explicitly stopped. This means that without sending a shutdown signal, the CI/CD pipeline will never terminate. Therefore, at the end of the testing step a shutdown command is sent to ZAP through its API. This results in a graceful termination of ZAP. However, WebGoat does not provide such an endpoint and therefore a different approach is needed. In order to ensure that WebGoat terminates after testing, the testing tool's image has been constructed using the same docker-compose in docker image as the CI pipeline itself. In order to make this work, a volume must be mounted into this image to make the pipeline's docker daemon socket available to docker inside the testing tool's image. After test execution the testing tool is then able to call the *docker kill* command. This will cause WebGoat to terminate once testing is complete.

The aforementioned artifacts that were exported in the previous stage are imported into an evaluation step. This last step then executes a python script that reads the JSON output from ZAP, and uses this to decide whether the pipeline should pass or fail. If a pipeline should fail, it is sufficient to exit the script with a non-zero status code. For an actual production setup, this step would then be followed by a deploy step. This deploy step will only be executed if the evaluation script exits with a zero as status code.

Integration of WAST in CI/CD

For the WAST integration we used the containerized ZAP and WebGoat containers. In order to control the tests, we added another container containing a simple script written in Python. The script makes use of the Python ZAP client library. With this library, one can easily control pro-active scans of ZAP. It receives the URL address of the WebGoat application as an argument to trigger passive and active scans in ZAP. This setup allows a complete scan of the WebGoat application. After the scans are finished, the aforementioned methods are used in order to terminate the docker-compose setup and evaluate the test results.

Integration of SAS in CI/CD

For the SAS test scenario we used again the containerized ZAP application for detecting malicious HTTP traffic. However, ZAP is now used in proxy mode and hence only forwards all traffic to the target application and analyses the responses. As was explained in Section 2, we used JMeter to execute specific API scans. Therefore, we installed JMeter and the `.jmx` files in into a docker container. In order to execute the tests one has to add entrypoints to the container that takes arguments and forwards them to the JMeter CLI inside the container. Thus, we can dynamically specify test files and setups ZAP as the proxy. This is important to detect security issues within the HTTP communication initiated by JMeter.

Integration of BDST in CI/CD

For the BDST technique we applied the SeleniumBase framework, which is installed in its own docker container together with two test cases. The first test case registers in the WebGoat application and the second uses the credentials created to log in and perform an SQLi attack. Similar to the JMeter docker container we needed to add an extra docker entrypoint in order to start seleniumbase via the docker-compose command section. Because the seleniumbase configuration refuses to accept the default docker-compose generated host names to configure ZAP as proxy (docker-compose gives random IP addresses to each container which can then be accessed using a mnemonic host name), we had to add a customized docker network to introduce static IP addresses to each container in order to configure scanning for vulnerabilities while the test is executed. Before we could execute the tests we had to change the URLs in the test cases to the new docker environment. This is because the automatic generated selenium files uses the URLs of the local setup during recording rather than the one used in CI/CD.

5 Results

In this section we present our findings that we encountered during the integration and execution of the automated DAST in CI/CD. The results for every test for each CI job can be downloaded from the GitLab CI interface. This reveals that all three security tests have detected vulnerabilities [8]. The detection of these vulnerabilities cause the evaluation step of the pipeline to fail as intended. A single CI job, covering all three test scenarios, takes 14 minutes and 6 seconds. This run-time includes building of all components, starting the applications,

[4]https://www.docker.com

[5]https://docs.gitlab.com/ee/ci

[6]https://hub.docker.com/r/docker/compose/

[7]https://github.com/vishnubob/wait-for-it

[8]All results are derived analysing this CI job `https://gitlab.com/rvbuijtenen/continuous-security/pipelines/128935397`

Table 1. Vulnerabilities detected for each automated DAST technique

Test Type	# Tests	Inform.	Low	Medium	High	Total
WAST	13 URLs	7	6	1	1	15
SAS	1 URL	1	3	0	0	4
BDST	2 UCs	32	28	0	0	50

performing the tests, evaluating the results of all tests, and exporting the artifacts containing the detected vulnerabilities. Subsequently, the three test techniques will be discussed individually. All setups used the WebGoat application as a service under attack, which is deployed using an already existing docker image. Therefore, it has no building time.

Results for WAST, SAS and BDST CI Jobs

WAST included three docker containers. The building time for ZAP container is 6 minutes and 52 seconds and the container used to control ZAP requires 1 minute and 32 seconds. The execution of WAST takes 6 minutes and 4 seconds in total. The passive scan identified 13 resources along the path `http://webgoat:8080/WebGoat/` and the active scan detected 15 vulnerabilities. ZAP categorized those vulnerabilities by risk which results into 7 “Informational”, 6 “Low”, 1 “Medium”, and 1 “High” risk. The vulnerability with the high risk was detected on the address `http://webgoat:8080/WebGoat/register.mvc` and denotes this resource to be vulnerable against a SQLi attack. Finally, the evaluation of this test technique took 1 minute and 8 seconds.

The SAS test setup also requires three components. The build time for its two build components are: ZAP in 6 minutes and 9 seconds and JMeter in 2 minute and 2 seconds. The run-time of the test takes 2 minutes and 31 seconds. The test was performed against `http://webgoat:8080/WebGoat/login` and ZAP detected two addresses that are vulnerable. The first address is exposed to a two “Low” risk vulnerability and one “Informational” risk. The second address is liable against 1 vulnerability which is a “Low”risk.

The BDST setup needs to build two containers, namely ZAP and SeleniumBase. The first component takes 6 minutes and 8 seconds to build and the second 4 minutes and 31 seconds. The duration of the test stage is 3 minutes and 45 seconds. During the two performed behaviour driven tests, ZAP detected 32 vulnerabilities, composed by 28 “Informational” and 4 “Low” risk security issues. Interesting is that one of the test cases included an SQLi attack where user passwords were exposed inserting SQL commands into the username field. This attack was not detected by ZAP. The evaluation of the test results took 1 minute and 5 seconds. An overview of these results can be found in Figure 5 and Table 1.

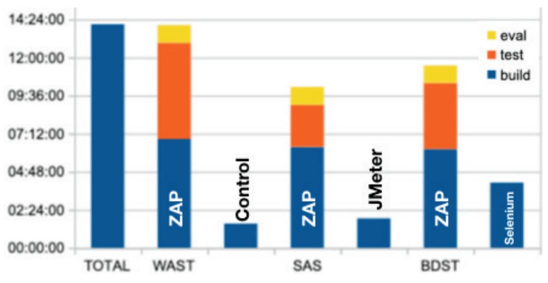


Fig. 5. Build-, test-, and evaluation-time for all pipeline stages

Challenges of security testing in CI/CD

In contrast to the quantified results of the case studies, we also present qualitative results because they are important to identify challenges in the integration of automated DAST into CI/CD. Solutions to the problems that are discussed here are presented in Section 6.

In the docker-compose setup of all three testing techniques, we encountered the problem that all containers were marked as ready but the application inside the container was still starting. This resulted in tests being triggered while WebGoat and ZAP were not ready. For SAS this caused the program to exit without any test results, while for BDST this caused the SeleniumBase container to crash with an error.

Another reoccurring problem was pipeline termination. Despite the tests finishing as intended, the remaining docker containers were still running. This is not a surprise because ZAP and WebGoat are standalone software systems that are designed to run until they are stopped explicitly. If this is not done the CI job will never finish and one could never determine if the dynamic security test has passed or failed.

A similar problem related to containerization was to get the results form ZAP. Since zap provides its results through a web UI, there was no clear way to extract these from the container. However, it is possible to make an HTTP request that downloads the test results in JSON or HTML format.

Another problem occurred with using SeleniumBase as BDD framework. SeleniumBase can be configured to redirect requests through a proxy which works fine in native installations. However, SeleniumBase only accepts an IP address as a proxy target. Because docker-compose assigns a dynamic IP to a container when it is started, it is not possible to refer to this IP using the default configuration, hence further customization is required.

Using SeleniumBase we defined a test case that performed an SQLi against the WebGoat application. However, it turns out that we configure is between the testing application and WebGoat’s UI, rather than between WebGoat and its API. This resulted in ZAP not detecting the presence of leaked information because the leak was outside of the scope of what SeleniumBase was able to test.

6 Discussion

This study investigates how dynamic security test can be integrated into CI/CD pipelines and what challenges software practitioners have to meet with this integration. As the results show, all three tests can be performed, vulnerabilities were detected applying existing tools for test automation, and the evaluation of the results stopped the pipeline which would prevent undesired deployment of security flaws to a production system. This shows that adding security tests into CI/CD pipelines in general is feasible. One can easily see that the tests detected several security issues that were categorized on a low or even informal risk level (Table 1). Depending on the scope of the system, the evaluation of the ZAP results can be configured in such a way that those alerts are ignored or only reported but do not lead to a pipeline failure.

In order to prevent the introduction of additional delays into the development process, it is important to keep the run-time of a pipeline to a minimum. The duration of the resented pipeline is around 13 minutes. Adding more test cases will increase this run-time, but as the project grows this can be reduced by distributing test cases among multiple parallel CI jobs. Therefore, we consider this result as adequate as it is acceptable for a software developer to wait around 15 minutes for the results of the scans. Note that this is only applicable for SAS and BDST due to the definition of individual test cases. WAST using ZAP in proactive mode provides no built-in functionality that allows for distributed testing. If a WAST scan takes longer than what is considered as an acceptable waiting time for regular development, we recommend to only execute this type of testing for the continuous delivery and continuous deployment stages of the project.

Building the ZAP image is the slowest part of the CI/CD pipeline. A closer look at our setup shows that we use the same ZAP version to build each docker image and this version is independent of individual test configuration. Therefore, it is possible to create a pre-built version of this image and host it in a public or private repository. Considering that the run-time of the entire CI job is composed of the longest duration of every stage, this would save approximately 4,5 minutes for

WAST, 4 for SAS and 2 minutes for BDST. The image then only has to be rebuild when a new version of ZAP is released.

We found that the WAST technique requires the least amount of integration effort. However, the default setup of ZAP that was used in this study is not capable of finding all resources since the passive scan cannot detect resources that e.g. require authorization [8]. Also, the run-time of passive and active scans grows with application size. While testing the different automation techniques, we performed a scan against the *AltoroMutal* test web application. This scan lasted several hours and is therefore not feasible for testing every single commit. For large projects we recommend to perform an entire scan only before a release.

The BDST allows testing from the perspective of a hacker. The setup that is suggested is capable of performing these scenarios as our two test cases show. However, ZAP was not able to detect the SQLi attack scenario. This is not a surprise because ZAP is only a proxy between the BDD framework and the web application. The malicious request however is send between the web application and its underlying web service. In order to detect those security flaws we suggest to use SAS in addition to BDST to analyse the requests sent to the web service's API.

The SAS testing technique is the most flexible because it allows to test every single endpoint individually. This is important as was shown by the previous example of BDST. However, creating and managing tests for every single endpoint of an API can become increasingly complex for large applications. Furthermore, the flexibility of the tests can easily lead to forgetting certain aspects in the tests. Therefore, we suggest to consider SAS already in the API design. The team should fall back to the experience of a security expert in order to determine possible attack scenarios against this API. Subsequently, the API development should follow test driven development (TDD) and start with creating the SAS test case.

Solving the challenges of security testing in CI/CD

As was stated in Section 5, we encountered several challenges during our case studies. Generally speaking, we found that many of these challenges come from the isolated nature of containerized applications and therefor a fair amount of knowledge of tools like Docker and GitLab CI are required. In the remainder of this section we discuss how we managed to overcome most of these challenges.

The first problem we encountered was related to the startup order of services, where testing tools were executed before ZAP and WebGoat were ready for requests. For this, we added the wait-for-it.sh[9] script to each docker container. In addition to this, we added a customized entrypoint for the testing container that uses this script to wait for the other services to be up and running. This is achieved by attempting to make an HTTP request to a known endpoint of the dependencies until it succeeds. The result is that the testing container is paused until all services are available, thus solving this issue.

The second problem was that the CI/CD pipeline did not terminate within the given timeout of 1 hour due to web services that wait for requests until an explicit shutdown command was given. We have extended the aforementioned entrypoint to make a CURL request to ZAP's shutdown endpoint, resulting in a graceful termination of the ZAP container. For WebGoat, another approach was required since it does not provide an API endpoint like ZAP does. Instead, we used the docker-in-docker image to use the *docker kill webgoat* command to terminate WebGoat.

In order to extract results from ZAP's web UI, we mounted a volume for results into the testing containers (JMeter and SeleniumBase). Since it takes a few seconds for ZAP to aggregate the results, we added a 10 second delay after test execution followed by a CURL request to ZAP's UI to extract the HTML and JSON results. These were then written to the volume, and the contents of this volume were then exported as a GitLab CI artifact that can be downloaded by developers. This delay and the execution of CURL requests were once again added to the aforementioned entrypoint.

[9]https://github.com/vishnubob/wait-for-it

The problem that we encountered with SeleniumBase only accepting a fixed IP address as its proxy address was solved using a custom virtual network defined in our docker-compose setup. This allowed us to assign a fixed IP address to each image, which in turn enables us to provide this fixed IP to SeleniumBase resulting in it correctly using the ZAP container as a proxy for the test cases.

7 Future Work

A possible future extension of our work is to improve the reporting of results. Currently, we take the output provided by ZAP for all three testing methods as-is and aggregate them into a single ZIP file. A drawback of this approach is that results from multiple testing methods are reported separately, and this may result duplicate vulnerability detections. Therefore it would be a good extension to further process these results and provide a single, well organised method of reporting the results. Furthermore, in the case of SAS, ZAP does not distinguish between multiple tests against the same endpoint. Therefore it is necessary to find a way to map these results back to individual test cases to provide clear results to developers.

References

[1] Owasp top ten. `https://owasp.org/www-project-top-ten/`. Accessed: 27-02-2020.

[2] Pytest (https://docs.pytest.org/en/latest/contents.html). Accessed: 22-03-2020.

[3] Seleniumbase (https://seleniumbase.com/). Accessed: 16-03-2020.

[4] S. A. I. B. S. Arachchi and I. Perera. Continuous integration and continuous delivery pipeline automation for agile software project management. In *2018 Moratuwa Engineering Research Conference (MERCon)*, pages 156–161, May 2018.

[5] G. EU.org. Gdpr eu.org - fines and penalties. `https://www.gdpreu.org/compliance/fines-and-penalties/`. Accessed: 26-02-2020.

[6] B. Fitzgerald and K.-J. Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 25, 07 2015.

[7] T. Hsu. *Hands-On Security in DevOps: Ensure Continuous Security, Deployment, and Delivery with DevSecOps*. Packt Publishing, 2018.

[8] T. H.-C. Hsu. Practical security automation and testing: tools and techniques for automated security scanning and testing in devsecops. 2019.

[9] J. Humble and J. Molesky. Why enterprises must adopt devops to enable continuous delivery. 24:6–12, 08 2011.

[10] S. Kraemer, P. Carayon, and J. Clem. Human and organizational factors in computer and information security: Pathways to vulnerabilities. *Computers Security*, 28:509–520, 10 2009.

[11] M. Kreitz. Security by design in software engineering. *SIGSOFT Softw. Eng. Notes*, 44(3):23, Nov. 2019.

[12] R. K. Lenka, S. Kumar, and S. Mamgain. Behavior driven development: Tools and challenges. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 1032–1037, Oct 2018.

[13] A. Mehdi. Continuous release practices are evolving, here is our story (https://medium.com/the-telegraph-engineering/continuous-release-practices-are-evolving-here-is-our-story-2a4d164e9cac). Accessed: 23-03-2020.

[14] P. M. Mell and T. Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, USA, 2011.

[15] H. Myrbakken and R. Colomo-Palacios. Devsecops: A multivocal literature review. pages 17–29, 09 2017.

[16] N. Tomas, J. Li, and H. Huang. An empirical study on culture, automation, measurement, and sharing of devsecops. pages 1–8, 06 2019.

[17] J. Willis. What devops means to me. `https://blog.chef.io/what-devops-means-to-me/`, 07 2010. Accessed: 26-02-2020.

[18] H. Yasar and K. Kontostathis. Where to integrate security practices on devops platform. *International Journal of Secure Software Engineering*, 7:39–50, 10 2016.

[19] P. Zech. Risk-based security testing in cloud computing environments. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 411–414, March 2011.

A survey on surface interrogation methods

Luc Breeman and Robert Riesebos

Abstract— In geometric modelling, computer-aided design, and computer graphics shape smoothness can be an important attribute of a surface. In order to ensure that a surface is sufficiently smooth, a process called surface interrogation is applied. In this process the surface is analysed in order to detect and fix irregularities. The smoothness and quality can be assessed with the help of lines displayed on a shape or surface. Several surface interrogation methods exist, such as contour lines, isophotes, highlight lines and reflection lines. In this paper an overview of these methods is provided, comparing them on criteria such as sensitivity to changes, intuitiveness and relation to underlying smoothness. Finally recommendations are given on when to use which method.

Index Terms—Geometric Modelling, Computer Aided Design, Computer Graphics, Surface Interrogation, Surface Smoothness, Contour Lines, Isolines, Isophotes, Highlight Lines, Reflection Lines

1 Introduction

Computer Aided Design (CAD) has become a standard design tool for architecture, engineering and construction [2]. Several sectors employ the use of CAD systems for the design process of their products. Examples include the automotive industry, aviation industry and ship design sector [20, 24]. An important aspect of Computer Aided Design is the evaluation of surfaces, mainly with respect to their smoothness. Cars require smooth surfaces for optimal performance and visual appeal, ship hulls need smooth surfaces in order to have as little friction as possible, and blades in aircraft engines require exact sizes in order to work efficiently and effectively. Smooth surfaces are also essential for creating high quality 3D models.

In order to guarantee that the generated surfaces are as smooth as the product requires, engineers perform a process called “surface interrogation”. As the name suggests, they take a detailed look at the surface in order to detect and correct surface irregularities. Surface interrogation methods provide engineers with tools to analyse the intrinsic shape and curvature of surfaces, with the goal of finding anomalies and imperfections [13].

Each surface interrogation method has its benefits and drawbacks. Certain situations require the use of one method over another, while in others it is simply a matter of preference of the engineer. This survey aims to provide a clear overview of several surface interrogation methods, and serve as a pointer on when to use which method. We present a comparison between four surface interrogation methods: contour lines, isophotes, highlight lines and reflection lines. A thorough literature research has been done to gain knowledge of each method. We applied this knowledge to compare the methods and constructed example surfaces to aid our explanation. For each method we first give a short theoretical background. Then we provide a more detailed analysis and comparison. Finally recommendations on which methods are best suited for certain tasks are given, followed by a conclusion highlighting the results of this survey.

In Section 2, we introduce four surface interrogation methods and provide theoretical definitions of each method. Section 3 discusses the pros and cons of each methods, along with special properties of each method. In Subsection 3.5 a direct comparison is made, comparing the methods on the properties of *maximum observable surface continuity*, *sensitivity to changes*, *view-point dependency* and *intuitiveness*. In Section 4 we provide recommendations on when to use which method. Finally, in Section 5, we provide a conclusion and an indication of possible future work in this field of research.

- *Luc Breeman, Msc. student Computing Science at the University of Groningen, E-mail: l.breeman@student.rug.nl.*
- *Robert Riesebos, Msc. student Computing Science at the University of Groningen, E-mail: r.j.riesebos@student.rug.nl.*

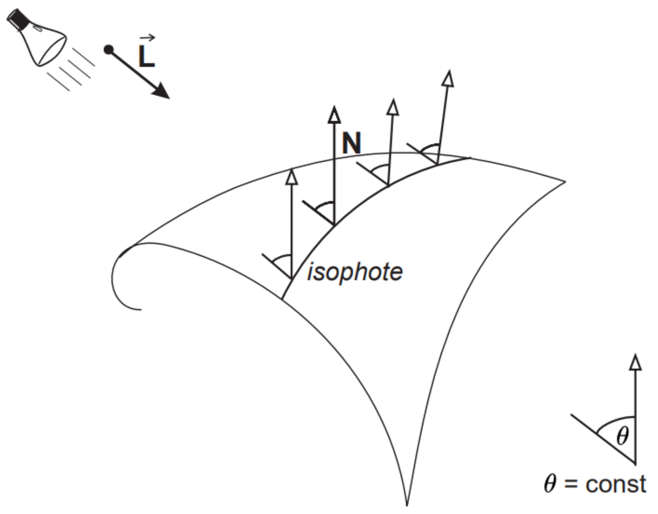


Fig. 1: Isophote illustration (adapted from [13])

2 Surface interrogation methods

Several surface interrogation methods exist for the purpose of analysing surfaces. In this section, we introduce the four surface interrogation methods discussed in this survey. General information and a theoretical definition of each method is given in order to familiarise the reader with the details of each method.

2.1 Contour lines

The use of contour lines, also referred to as isolines, provide a natural interpretation of computer-generated curves surfaces [5]. So called “contour-maps” are created by intersecting the interrogated surface by a family of user-defined, (non-)uniformly spaced parallel planes [5, 8]. In this survey we discuss contour maps where the distance between each ”contour-level” remains constant. This can formally be described as $z = constant$ for a 3d surface, with the 3 axes x, y, z defining the space [20].

The contour lines representing different levels, referred to as “level-curves”, are spaced at equal intervals. Contour-maps enable the user to identify critical points of the surface, which are points with special properties. Examples of these points are minima, maxima and saddle points, where saddle points represent critical points where two contour lines intersect [5].

2.2 Isophotes

Isophotes are curves of constant light intensity on a surface [12, 20, 22]. They are represented by lines connecting all points on a surface for which the isophote condition, given by Equation (1), holds.

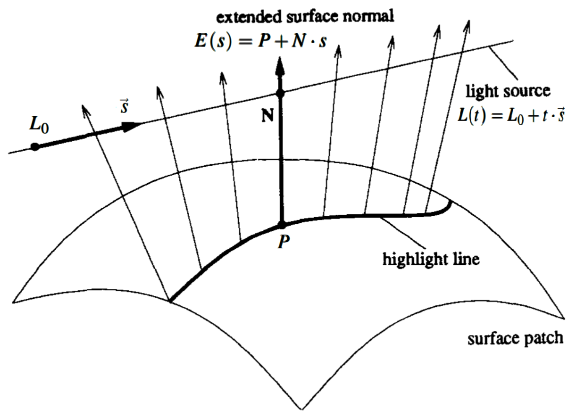


Fig. 2: Highlight line illustration (adapted from [6])

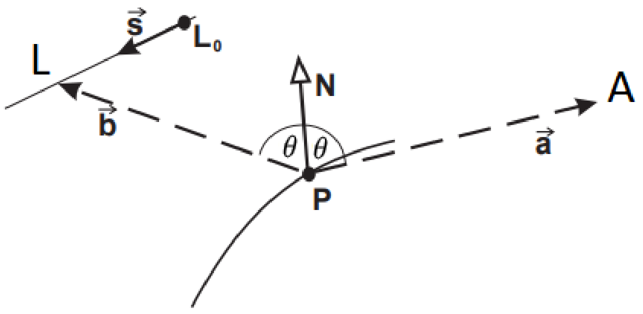


Fig. 3: Reflection line illustration (adapted from [13])

If $X(u,v)$ is a parameterization of a surface and L is the direction of a parallel lighting — a lighting produced by a point light source at infinity with direction L — then the isophote condition is given by:

$$N(u,v) \cdot L = \cos(\theta) = c\ , \tag{1}$$

where $N(u,v)$ is the unit normal vector of the surface $X(u,v)$ and c is a constant representing the light intensity on the surface. The angle θ represents the angle between the unit normal vector $N(u,v)$ and the light direction unit vector L [12, 22].
The isophote condition holds for angles $\theta \in [-90, 90]$, but in practical applications such as surface interrogation only angles between 0 and 90 degrees ($\theta \in [0,90]$, $c \in [0,1]$) are considered [20, 22]. This is because only curves with angles between 0 and 90 degrees are in the illuminated part of the surface. A visual representation is given in Figure 1.
If the surface is C^k-continuous, then the isophote will be C^{k-1}-continuous [12, 15, 22, 26]. For planar parts of the surface all normals are parallel and isophotes are not well defined. In case of $c = 0$ the light direction is perpendicular to the surface normal, and the isophote is the surface silhouette. Finally, for $c = 1$ the light direction is perpendicular to the surface.

2.3 Highlight lines

Highlight lines can be effectively used as smoothness indicators for smooth surfaces. By moving the light source, the user is able to sweep the highlight line over the entire surface, thus enabling him/her to inspect the whole surface [6, 28]. As with isophotes and reflection lines, the calculation of highlight lines involves solving a system of partial differential equations. Highlight lines can be described by the following set of equations [28]: We denote a linear light source $L(t)$, with the definition

$$L(t) = L_0 + t \cdot \vec{s}, \quad \text{where } t \in \mathbb{R}\ . \tag{2}$$

L_0 is a point on $L(t)$, and $\vec{s}$ represents a vector defining the direction of $L(t)$. For a surface point P, let N be the normal vector of the surface at P.
The line at $E(s)$ that passes through Q in the direction of N can be defined by the formula:

$$E(s) = P + N \cdot s\ . \tag{3}$$

In order to know if point Q belongs to the highlight line, $L(t)$ and $E(s)$ have to intersect, which is true if the perpendicular distance d, given by (4), is zero.

$$d = \frac{|(\vec{s} \times N) \cdot (L_0 - P)|}{\| (\vec{s} \times N) \|}\ . \tag{4}$$

Highlight lines have two interesting properties that make them suitable for surface interrogation [6]. The first one is that a highlight line is viewer independent. The line is not dependent on the viewing point and angle of the user, and thus does not require new calculations if the user wants to view the surface from another view-point.
The second property is the fact that imperfections of a surface are magnified by an order of one in the highlight line. There are several orders of discontinuity in surfaces, and highlight lines have the ability to magnify these imperfections. The order of one relates to how the highlight lines react to this imperfection. The involvement of surface normals in the calculations means that small changes in the surface result in changes of the surface normals, which get magnified during the calculations of the highlight lines. This enables the user to clearly spot small imperfections in the interrogated surface.

2.4 Reflection lines

Reflection lines were first introduced by Klass in [17]. Much like highlight lines, reflection lines are used to determine unwanted curvature regions (dents) by inspecting irregularities in the reflection line pattern of parallel light lines. The main difference between reflection lines and highlight lines is that the view point is fixed for reflection lines [6, 17].
To start with explaining reflection lines we again let $X(u,v)$ be a parameterization of a surface and $N(u,v)$ the unit normal vector of the surface. We define a reflection line as the reflected image of a straight line L on a surface seen from a fixed view point A. Line L is the so-called light line, and is defined in Equation (2). Next, to formulate a reflection condition for a reflection point P on surface $X(u,v)$, we abbreviate normalized directions PA and PL to $\vec{a}$ and $\vec{b}$ respectively.
Using these abbreviations and Figure 3 we can derive the following reflection condition:

$$\begin{aligned} \vec{a} + \vec{b} &= 2(N(u,v) \cdot \vec{b})N(u,v) \\ &= 2(N(u,v) \cdot \vec{a})N(u,v)\ . \end{aligned} \tag{5}$$

After a simple transformation of Equation (5) we obtain Equation (6), providing a relation between the light point, view point and reflection point [17, 12, 20].

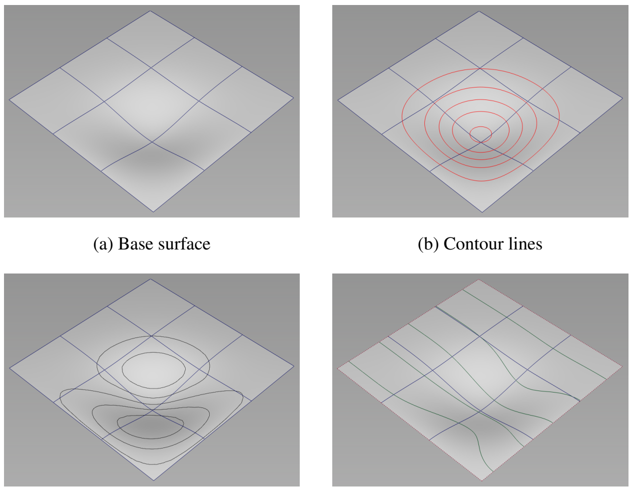
$$\vec{a} = 2(N(u,v) \cdot \vec{b})\, N(u,v) - \vec{b} \tag{6}$$

This relation can be used to evaluate a curve on the surface. When we move L on the light line given by Equation (2) and keep A fixed, point P will move in a curve on the surface; creating the reflection line.

As explored in [26], isophotes and reflection lines are different (but not disjunct) classes of surface curves. To show this, the definition of reflection lines can be simplified in such a way that they are only dependent on the surface normals, and not on the view-point and light line. This enables us to directly compare the two methods.

3 Comparative Analysis

In this section we perform a comparative analysis of the surface interrogation methods introduced in Section 2. Before we do so, we first provide visual examples to illustrate each method.



(a) Base surface

(b) Contour lines

(c) Isophotes

(d) Highlight/reflection lines

Fig. 4: Result of different surface interrogation methods on the same base surface

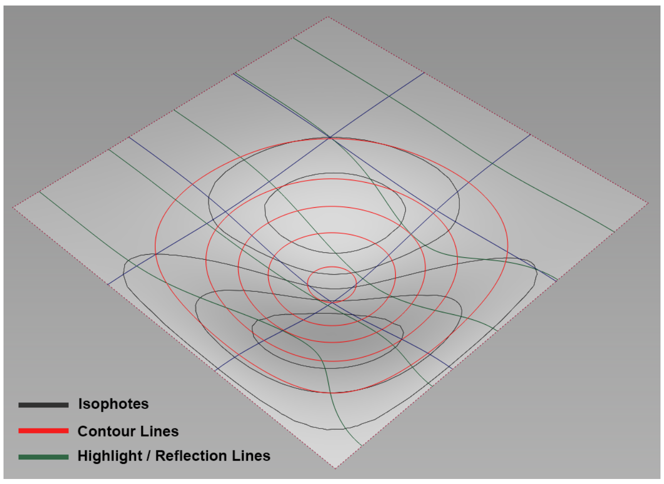


Fig. 5: Combined results of contour lines, isophotes and highlight/reflection lines

The images in Figure 4 were created using Autodesk Alias Surface 2020 [4]. The base surface in Figure 4a is a C^2-continuous surface. It is manipulated to have a small irregularity below the bottom-most corner, where it is dented in a downwards direction. Patches of a compound surface are divided by blue lines. The lines in Figure 4d represent both highlight lines and reflection lines because the view point is stationary. In Figure 5 a larger image is provided showing all methods in one figure. Finally, Figure 6 shows a surface with a similar irregularity as Figure 4a, but this time the surface is C^1-continuous. Isophotes and reflection/highlight lines are drawn on this surface in black and green respectively.

In the next few subsections advantages and disadvantages of each surface interrogation method are discussed, along with general information and properties unique to the method. A comparison table is provided to give a clear view of how all of the methods compare.

3.1 Contour lines

Contour lines provide a simple yet unambiguous view of the surface to the user. Since these lines relate directly to the shape of the surface, the user is able to quickly get an idea of the shape of the surface [9]. Contour lines are excellent for spotting interesting surface features

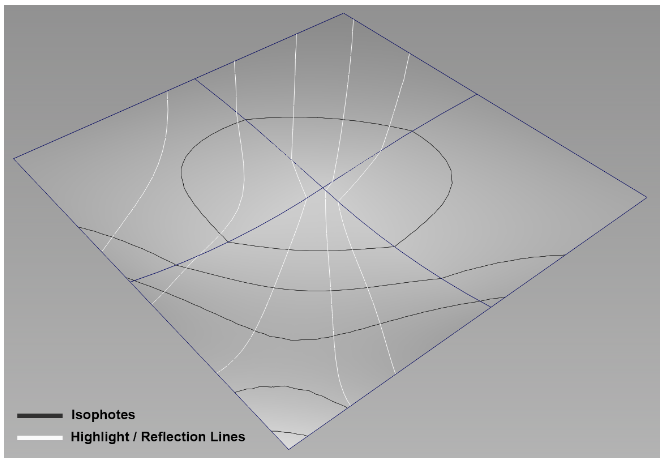


Fig. 6: Isophotes and reflection/highlight lines on a C^1-continuous surface

such as ridges and valleys. They also allow the user to find special points on the interrogated surface, such as maxima, minima and saddle points [5]. Contour lines are able to describe 3d features in a 2d medium, thus providing more information of the surface than only looking at the rendered surface [21]. This can be clearly seen in Figure 4b. The irregularity of the surface in Figure 4a can be difficult to spot. The exact position of the surface irregularity can be easily identified in Figure 4b, along with the impact of the irregularity on the rest of the surface.
However, the use of contour lines also has several downsides. The process of generating a contour map requires solving plane intersection problems for a number of planes. A mathematical solution of these problems require high computational effort, especially for a large number of planes. Guid et al [10] mention that parabolic lines, as well as convex and concave regions, are difficult to recognise, and can only be approximated. Because of the nature of contour lines, contour maps have poor sensitivity to small changes in the interrogated surface. The user is also responsible for selecting good interval between each plane, in order to get the best results. Finally, contour lines are only able to distinguish C^0 continuity between surface patches [10]. In short, this indicates that contour lines are able to show the overall shape of the surface, but are not able to provide information about the details of the smoothness of the surface.

Since finding a mathematical solution for the plane intersection problem is costly to compute, a common alternative is to develop algorithms that approximate a solution to the intersection problem. Numerous solutions work by dividing the surface into small subpatches, after which a search is done to find the subpatches that intersect the plane. Finally, a line is drawn between these patches to form the contour line. Examples of algorithms that employ this approximation method can be found in [21], [24] and [29]. Lee and Fredericks [18] propose a different algorithm that relies on solving a series of rewritten equations in order to calculate the intersection curves. The algorithm requires the surface to be defined by a specific mathematical formula, and employs a rewritten equation in order to calculate tangents to the surface, which it can use to estimate the intersection points. This requires the surface to be defined in a special way, but the equations can be solved efficiently by the computer. All in all, the mathematical solution for the intersection problem for each contour plane can be costly to compute. However, approximate solutions provide reasonable results in reasonable time.

3.2 Isophotes

As discussed in Section 2, isophotes are curves connecting points of equal light intensity on a surface. Isophotes are C^{k-1}-continuous when

the surface is C^k-continuous, which makes them very suitable for detecting the continuity order between patches of a compound surface [10]. When looking at Figure 4c and Figure 6 we can observe that the isophotes are only drawn near the fabricated irregularities and not at the planar parts of the surface. This is because isophotes do not exist on plane surfaces.
To illustrate that isophotes can be used to detect the continuity order between patches of a compound surface, we take a closer look at the isophotes in Figure 6. The isophotes are continuous at the patch boundaries, but their first derivatives are not, as seen by the abrupt changes in direction between patches. Because of this the isophotes are C^0-continuous. This indicates that the base surface is at most C^1-continuous.
Using isophotes to perform surface interrogation should be done with care because an ill-conditioned light direction can lead to properties of the isophotes not being recognized [12]. For example a C^0-continuous isophote can be mistaken to be C^1-continuous. This is compounded by the fact that isophotes are view-point independent. To alleviate this problem, the view-point can be used as light direction, causing the isophotes to update when the view-point is moved [3, 4]. Another potential disadvantage of using isophotes is that the user has to specify for how many values of c Equation (1) has to be evaluated. This introduces the problem that we can test a high number of values, but still not be certain that there is a discontinuity in an unexplored value [12].
In modern applications such as Autodesk VRED [3], isophotes (as well as highlight and reflection lines) are implemented using ray tracing. In [12] and [19] algorithms are discussed using surface point sampling. The run-times of point sampling based methods are discussed in [10] and [19], and range from 10 to 150 seconds on extremely outdated hardware. As of writing this paper, point sampling based methods, as well as the ray tracing method, can be run in real time.

3.3 Highlight lines

Highlight lines make use of light reflections in order to show information about the surface. Since reflections are intuitive to understand for humans, highlight lines are an effective method for detecting irregularities in surfaces. They provide an intuitive view for the user [6]. The reflection of these irregularities on the surface do not correspond to the expected reflections if the surface was flat, thus showing a visual indication to the user of a surface irregularity. In Figure 4d the exact position of the surface irregularity is easy to identify, since all highlight lines converge to that point, as well as give an indication of the surface features surrounding the irregularity.
Highlight lines are calculated independent of the users viewpoint, and are only sensitive to the position of the light source. This enables the user to view the surface interactively without requiring to recalculate the highlight lines. The user is able to move the light source in order to analyse the surface by looking at how the highlight lines change [28]. The smoothness, coherence and transitional pattern of the collection of highlight lines can be used as indicators to show how smooth and continuous the interrogated surface is [28]. Because of the use of normals in the calculation of highlight lines, highlight line models are highly sensitive to small changes in the surface. Finally, highlight lines are C^{k-1}-continuous when the surface is C^k-continuous [15].

The original algorithm for generating highlight lines is given by Beier and Chen [6]. Utilising an efficient traced contouring technique, they developed a robust and fully automatic algorithm that is suited for real-time calculations. Other methods have been proposed that use highlight lines in order to smooth out surfaces. Non-uniform rational B-spline (NURBS) surfaces are commonly used for this, where NURBS is a mathematical model to generate and represent curves and surfaces in computer graphics. Zhang and Cheng [28] developed a method that allows a user to remove local irregularities of NURBS surfaces by modifying highlight lines by using control points on said surface. Gyurecz [11] expanded on this. He used the method created by Zhang and Cheng [28], and applied a genetic algorithm to find solutions for the placement of the control points after a highlight line has been moved.
The calculation of highlight lines can be done quite efficiently and in real time. Properties of highlight lines make them suitable to use in methods for removing irregularities of a surface.

3.4 Reflection lines

Reflection lines simulate a real-world method used by automobile-stylists to see if the shape of a car is acceptable: the car (model) is placed in a room with parallel lines of fluorescent lamps on the ceiling. The reflection lines of these lamps as seen on the (polished) car surface are a good indication of the smoothness of the surface [17, 27]. Because reflection lines simulate the real-world, they are intuitive to understand. Simulation of reflection lines involves calculating the solution of a set of non-linear equations given by Equation (6). This calculation has to be done for the unknown parameters u and v of each reflection point to be determined.
Unlike highlight lines and isophotes, reflection lines are view-point dependent. This can be disadvantageous because the existence of a solution to the aforementioned non-linear equations is not guaranteed for all view-points [17].
Reflection lines are, just like isophotes and highlight lines, dependent on the normal vector, which in turn depends on the first derivatives of the surface. Because of this, reflection lines have one order less continuity than the considered surface [15, 25, 26]. This can also be observed in Figure 6, by using the same reasoning as for the isophotes.
Methods to calculate reflection lines evolved from numerical integration of the differential Equation (6) [17], to using a method presented in [14] that defines reflection lines using a family of splines derived from the intersection curves of a surface and a family of planes. This reduces the amount of needed computations. In [1], the computational complexity is further improved by using iso-parametric lines [12, 20] instead of the aforementioned family of splines. As of writing, CAD systems are powerful enough to use real-time ray tracing [3] to simulate reflection lines.

3.5 Direct comparison between interrogation methods

To directly compare the different methods we consider four core properties: the maximum observable surface continuity, sensitivity to changes, whether they are view-point dependent and the intuitiveness of each method. In Table 1 these properties are listed for each method. Subsequently we provide a brief explanation of each property.

Table 1: Comparison of surface interrogation methods

Method	Maximum continuity[1]	Sensitivity to changes	View-point dependent	Intuitiveness
Contour lines	C^0	Poor	No	High
Isophotes	C^1	High	No	Low
Highlight lines	C^1	High	No	Medium
Reflection lines	C^1	High	Yes	High

[1] Abbreviation for the maximum observable surface continuity

3.5.1 Maximum observable surface continuity

Since surface smoothness is a key aspect in surface interrogation, the first property we compare each method on is the maximum observable surface continuity. We define the maximum observable surface continuity as the highest order of continuity that can be determined using a specific method. The maximum observable surface continuity implies that all lower-order continuities can also be determined.
For contour lines the maximum observable surface continuity is C^0 because only discontinuous curves can be discovered. The maximum observable surface continuity of isophotes, highlight lines and reflection lines is C^1, because these lines are C^{k-1} continuous if the surface is C^k continuous. In theory the maximum observable surface continuity of these methods is only limited by the maximum continuity a human 'interrogator' can perceive, but we assume that a human can not distinguish a C^1-continuous line from a C^2-continuous line.

3.5.2 Sensitivity to changes

Because isophotes, highlight lines and reflection lines are all dependent on the surface normals they are very sensitive to minute changes in the surface. This is a useful property because it allows us to detect irregularities that might otherwise be missed. Contour lines are not suitable for spotting these small irregularities, as they have a poor sensitivity to changes in the surface.

3.5.3 View-point dependent

The convenience of a method being view-point dependent can depend on the situation. For non view-point dependent methods the user is able to analyse the lines from different view-points. This can be used to show irregularities, or provide different views of a surface that can aid the process of analysing the surface. Nonetheless, the view-point dependent property of reflection lines cannot be regarded as a negative. Moving the object, and thus changing the reflection lines, can provide the user with an interactive way to spot surface irregularities.

3.5.4 Intuitiveness

We define intuitiveness in the context of this paper as "the ease with which the information provided by the surface interrogation method is interpreted and understood". We are aware that this is a subjective measurement, however there certainly is a difference in the intuitiveness of the methods, and therefore they can be compared on this property.
The analysed methods show several levels of intuitiveness, ranging from high to low. Contour lines and highlight lines provide the user with a high level of intuitiveness. The shape of a surface can be easily identified by looking at the contour lines, and important points such as minima and maxima can be distinguished quickly. Since the reflection of light is an intuitive property of light for most humans, the use of reflection lines provides a natural way of analysing a surface. Highlight lines provide a part of this functionality. However, the fact that highlight lines remain stationary while analysing can make them hard to grasp. Isophotes provide the least amount of intuitiveness. Since these lines do not model a phenomenon in real life, they can be hard to understand. Even though contour lines also do not model a real life phenomenon the information they provide is still intuitive to understand for the user.

4 Applications and recommendations

Each of the surface interrogation methods discussed in this survey has its own unique properties. These properties make some methods more suitable for certain tasks and situations than others. For each method we provide existing applications, and recommendations of when these methods are best applied.

Contour lines are able to give a broad and intuitive overview of the shape of a surface. This method is well-suited for applications such as geography, providing an intuitive view of a surface and detecting special points on a surface such as maxima and minima. Contour maps have also been successfully used for designing large smooth surfaces, such as with ship hulls [24], plane fuselages and turbine blades [20]. We recommend contour lines to be used in situations where an intuitive broad overview is required, as well as situations where large models are broadly designed and reviewed.

Uses of isophotes range from the design of car surfaces [22] to designing and fairing ship surfaces [23], and they are even used to inspect the shape and size of galaxies [7]. Furthermore, the special case of $c = 0$ in Equation (1) can be used to construct silhouette curves [16, 19]. We recommend to use isophotes when the primary purpose is to inspect curved surfaces for small irregularities. When using isophotes we recommend to try different light directions, and various rotations of the surface to be inspected. This process can be eased by using the view-point as light direction [4, 3].

The properties of highlight lines make them excellent for the interactive evaluation of the smoothness of a surface [28]. This method is highly sensitive to changes in the flow of a surface, thus it can be effectively used to spot surface irregularities. Since highlight lines are view-independent, the user can rotate the surface to spot slight deviations in the generated highlight lines. The ability to show these slight deviations makes this method a viable option for the design of fully smooth surfaces, as can be found in car design and airplane design. Another interesting application of highlight lines is to use them as a tool for fixing surface irregularities, as methods proposed by Gyurecz [11] and Zhang & Cheng [28] show. We recommend the use of highlight lines for situations that require a fine analysis of a surface, where view-independent and intuitive representations are preferred.

Reflection lines originated from the automotive industry [14, 17] and were used even before the computer era [15]. Because of these origins, reflection lines are still primarily used in the process of designing cars [27]. However, the ship-design industry also uses reflection lines for surface interrogation [23]. Reflection lines can not only be used to inspect surfaces, but also to correct them [14, 17]. Our recommendation for the use of reflection lines is to use them as an intuitive method to discover small surface irregularities, when view-dependent behaviour is desired. Their use is strongly recommended when evaluating car surfaces.

5 Conclusion

In this paper we have explored four different methods used for surface interrogation: contour lines, isophotes, highlight lines and reflection lines. First we defined each method using theory from the existing literature. Then we further explored the properties of each method and compared them based on the maximum observable surface continuity, sensitivity to changes, view-point dependency and intuitiveness. We also incorporated visual examples for each method to aid the understanding of the reader. Finally, we provided applications for each method, as well as recommendations on when each method is best applied. Our recommendations for each method are as follows:

- Contour lines should be used for situations where an intuitive and broad overview is required.
- Isophotes should be used when the primary purpose is to inspect curved surfaces for small irregularities. When using isophotes we recommend to try various light directions, and various rotations of the surface or shape to be inspected.
- Highlight lines should be used in situations that require a fine analysis of a surface, where view-independent and intuitive representations are preferred.
- Reflection lines should be used as an intuitive method to discover small surface irregularities, when view-dependent behaviour is desired. Their use is strongly recommended when evaluating car surfaces.

The goal for this survey is to act as a comprehensive resource on the major surface interrogations methods. It is of significance because it clearly delineates the different methods and provides clear recommendations on their use and applications.

5.1 Future work

As for future work, we recommend to focus further research on developing the isophotes and reflection line methods, since these methods are the most effective in finding surface irregularities and discontinuities. Although current hardware is able to calculate these lines in a reasonable time, further improvements of these algorithms could enable real-time interrogation of larger surfaces, or a faster analysis process.
Additional research in isophotes and reflection lines might also lead to discoveries about features and interesting properties of these lines. These properties could be used as a base for new algorithms. A concrete example of this is mentioned in Section 3.3 and concerns highlight lines being applied in automatic surface smoothing algorithms.

Finally, future work could compare a wider selection of surface interrogation methods. Examples of methods not included in this paper are surface curvatures, curvature plots and focal surfaces. Surface curvatures are an example of a surface interrogation method that does not utilize lines, but instead uses a colour mapping of the surface.

ACKNOWLEDGEMENTS

We would like to thank our expert reviewer Jiří Kosinka and our colleagues for their valuable feedback.

REFERENCES

[1] S. Abrams, L. Bardis, C. Chryssostomidis, N. Patrikalakis, S. Tuohy, F.-E. Wolter, and J. Zhou. The geometric modeling and interrogation system praxiteles. *Journal of Ship Production*, 11:116–131, 01 1995.

[2] G. Aouad, S. Wu, A. Lee, and T. Onyenobi. *Computer aided design guide for architecture, engineering and construction*. Routledge, 2013.

[3] Autodesk. 3D Visualization Software — VRED — Autodesk. `https://www.autodesk.com/products/vred/overview`.

[4] Autodesk. Alias — Industrial Design & Product Design Software — Autodesk. `https://www.autodesk.com/products/alias-products/overview`.

[5] J. M. Beck, R. T. Farouki, and J. K. Hinds. Surface analysis methods. *IEEE Computer Graphics and Applications*, 6(12):18–36, 1986.

[6] K.-P. Beier and Y. Chen. Highlight-line algorithm for realtime surface-quality assessment. *Computer-Aided Design*, 26(4):268–277, 1994.

[7] J. Binney, J. Binney, M. Michael, and M. Merrifield. *Galactic astronomy*, volume 9. Princeton University Press, 1998.

[8] R. T. Farouki. The characterization of parametric surface sections. *Computer Vision, Graphics, and Image Processing*, 33(2):209–236, 1986.

[9] A. R. Forrest. On the rendering of surfaces. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 253–259, 1979.

[10] N. Guid, Č. Oblonšek, and B. Žalik. Surface interrogation methods. *Computers & graphics*, 19(4):557–574, 1995.

[11] G. Gyurecz. Removing local surface irregularities by modifying highlight lines. In *2007 International Symposium on Logistics and Industrial Informatics*, pages 133–136. IEEE, 2007.

[12] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, B. Wördenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. *IEEE Computer Graphics and Applications*, 12:53–60, 1992.

[13] S. Hahmann. Visualization techniques for surface analysis. In C. Bajaj, editor, *Data visualization techniques*, pages 44–74. John Wiley, 1999.

[14] E. Kaufmann and R. Klass. Smoothing surfaces using reflection lines for families of splines. *Computer-Aided Design*, 20(6):312 – 316, 1988.

[15] P. Kiciak. Geometric continuity of curves and surfaces. *Synthesis Lectures on Visual Computing: Computer Graphics, Animation, Computational Photography, and Imaging*, 8(3):1–249, 2016.

[16] K.-J. Kim and I.-K. Lee. Computing isophotos of surface of revolution and canal surface. *Computer-Aided Design*, 35(3):215 – 223, 2003.

[17] R. Klass. Correction of local surface irregularities using reflection lines. *Computer-Aided Design*, 12(2):73–77, 1980.

[18] R. B. Lee and D. A. Fredricks. Intersection of parametric surfaces and a plane. *IEEE Computer Graphics and Applications*, 4(8):48–51, 1984.

[19] A. Lennings, J. Peters, and J. Vergeest. An efficient integration of algorithms to evaluate the quality of freeform surfaces. *Computers & Graphics*, 19(6):861 – 872, 1995.

[20] N. M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[21] C. S. Petersen. Adaptive contouring of three-dimensional surfaces. *Computer Aided Geometric Design*, 1(1):61–74, 1984.

[22] T. Poeschl. Detecting surface irregularities using isophotes. *Computer Aided Geometric Design*, 1(2):163 – 168, 1984.

[23] N. S. Sapidis. *Designing fair curves and surfaces: Shape quality in geometric modeling and computer-aided design*. SIAM, 1994.

[24] S. G. Satterfield and D. F. Rogers. A procedure for generating contour lines from a B-spline surface. In *Frontiers in Computer Graphics*, pages 66–73. Springer, 1985.

[25] G. Sußner, G. Greiner, and S. Augustiniack. Interactive examination of surface quality on car bodies. *Computer-Aided Design*, 36(5):425 – 436, 2004.

[26] H. Theisel. Are isophotes and reflection lines the same? *Computer aided geometric design*, 18(7):711–722, 2001.

[27] D. E. Ulmet. Customized reflection lines for surface interrogation in car body design. In *Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2007)*, pages 124–129, Sep. 2007.

[28] C. Zhang and F. F. Cheng. Removing local irregularities of NURBS surfaces by modifying highlight lines. *Computer-Aided Design*, 30(12):923–930, 1998.

[29] J. Zheng and C. Millham. A linear pivoting method for detecting and tracing planar section curves of free-form surfaces. *Computers & graphics*, 16(4):411–420, 1992.

An Analysis on Code Smell Detection Tools

Anil P. Mathew, Filipe A. R. Capela

Abstract—Software correctness is considered one of the principal aspects of software maintenance and evolution. A well-known issue that deteriorates software stability is the presence of code smells in the system. A code smell can be defined as any dangerous practice found in the source code of a software; these smells impact systems in the long term, thus negatively affecting their maintainability. To mitigate this, detecting smells in the system at an early stage can reduce technical debt, allowing companies to reduce monetary costs. Multiple code smell detection tools have been developed to detect code abnormalities in a system. These tools can operate in one or more programming languages, primarily focusing on the Java environment.
For our analysis, we start by introducing the definition of what a code smell is, the segregation of these, and a brief explanation of the most occurring code smells. Following that, we introduced the concept of technical debt, demonstrating its significance as a metric for software evaluation. Subsequently, we conduct a comparative study to understand which are the most reliable code smell detection tools to enhance software correctness. We started by gathering the most popular tools available and considering parameters like the tool's precision and recall, the languages they support, their usability, and the type of smells they detect.
Alongside our analysis, we review the results given in published papers on the topic, to measure the quality of the tools when applied to Open-source projects and study their effects on the parameters mentioned previously as well as to understand how some tools indicate different measures of technical debt, using different indexes.

Index Terms—Code smells; Technical Debt; Code smell detection tools; Software correctness

1 Introduction

Bad practices found on a piece of software, that can range from problems in the system's architecture, internal design and source code are defined as smells, namely architectural smells, design smells, and code smells, respectively. In this paper we will be focusing on the last category, code smells.

Code smells affect the system in the long term, which means that the software may not reach it's optimal state in terms of performance or maintainability. The increasing maintenance cost of software is commonly associated with poor software quality and improper coding standards. The maintainability of software is crucial since it is one of the factors influencing the cost of future development activities. In a report from CISQ (Consortium for IT Software Quality), the cost of substandard quality software in the United States in 2018 was around $2.84 trillion, with 50% of it being assigned to software malfunction and bugs [1].

The software engineering community has performed extensive research on identifying code smells. The first attempt was made in the late 1990s by Fowler & Beck [2], demonstrating that the problem of maintaining software and refactoring it has been affecting software companies from the inception of software development. Several tools have been developed to identify and point out solutions towards code smells before a system is deemed as production-ready. Therefore, companies and research groups have developed code smell detection tools, which perform automated analysis. These rely on different detection strategies, such as metric-based and visualization support. The aim of these code smell detection tools is to allow developers to refactor their code, saving them time and money, which can then be employed into developing new features.

There exist a plethora of code smell detection tools, and developers find it challenging to decide which proves more beneficial to their needs. This paper demonstrates the current state of the problem with regards to how code smell detection tools can prove to be beneficial for all parties involved. This allows developers to obtain an overall insight on which tool would be best suited for them based on the selected software, programming language, and environment. Note that in our paper, we will be restricting our analysis to tools that are able to scan Java source code.

We combine past papers on the matter, and incorporate their research with some research of our own, to answer two questions: *What are the differences between code smell detection tools in terms of their benefits, limitations, and missing features?* and *How much is technical debt impacted by code smells and how can these be reduced?*

We start off in Section 2 where we present the definitions of code smells, their categories, and the most reoccurring code smells in software systems. In Section 3 an explanation of the workflow of the code smell detection tools is presented, as well as an highlight on what are the main differences among the selected tools. In Section 4, we will be providing a brief overview of the concept of technical debt and what this metric can provide to software developers, as well as how this metric can be mitigated with the usage of code smell detection tools. In Section 5 we present the three researches which serve as an inspiration for our paper, by highlighting their most significant remarks regarding the topic of code smell detection tools and how these are beneficial for systems. Additionally, we provide our small contribution to the topic by analyzing two code smell detection tools. To finish our paper, in Section 6, conclusions are drawn which aim to answer the two questions presented before, and finally in Section 7 we debate on the future work on this matter.

2 Code Smells

Martin Fowler, in his book "Refactoring: improving the design of existing code" [2], has cataloged twenty-two symptoms in the code to be used as indicators for harmful practices and implementations. Other papers [3][4][5] reach a consensus that code smells are laborious to detect, which deteriorates the software and makes it hard to maintain in the long term. Combining all the observed attempts at defining code smells mentioned above, we provide our definition of code smells as being symptoms in the source code. Thus, indicating that there are some issues with the architecting and programming standard followed when developing the software. The presence of code smells does not imply that the software will not compile or give the expected results. Instead, quality attributes like performance, reliability, and maintainability will be severely impacted, which results in poor code quality.

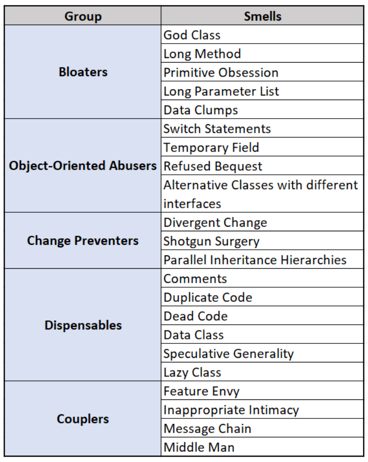
• *Anil P. Mathew is a MSc Computing Science student at the University of Groningen, E-mail: a.palayiparambil.mathew@student.rug.nl*
• *Filipe A.R. Capela is a MSc Computing Science student at the University of Groningen, E-mail: f.a.de.capela@student.rug.nl*

Manuscript submitted on 23 March 2020. For information on how to obtain this document, please contact Student Colloquium of FSE, University of Groningen.

2.1 Categories

The twenty-two code smells that have been defined by Martin Fowler can be classified into five different categories [2], with Figure 1 providing a high-level categorization of the code smells:

- **Bloaters**: Classes, methods, or code that has grown excessively over a period, marking the system complex to manage. Typically, bloaters do not happen immediately but accumulate as the codebase grows.
- **Object-Oriented Abusers**: This comprises of code smells in which object-oriented programming principles are not applied to code accurately.
- **Change Preventers**: Modifying code in one place will require adjustments in other sections where the code is dependent, hence making the development process laborious.
- **Dispensables**: This consists of code that is irrelevant, and its exclusion makes the code more coherent.
- **Couplers**: Code structures that have high coupling between classes and methods appear in this category of code smell.



Group	Smells
Bloaters	God Class
	Long Method
	Primitive Obsession
	Long Parameter List
	Data Clumps
Object-Oriented Abusers	Switch Statements
	Temporary Field
	Refused Bequest
	Alternative Classes with different interfaces
Change Preventers	Divergent Change
	Shotgun Surgery
	Parallel Inheritance Hierarchies
Dispensables	Comments
	Duplicate Code
	Dead Code
	Data Class
	Speculative Generality
	Lazy Class
Couplers	Feature Envy
	Inappropriate Intimacy
	Message Chain
	Middle Man

Fig. 1. Code Smells Categories

2.2 Frequently occurring code smells

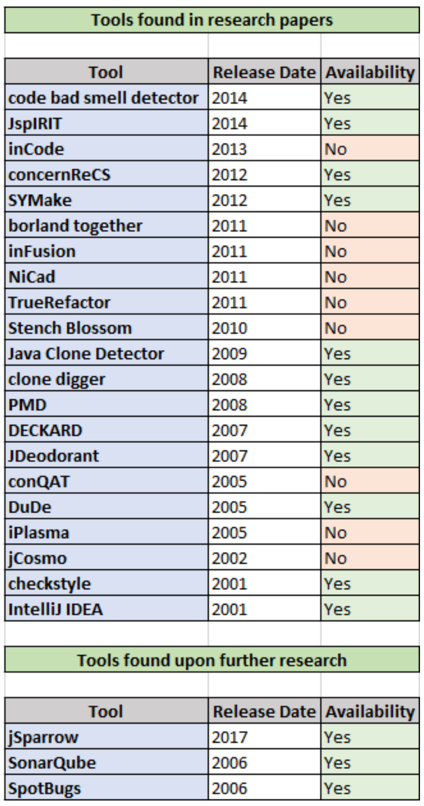
For this study, we decided to provide a high-level explanation of the five most reoccurring code smells: *God Class, Long Method, Duplicate Code, Feature Envy, and Data Class* [4]. These smells were selected due to the following reasons: Commonly found in the analyzed Open-source projects, recurrence in the systems from the analyzed studies and its association with software degradation symptoms[6]. Here is a brief description of the selected code smells:

- **God Class**: A type of code smell in which classes tend to centralize the essential working of a system in one class. It does much of its work by delegating more straightforward tasks to other classes and consuming a considerable amount of data from dependent classes. [7]
- **Long Method**: This type of smell is associated with methods that have many lines of code. Therefore, impacting the readability of the method. Programs prevail longer when the methods are kept compact. This smell violates the "Single Responsibility Principle" [8]. [Alternative name: God Method [9]]
- **Duplicate Code**: A duplicate code smell is a result of copy-pasted code, and hence it requires modifications to all copies of the code in case the logic needs adjustment. Furthermore, this can finally result in inconsistency issues. [Alternative name : Clones, Code clones [9]]
- **Feature Envy**: This smell describes the situation in which an object gets the fields of another object in order to perform computations, rather than asking the object to do the computations itself. [10]
- **Data class**: These are classes that comprise fields and raw methods, which are accessed by other classes. These classes do not offer any additional functionality and cannot operate autonomously. [11]

3 Code Smell Detection Tools

Code smells can be detected manually when the application is trivial, but when it grows in size and complexity some smells may go unnoticed. Hence, it would require an experienced user's in-depth knowledge of the system to detect these. In order to improve software and prevent smells from propagating and impacting the entire system, it is necessary to analyze it using the proper tools. These are referenced as code smell detection tools, as their purpose is to scan the software and detect potential code smells. There are countless tools available online, they vary in the code smells they detect, their execution method (plug-in or standalone), the supported programming languages, and even in the names they define for specific code smells [4]. In our analysis, we focus on analyzing the most used tools and making comparisons over which are the best regarding precision and recall.

Java has proven to be the programming language that has the most detection tools available to analyze source code [4]. In order to keep our analysis compact, we refine our analysis to tools that operate under this constraint. The next step was to analyze the list of code smell detection tools found in [4] and understand if these are still operational or relevant. Additionally, we added more code smell detection tools, which were gathered from our own experience - jSparrow [12], SonarQube [13] and SpotBugs [14]. The result of this analysis gives us a refined list of code smell detection tools found in Figure 2.



Tools found in research papers

Tool	Release Date	Availability
code bad smell detector	2014	Yes
JspIRIT	2014	Yes
inCode	2013	No
concernReCS	2012	Yes
SYMake	2012	Yes
borland together	2011	No
inFusion	2011	No
NiCad	2011	No
TrueRefactor	2011	No
Stench Blossom	2010	No
Java Clone Detector	2009	Yes
clone digger	2008	Yes
PMD	2008	Yes
DECKARD	2007	Yes
JDeodorant	2007	Yes
conQAT	2005	No
DuDe	2005	Yes
iPlasma	2005	No
jCosmo	2002	No
checkstyle	2001	Yes
IntelliJ IDEA	2001	Yes

Tools found upon further research

Tool	Release Date	Availability
jSparrow	2017	Yes
SonarQube	2006	Yes
SpotBugs	2006	Yes

Fig. 2. List of Code Smell Detection Tools

3.1 Tools Workflow

The tools which are mentioned in Figure 2 operate using a workflow that can be seen in Figure 3. This process commences when the tools scan the source code or the compiled byte code. Then detection strategies are applied to it, and finally, it outputs the detected code smells. The detection strategies are what differs between the tools and can be based on metrics, trees, textual analysis, program dependence graphs or token analysis [4].

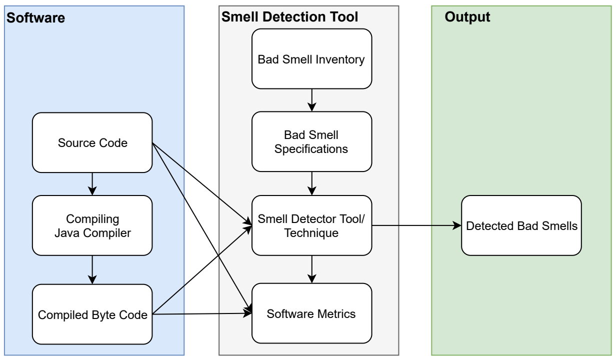
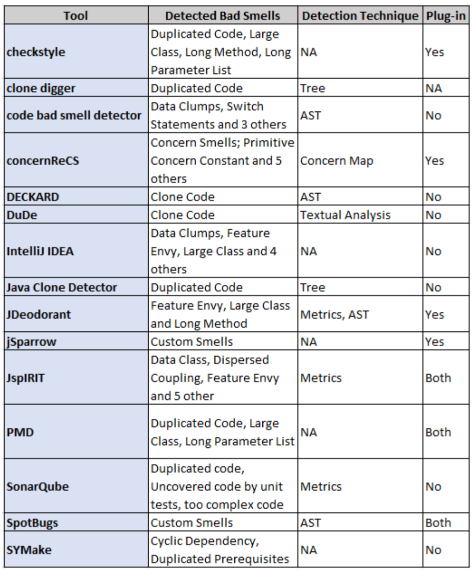


Fig. 3. Detection Process of Tools

3.2 Main differences

Different tools look for different code smells. With this paper we will demonstrate that the tools which were gathered during the refinement process shown previously in Figure 2 have different code smells repositories. Based on the analyzed studies performed in [4][5], we created a comparison of the different tools which is depicted in Figure 4. The aim of this comparison was to identify the code smells detected by each tool, the employed technique, and whether the tool is a plug-in or not. In this comparison it is shown that most of the tools employ a detection technique based on AST[1]. It is also noticed that the majority of the tools are offered as standalone applications. An important fact is that more than half of the tools look for the "Duplicated Code" and "Large Class" code smells. This corroborates the statements made in Section 2.2, where we claim that these are some of the most frequent smells detected in software.



Tool	Detected Bad Smells	Detection Technique	Plug-in
checkstyle	Duplicated Code, Large Class, Long Method, Long Parameter List	NA	Yes
clone digger	Duplicated Code	Tree	NA
code bad smell detector	Data Clumps, Switch Statements and 3 others	AST	No
concernReCS	Concern Smells; Primitive Concern Constant and 5 others	Concern Map	Yes
DECKARD	Clone Code	AST	No
DuDe	Clone Code	Textual Analysis	No
IntelliJ IDEA	Data Clumps, Feature Envy, Large Class and 4 others	NA	No
Java Clone Detector	Duplicated Code	Tree	No
JDeodorant	Feature Envy, Large Class and Long Method	Metrics, AST	Yes
jSparrow	Custom Smells	NA	Yes
JspIRIT	Data Class, Dispersed Coupling, Feature Envy and 5 other	Metrics	Both
PMD	Duplicated Code, Large Class, Long Parameter List	NA	Both
SonarQube	Duplicated code, Uncovered code by unit tests, too complex code	Metrics	No
SpotBugs	Custom Smells	AST	Both
SYMake	Cyclic Dependency, Duplicated Prerequisites	NA	No

Fig. 4. Comparison of the tools

[1]https://www.techopedia.com/definition/22431/abstract-syntax-tree-ast

4 Technical Debt

Technical debt is defined as a measurement of how much software needs refactoring. The first attempt at defining it was in 1992, on a paper by Ward Cunningham [15], where he describes technical debt as when software development compromises in maintainability to increase short term productivity. In Figure 5 it is possible to see the reasons for high technical debt in a system. When there is pressure to increase productivity, technical debt increases, leading to reduced software quality and motivation. Technical debt has no defined unit, varying depending on the tool which provides this measurement, either outputting cost, time, or Lines of Code (LOC). This metric incorporates code smells to provide an overview of the software state and provides developers insightful information on where the majority of problems lie in the system. It also provides an estimate of how many resources will need to be consumed in order to patch the existing issues.

Different components are used by tools in order to compute the values for technical debt. These range from *Code Smells*, *High Complexity of components*, *Bugs*, *Improper style of coding*, *Increased loading times*, *Performance issues* as well as *Software age*.

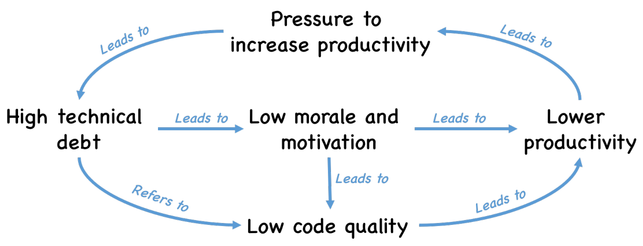


Fig. 5. Technical Debt Lifecycle [16]

4.1 Impact on Software

High technical debt indicates that the software contains problems regarding maintainability and performance. Factors like accumulated debt, type of debt, and the team involved in the development will impact the software's capability to have new features, since it will be difficult for developers to comprehend the source code.

As mentioned in the introductory section, trillions of dollars are put into refactoring. From an enterprise point of view, it is easier to spend these resources on refactoring an old system, than making the decision to shift to a brand new one. This decision is taken because starting from scratch has a substantial effort associated with it. It is impossible to quantify how many hours have been put into fixing and refactoring systems, but if these were to be developed correctly in the first place, most of that time could have been employed into developing new features, new systems, and overall creating better programming environments.

The reason why technical debt occurs so much in systems comes from the fact that developers are always under strict deadlines. This forces developers to opt for shortcuts as they tend to follow bad coding practices, and hence degrading the performance of the systems. With this, if the software is created inefficiently, then it will impact build time, as it contains too many lines of code. One phenomenon that can occur is the famous "spaghetti code"[2], which makes code comprehension difficult. This damages the productivity of the teams, which can consequently lead to some frustration, creating a snowball effect that can lead to software reaching the end of their lifecycle.

4.2 How to cope with Technical Debt

As most companies are aware of the concept of technical debt, it should be of their interest to find ways to mitigate it so that it does not lead to the worst-case scenario, company closedown. In Figure 6

[2]https://sourcemaking.com/antipatterns/spaghetti-code

it is possible to understand how technical debt can be managed during the course of the system's lifecycle.

In our opinion, the most important thing companies should aim at is to have small iterations, where after each one of them, a specialized team would intervene and refactor the smelly code, managing the system's technical debt. Another way to decrease technical debt is for the software to be accessible to the Open-source community, hence improving based on the input from independent developers, reducing the burden of the DevOps [3] teams in fixing issues.

As mentioned, we believe that if developers stick to good coding practices in the first place, the debt is much more manageable, and not a lot of effort needs to be put into refactoring and analyzing the source code. Unfortunately, this is near impossible since it is tough to make code which is understandable while following time-sensitive deadlines.

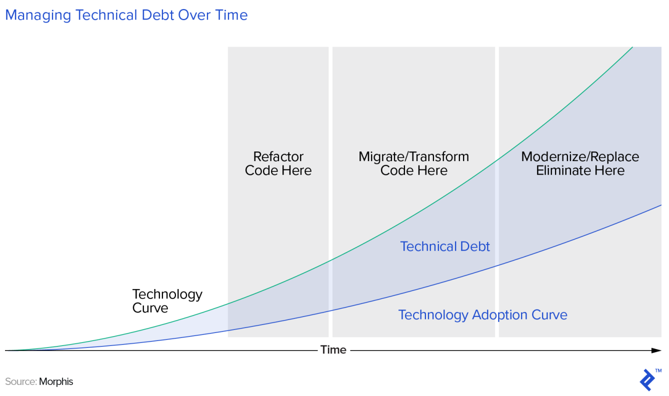


Fig. 6. Technical Debt Roadmap [17]

4.3 Code Smell Detection Tools effect on Technical Debt

Technical debt can be drastically reduced if tackled at the right time. The first place where technical debt arises is during the development phase, where code is involved. It is possible to see from Figure 7, the later the smells are detected, the harder it will be to mitigate technical debt, since the complexity of this task increases linearly or even exponentially. Code smell detection tools play a crucial role when it comes to reducing this technical debt. If applied during the "Code" stage, the costs for refactoring will become lesser than when compared to phases such as "Architecture", "Process", "Organization" and "Final Product".

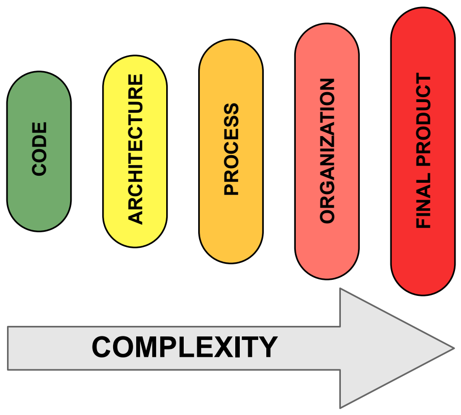


Fig. 7. Technical Debt accumulation over iterations [18]

[3]https://www.atlassian.com/devops

5 Discussion

As stated in the introduction, software correctness has been researched by various developers, right from the origin of software development. Our study will be based on three different studies, which focus on the following issues:

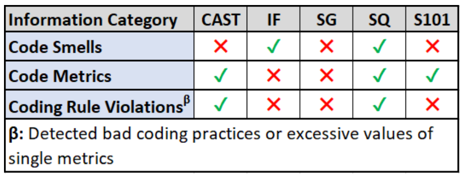
- Indexes of technical debt [3] (Section 5.1).
- Comparative studies on several code smell detection tools [4] [5] (Sections 5.2 and 5.3, respectively).

The first study [3] focuses on describing how the technical debt of a software is calculated, using five different tools, CAST, inFusion, SonarGraph, SonarQube, and Structure101. The other two studies [4] [5] focus directly on using different code smell detection tools and understanding which of them are the most suitable for different contexts of analysis. We intend to analyze the approaches mentioned above and provide our insights based on our experience with using code smell detection tools and combining it with the outcome of these studies.

5.1 Technical Debt indexes provided by tools [3]

Francesca Fontana et al. (2016) discussed different tools which intend to provide the users with a measurement of the technical debt of their systems. This study is not a hands-on experiment, but rather it grasps the theory behind how the tools obtain their results for technical debt. They analyze different tools, but we will be focusing on how issues with code impact technical debt.

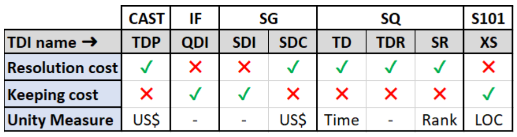
- Five tools are introduced, CAST (CAST), inFusion (IF), SonarGraph (SG), SonarQube (SQ), and Structure101 (S101).
- This study presents the formulas each of the tools utilize to display technical debt, and shows how each tool uses different indexes to calculate it.
- The main difference between the tools reside on which information they utilize to calculate the technical debt, as seen in Figure 8, where it is possible to see that these may utilize "Code Smells", "Code Metrics" and "Coding Rule Violations".
- The index provided by each tool is presented in Figure 9, where Technical Debt Index is represented by the acronym TDI. Here it is possible to see whether the tools provide the resolution costs and the keeping costs, as well as displaying which unit of measurement they output for technical debt.
- An example is CAST, which uses TDP (Technical Depth-Principal) while SonarGraph, uses not one but rather two indexes: SDI (Structural Debt Index) and SDC (Structural Debt Cost).
- SonarQube seems to be the tool that uses the most information regarding code problems, making it the most reliable tool. Other results indicate that CAST and SonarGraph opts for using cost (US$) as their unit for measuring a system's technical debt. On the other hand, tools either use Time, Rank, or LOC to indicate how much effort needs to be put into the software to fix it.



Information Category	CAST	IF	SG	SQ	S101
Code Smells	✗	✓	✗	✓	✗
Code Metrics	✓	✗	✗	✓	✓
Coding Rule Violations$^{\beta}$	✓	✗	✗	✓	✗

β: Detected bad coding practices or excessive values of single metrics

Fig. 8. Input information used by tools [3]



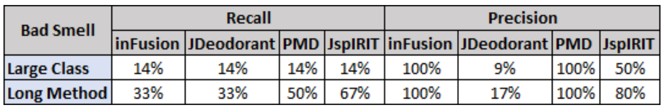
	CAST	IF	SG		SQ			S101
TDI name ➔	TDP	QDI	SDI	SDC	TD	TDR	SR	XS
Resolution cost	✓	✗	✗	✓	✓	✓	✓	✗
Keeping cost	✗	✓	✓	✗	✗	✗	✗	✓
Unity Measure	US$	-	-	US$	Time	-	Rank	LOC

Fig. 9. Output of the indexes provided by the tools [3]

5.2 A comparative study of bad smell detection tools [4]

Eduardo Fernandes et al. (2016) did a thorough study where they compare eighty-four bad smell detection tools. This list was then refined based on the availability of the tools and a small portion of these was used to analyze two different projects: JUnit and MobileMedia.

- The paper used a reference list containing code smells detected manually, which were provided as part of a study to compare these results with the output of the detection tools. This reference list was only made for MobileMedia software, as this was created as part of a research project.
- Since some tools do not detect the same smells, this study chose only to analyze Large Class and Long Method.
- The results obtained demonstrated that, when comparing these tools to the actual smells in the source code, some tools got better scores for precision and recall than others, as seen in Figure 10.
- PMD scored 100% on precision and had the best results for recall out of the four detection tools analyzed. Tools such as inFusion (deprecated) and JSpIRIT (deprecated) scored well in these criteria, with inFusion also standing out with values of 100% for precision, but scoring lower on the recall of both smells. On the downside, JDeodorant, which is a tool that is still around, scored very low on both precision and recall, drastically reducing its reliability.



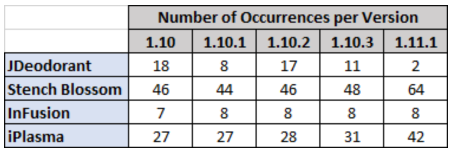
Bad Smell	Recall				Precision			
	inFusion	JDeodorant	PMD	JspIRIT	inFusion	JDeodorant	PMD	JspIRIT
Large Class	14%	14%	14%	14%	100%	9%	100%	50%
Long Method	33%	33%	50%	67%	100%	17%	100%	80%

Fig. 10. Results for Precision and Recall [4]

5.3 An experience report on using code smells detection tools [5]

Francesca Fontana et al. (2011) experimented to understand which tools would be most fit for improving software correctness. This study analyzed six different tools used for scanning source code written in the Java programming language.

- This experiment used the GanttProject source code in order to extract its conclusions, applying all six tools to detect code smells.
- One remark is that this paper analyzed five different versions of the software, to understand how the number of code smells increased throughout its development, consequently affecting its complexity.
- This paper used eight code smells, ranging from God Class (Section 2.2), Long Method (Section 2.2), and more. The procedure involved scanning the GanttProject with all the tools and display the number of occurrences of each code smell per tool.
- The results displayed are inconclusive to determine which is the best tool. However, it is possible to understand how each of the tools operates for each code smell, compared to others.
- An example of this is the Feature Envy smell which can be seen in Figure 11; Stench Blossom detected 64 occurrences, while iPlasma, inFusion, and JDeodorant detected 42, 8, and 2, respectively, for the last analyzed version of GanttProject.
- Since there is no reference table with the correct values of the number of occurrences of the code smells (usually done manually by a panel of certified analysts), it is hard to say which of the tools is more reliable, since it is hard to quantify the occurrence of false positives, which is an indicator of the precision and recall of a tool.



	Number of Occurrences per Version				
	1.10	1.10.1	1.10.2	1.10.3	1.11.1
JDeodorant	18	8	17	11	2
Stench Blossom	46	44	46	48	64
InFusion	7	8	8	8	8
iPlasma	27	27	28	31	42

Fig. 11. Results for Feature Envy smell on different tools [5]

5.4 Our insight on Code Smell Detection Tools

We analyzed the tools mentioned in the analyzed studies and decided to acquire some data regarding some up-to-date tools. We were able to identify new tools that were not covered by such papers. Two of the tools that were introduced by the group on the table in Figure 2, SonarQube, and SpotBugs will be the focus of analysis.

SonarQube is a complete tool since it presents the users with a list of existing code smells, as well as a metric for visualizing the Technical Debt of the scanned system. This software is widely used in the Open-source community, as it provides the analytics to the developers, as well as to the users. This way, they can see where bugs are present and help to correct them. This tool provides metrics for Technical Debt in the form of Time, which allows software development teams to understand where most of the debt is focused and fix it right away. SonarQube is, therefore, an iterative tool, which means that once it is deployed, it will keep tracking the software changes and continuously post the developers the state of the Technical Debt, making it an excellent option for companies to utilize.

A tool that was not discussed in any of the analyzed papers was SpotBugs, a tool that was released in 2006, under the name FindBugs, as a research project from the University of Maryland. It has since grown to become one of the most reliable tools in the community. SpotBugs is used as a plug-in or as a standalone GUI that pinpoints the locations of the code smells, allowing for an easier refactoring of the code, hence improving the time it takes to reduce the Technical Debt.

6 Conclusion

In this paper, we have covered the notion of code smells, code smell detection tools, and technical debt. Later, we combined these notions in such a way that it is clear that by using code smell detection tools to fix code smells, companies can reduce their software's technical debt.

The first question we tried to respond to was: "*What are the differences between code smell detection tools in terms of their benefits, limitations, and missing features?*". We responded by presenting results from several papers, which analyze various tools and extrapolate their best points, as well as their weak points. Different tools will only apply to specific criteria, like the programming language or whether a developer wishes to test the occurrence of a specific code smell. One remark which corroborates the statement above is the fact that it is tough to conclude whether a tool is better than another just by testing it on a limited software. This happens since these do not represent a significant part of the population, which makes it impossible to generalize. Despite not being able to come up with the most suited tool, it is safe to say that some offer more support than others, like the case of SonarQube, which is a tool which offers a vast number of metrics and can be used to analyze an extensive list of programming languages, as well as providing CI/CD (Continuous Integration and Continuous Deployment).

The second question tackled in this paper is: "*How much is technical debt impacted by code smells and how can these be reduced?*". As mentioned in Section 4, to calculate technical debt, code smells play a significant role. With the existence of code smells, it is tough for a system to withstand continuous change. To reduce technical debt, code smells are an aspect that needs urgent fixing. To do so, we have mentioned already in the first question that code smell detection tools are vital in this aspect, allowing for developers to employ the time before the system is deployed to the customers into using such tools and refactoring the code. However, this is not the only way the issue of high technical debt can be tackled. As discussed in Section 5.1, we analyzed tools that measure technical debt in the systems. These act as a secondary line of defense for developers to refactor systems. It is hard to draw conclusion on which is the most reliable tool to measure technical debt, as these do not follow a standardized protocol to measure it. Starting by how different tools give different units of measure, this already restricts software companies from using some tools while disregarding others. If they, for example, seek to obtain a time estimate over money, they might opt for SonarQube instead of SonarGraph/CAST. Therefore, depending on the desired output, some tools will be better options. One reliable option which we were able to understand from the studies is the SonarQube tool, which uses the most information to calculate the technical debt, allowing for a complete analysis of this metric.

An additional conclusion we were able to extract from this study is that code smell detection tools have the power to maintain software on a low debt when applied at the right time. If this is achieved, it allows for the software to progress over time, with its main effort being put into creating new features and optimizing it, instead of wasting time fixing problems that should have been detected at earlier stages.

7 Future Work

In the future, a recommendation would be for code smell detection tools to aim to reach a unified definition of each code smell. This way there would be no problem with different tools detecting different code smells for the same portions of code. An example is shown in Section 2.2 where the "Duplicate Code" smell has other names such as "Clones" or "Code Clones".

Since there is a wide range of domains (such as consumer goods, financial and telecommunications) for which software is developed, these can be distinguished by their coding standards, programming languages and platforms. This variability makes it challenging for developers to choose the best code smell detection tools to suit their requirements. So, as future work, researchers need to perform experiments on multiple software belonging to different domains to get some potential benchmark that would assist software developers in making a more viable choice on selecting the appropriate code smell detection tool. With this data of the code smell detection tools performance based on the domains, the developers' time/effort would be reduced drastically as their preferences are rational for the field that they plan to develop.

Acknowledgements

The authors wish to thank Prof. R. Smedinga, Prof. M. Biehl and our supervisor Jie Tan for giving us the required guidance in writing this paper.

References

[1] H. Krasner, "The cost of poor quality software in the us: A 2018 report," *Consortium for IT Software Quality, Tech. Rep.*, 2018.

[2] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[3] F. A. Fontana, R. Roveda, and M. Zanoni, "Technical debt indexes provided by tools: A preliminary discussion," in *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2016, pp. 28–31.

[4] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, "A review-based comparative study of bad smell detection tools," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–12.

[5] F. A. Fontana, E. Mariani, A. Mornioli, R. Sormani, and A. Tonello, "An experience report on using code smells detection tools," in *2011 IEEE fourth international conference on software testing, verification and validation workshops*. IEEE, 2011, pp. 450–457.

[6] R. Oliveira, L. Sousa, R. de Mello, N. Valentim, A. Lopes, T. Conte, A. Garcia, E. Oliveira, and C. Lucena, "Collaborative identification of code smells: A multi-case study," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2017, pp. 33–42.

[7] S. Vaucher, F. Khomh, N. Moha, and Y.-G. Guéhéneuc, "Tracking design smells: Lessons from a study of god classes," in *2009 16th Working Conference on Reverse Engineering*. IEEE, 2009, pp. 145–154.

[8] "Single responsibility principle," Feb 2020. [Online]. Available: https://en.wikipedia.org/wiki/Single_responsibility_principle

[9] A. S. Cairo, G. d. F. Carneiro, and M. P. Monteiro, "The impact of code smells on software bugs: A systematic literature review," *Information*, vol. 9, no. 11, p. 273, 2018.

[10] G. 4, jhewlettjhewlett 2, and R. . bronze badges, "What is a" feature envy" code and why is it considered a code smell?" Aug 1963. [Online]. Available: https://softwareengineering.stackexchange.com/questions/212128/what-is-a-feature-envy-code-and-why-is-it-considered-a-code-smell

[11] G. Dhaka and P. Singh, "An empirical investigation into code smell elimination sequences for energy efficient software," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2016, pp. 349–352.

[12] "Automatic java refactoring - fits in every infrastructure!" [Online]. Available: https://jsparrow.eu/

[13] "Your teammate for code quality and security." [Online]. Available: https://www.sonarqube.org/

[14] "Spotbugs." [Online]. Available: https://spotbugs.github.io/

[15] W. Cunningham, "The wycash portfolio management system," *SIGPLAN OOPS Mess.*, vol. 4, no. 2, p. 29–30, Dec. 1992. [Online]. Available: https://doi.org/10.1145/157710.157715

[16] T. Sharma, "Four strategies for managing technical debt," Aug 2018. [Online]. Available: http://www.designsmells.com/articles/four-strategies-for-managing-technical-debt/

[17] C. Oliveira, "Think legacy modernization is a one-time event? think again!" Jul 2017. [Online]. Available: https://www.morphis-insights.com/technical-debt/

[18] "How to fight technical debt," Aug 2019. [Online]. Available: https://briisk.co/blog/how-to-fight-technical-debt/

[19] X. Liu and C. Zhang, "Dt: a detection tool to automatically detect code smell in software project," in *2016 4th International Conference on Machinery, Materials and Information Technology Applications*. Atlantis Press, 2017.

[20] F. A. Fontana, P. Braione, and M. Zanoni, "Automatic detection of bad smells in code: An experimental assessment." *Journal of Object Technology*, vol. 11, no. 2, pp. 5–1, 2012.

[21] T. Sharma, M. Fragkoulis, and D. Spinellis, "House of cards: code smells in open-source c# repositories," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 424–429.

[22] A. Kaur and G. Dhiman, "A review on search-based tools and techniques to identify bad code smells in object-oriented systems," in *Harmony search and nature inspired optimization algorithms*. Springer, 2019, pp. 909–921.

[23] J. Padilha, J. Pereira, E. Figueiredo, J. Almeida, A. Garcia, and C. Sant'Anna, "On the effectiveness of concern metrics to detect code smells: An empirical study," in *International Conference on Advanced Information Systems Engineering*. Springer, 2014, pp. 656–671.

[24] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Castor Filho *et al.*, "Evolving software product lines with aspects," in *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 2008, pp. 261–270.

[25] A. Yamashita and L. Moonen, "Do developers care about code smells? an exploratory survey," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 242–251.

[26] L. Hochstein and M. Lindvall, "Combating architectural degeneration: a survey," *Information and Software Technology*, vol. 47, no. 10, pp. 643–656, 2005.

Implementing Compositional Concurrency in Haskell

Deepshi Garg, *Student, University of Groningen*

Abstract—In times of large and complex computations, it is often desired to split the program into smaller chunks, which can be run concurrently. If a program is single threaded, CPU might end up spending a lot of idle time during I/O, memory reads, etc. With concurrent processing, this idle time can be allotted to some other process. This saves time and increases efficiency. However, if these processes communicate or are interdependent, designing them becomes difficult. This paper aims to provide easy implementation of concurrent operations in Haskell. We give detailed practical executions of the ability to split the program into parallely executing processes, use of mutable variables, transactional variables, and software transactional memory. We also provide use-case specific extensions of these features like consumer-producer message passing mechanism, buffered channels, choice, blocking and non-blocking processes, et al. Further, we make an attempt to compare them with the conventional concurrency mechanisms like locks.

Index Terms—Concurrency, Transactional Memory, Locks, Synchronisation, Atomicity

1 Introduction

Parallelism is about breaking a program into sub-tasks which can be run parallely in different computation units, without any internal interaction. However, concurrency is about breaking a program into sub-tasks which access the same resources. Concurrency leads to multiple processes running simultaneously and possibly reading and updating the same resources. This can lead to non-determinism. Such programs are often difficult to write, and their behaviour is difficult to predict and test. One thing we can do is make them as much modular as possible. Preserving this compositionality of a program will at least help us to avoid obvious mistakes by restricting modules from sharing their core logic and interacting only via designated interfaces. This paper makes an effort to put in feasible implementations of the tools of compositional concurrency in Haskell. Specifically :

- We show how the main process can be split up into multiple processes, thus providing with the ability to desing concurrent programs (Section 2.1)
- We walk through the implementation of mutable variables. This would help us design programs which share resources. This is further extended to give examples of useful abstractions which can be readily built over these mutable variables (Section 3)
- To be able to write atomic transactions and preserve compositionality in programs, we explain the concept of Software Transactional Memory and use transactional variables for a simple implementation of the same. We further extend this to explore blocking and choice mechanisms. (Section 4)
- We go through the benefits of Transactional Memory over other available concurrency mechanisms like locks (Section 5)

This paper is not about emphasising the need of concurrency and validating its benefits. Rather, it is about delving into a viable practice of the readily available tools of concurrency in Haskell, and make the programmer more comfortable with their use. The main aim is to give the programmer more control over concurrent systems via explicitly visible syntax. We also talk about preserving the compositionality of the program, while making it concurrent. In Layman's terms, we emphasize on letting the programmers control the atomicity and the blocking behaviour in their concurrent systems, while respecting the abstraction barriers among different modules [3].

- *Deepshi Garg is with University of Groningen doing Masters in Computing Science - Software Engineering and Distributed Systems, Student Number: S4199456, E-mail: d.garg@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 .

2 Starting with basics

Haskell is a pure, lazy, functional programming language created in the late 1980s by a committee of academics [7]. Explaining the three key terms mentioned here :

- Pure : Everything is immutable. Variables cannot be reassigned. A function gives same output every time it is invoked with the same set of arguments
- Lazy : Expressions are not evaluated beforehand. Rather, evaluations are done only when the result is actually needed
- Functional : Functions are first-class, i.e., they can be used like any other value in the program

With the above key terms in mind, let us write a simple Haskell program which runs a loop to print a character infinitely on the standard output screen.

```
1. loop :: Char -> IO ()
2. -- loop to print a character infinitely
3. loop ch = putChar ch >> loop ch

4. main :: IO ()
5. main = loop 'z'
```

Explaining this program for Haskell beginners :

1. It is a prototype of the function **`loop`**, which takes in a character as input and returns a value of type action **`IO ()`**. This indicates that the function **`loop`** may have side effects, like printing output on the console. However, **`()`** means a unit value, similar to **`void`** in C language.
2. Anything which starts with **`--`** becomes a comment
3. It is the function definition. The function **`loop`** takes an input character **`ch`**. This **`ch`** is passed to the function **`putChar`**, which prints it to the **`stdout`** and the function **`loop`** is invoked again with the same argument. Yes! It is a recursive function. The operator **`>>`** is an infix combinator which combines one action to another without caring about the output from the previous action. Note that a function call in Haskell is written by **`function-name arg1 arg2 ...`**
4. Prototype of **`main`** function which accepts no argument and returns **`IO ()`**. It may leave some side effects like printing on console, altering a memory location, et al.

5. We simply define the **`main`** function as a call to the **`loop`** function with argument **`z`**

When we run the program, **`main`** function, i.e., Line 4 is invoked. This program will print the character **`z`** infinite times on the **`stdout`**.

2.1 Processes

In the above example, our program runs in a single process. Say, if this process was to get stuck for some reason, it will hold up the execution of the whole program. For instance, we wish to run two loops in parallel now, such that the first **`loop`** prints a **`new-line`** character after every 500 milliseconds, and the second **`loop`** keeps on printing the character **`z`** continuously, like before. To do this, we need a mechanism to split the main process into two independent sub-processes. This enables the second **`loop`** to not get stuck while the first **`loop`** goes into a wait state for 500 milliseconds. Concurrent Haskell [5] provides the primitive **`forkIO`** which can be used to spawn a child process from the main parent process.

```
forkIO ::  IO () -> IO ()
```

`forkIO` is a function, which takes an action **`IO ()`** as an input argument, and spawns a concurrent process to perform this action. The side effects of this action are interleaved with the side effects of all the other processes running concurrently.

```
import Control.Concurrent

loop :: Char -> IO ()
-- loop to print a character infinitely
loop ch = putChar ch >> loop ch

loopWithDelay :: Char -> IO ()
-- loop to print a character after
-- a delay of 500 milliseconds
loopWithDelay ch = do { threadDelay 50000
                      ; putChar ch
                      ; loopWithDelay ch}

main :: IO ()
main = forkIO (loopWithDelay '\n') >>
       loop 'z'
```

Talking about new additions in the program :

- First line imports the package **`Control.Concurrent`** to provide namespaces for the functions **`threadDelay`** and **`forkIO`**
- We define the function **`loopWithDelay`**. The symbol **`do`** constructs an action by performing the smaller actions in sequence. The function **`threadDelay :: Int -> IO ()`** suspends the current process for the given number of microseconds.
- In the **`main`** function, we first call the function **`forkIO`** to spawn a new process for the first loop, i.e., **`loopWithDelay`**. The function **`loop`** becomes a part of the parent process.

`forkIO` has certain important properties :

- Because Haskell follows lazy evaluations, we can face examples of duplicate and stale evaluations. For instance, a newly spawned process might try to run an evaluation, which is already being run by another process. In such a case, the two processes should communicate, and the former should wait till the latter is done. Thus, **`forkIO`** needs inter-process synchronization with utmost superiority.
- Since concurrently running processes can be mutating the same resource, non-determinism is inevitable. Like in the above example, it is not possible to state as to how many **`z`** will be printed before each **`new-line`**
- The forked process is untraceable from the parent process. Thus, it is impossible to kill it or wait for its termination.
- **`forkIO`** is asymmetrical, i.e., forked process is the child process, while the other is the parent process. If the parent process terminates, child process will be terminated as well.

3 Mutable Variables

While lazily implemented **`forkIO`** gives us an ability to spawn multiple concurrent processes and provides an efficient way to synchronise among them, it still leaves out some important issues. There might be a case when processes demand exclusive access to resources. Sometimes we need explicit inter-process communications, like when one process waits for a signal from another process to do it's own job, or a simple producer-consumer message passing model. Concurrent Haskell [5] presents an elegant solution for this. Taking into account the work on mutable state [6], I-Structures [2] and M-Structures [1]; it introduces us to a new primitive type **`MVar`**:

```
type MVar a
```

A variable of type **`MVar a`**, either holds a value of some data type **`a`**, or is empty. It provides a means for information sharing among concurrent processes. It acts as a buffer variable in which a process can write into, while another one can read from. It exposes the following interface:

```
newMVar ::  IO (MVar a)
takeMVar ::  MVar a -> IO a
putMVar ::  MVar a -> a -> IO()
```

Here,

1. **`newMVar`** creates and returns a new **`MVar`** of type **`a`**.
2. **`takeMVar`** reads and returns the value stored in the given **`MVar`**, leaving it empty. If the given **`MVar`** is already empty, it blocks till it becomes non-empty.
3. **`putMVar`** writes the given value to the given **`MVar`**. It blocks if the given **`MVar`** is non-empty.

Explaining this through an example, let us consider the following program. Here, two processes run concurrently. The first process reads a character from **`stdin`**, sleeps for 2 seconds and then writes this character to a shared **`MVar`**. The second process reads the value of this **`MVar`** and prints it to **`stdout`**.

```
import Control.Concurrent

main :: IO ()
main = getChar >>= \c ->
       do { mVar <- newEmptyMVar
          ; forkIO (do { m <- takeMVar mVar
                       ; print m})
          ; threadDelay 2000000
          ; putMVar mVar c}
```

- **`getChar :: IO Char`** reads a character from **`stdin`** and returns it.
- **`>>=`** is an infix operator which combines two actions such that it passes on the output of the first action to the second.
- **`c -> E`** for some expression **`E`** denotes a lambda expression, whose scope extends as far to the right as possible.

One important thing to note here is that the second process here will wait till the the value of **mVar** becomes non-empty, i.e., for 2 seconds.

The type **MVar** can be extended to fit in multiple use-cases of concurrency. As explained in the above example, it can be used as a shared memory location, which allows multiple processes to exchange information. It can also be used as binary semaphores with **signal** and **wait** operations being implemented using **takeMVar** and **putMVar**. With some abstractions, it can also be used as message passing channels between a set of producer and consumer processes. Let us have a look at some of these implementations.

3.1 Producer/Consumer Buffer Channel

We often come across situations where a producer process needs to pass on some message to a consumer process. **MVar** can be suitably used for this as described in the previous example. Only issue is that of acknowledgement from consumer to producer about successful message consumption. As suggested in [5], this can be easily resolved using the following abstraction :

```
type CVar a = (MVar a, MVar ())
```

Say **CVar** be the channel variable. It composes of two **MVar**s.

1. First is the **dataVar** : Used to pass on data from Producer to Consumer. This data can be of any type **a**.
2. Second is the **ackVar** : Used to pass acknowledgement for successful message consumption from consumer to producer. Since acknowledgment is only a notification event and does not contain any information for now, it is safely assumed to be of unit type **()**.

The following interface can be used :

```
newCVar ::  IO (CVar a)
putCVar ::  CVar a -> a -> IO ()
takeCVar ::  CVar a -> IO a
```

- **newCVar** : Creates and returns a new channel variable.
- **putCVar** : Takes a channel variable **CVar** and a data variable **a**. It waits for the acknowledgement of successful consumption of the last message from the consumer before putting new data.
- **takeCVar** : Takes a channel variable **CVar**. It reads the message data **a** from it, publishes the acknowledgment for successful consumption, and returns the value **a**.

Putting the whole program together in executable form :

```
import Control.Concurrent

-- dataVar : Producer -> consumer
-- ackVar : Consumer -> producer
type CVar a = (MVar a, MVar ())

newCVar :: IO (CVar a)
newCVar = newEmptyMVar >>= \dataVar ->
              newEmptyMVar >>= \ackVar ->
                  putMVar ackVar () >>
                  return (dataVar, ackVar)

putCVar :: CVar a -> a -> IO ()
putCVar (dataVar, ackVar) val =
    print "consumed␣ackVar" >>
    takeMVar ackVar >>
    print "publishing␣dataVar" >>
    putMVar dataVar val

takeCVar :: CVar a -> IO a
takeCVar (dataVar, ackVar) =
    print "consumed␣dataVar" >>
    takeMVar dataVar >>= \val ->
        print "publishing␣ackVar" >>
        putMVar ackVar () >>
        return val

main = getChar >>= \c ->
  do { cVar <- newCVar
     ; forkIO (do { threadDelay 2000000
                  ; val <- takeCVar cVar
                  ; print val}) >>
      putCVar cVar 'f' >>
      print ("produced␣first␣value␣-␣", "f") >>
      putCVar cVar c >>
      print ("produced␣second␣value␣-␣", c)}
```

Understanding the flow here :

- First, the program takes an input character **c**
- Then, it creates a new channel variable **cVar**
- Then, it spawns a child process, which first sleeps for 2 seconds, and then consumes data from the shared channel variable **cVar**
- While the child process is at work, the parent process first produces a character **f** in the channel, and then attempts to produce the given input character **c** to the same channel.

However, note that, for the producer to be able to produce the second value to the channel, it has to wait for the consumer to consume the first value and publish an acknowledgment. This is clearly visible by the **print** statements put in place. In this program, given some input character **k**, the following output is received :

```
"consumed␣ackVar"
"publishing␣dataVar"
("produced␣first␣value␣-␣","f")
"consumed␣ackVar"
"consumed␣dataVar"
"publishing␣ackVar"
'f'
"publishing␣dataVar"
("produced␣second␣value␣-␣",'k')
```

3.2 Channel with a infinite buffer stream

The limitation in the previous example is that it can handle only one message at a time. Concurrent Haskell [5] also suggests an abstraction where we use **MVar** to derive a channel variable which can handle multiple messages at once. Somethings to note here :

- It follows the First In First Out data rule for reading and writing data. One can say it is very similar to Queue in conventional object oriented languages. Read and write operations happen at opposite ends of the channel.
- We do not have the mechanism for acknowledgment anymore. It is not needed because producer can now produce as many messages as it wants without caring about consumption.

Defining the channel type,

```
type Channel a = (MVar (Stream a), MVar
(Stream a))
type Stream a = MVar (Item a)
data Item a = Item a (Stream a)
```

- **Channel** consists of 2 **MVar**s, one to point to the reading end, other for the writing end.

- **Stream** consists of the item to be held in the channel.
- **Item** is a pair of 2 values : the first element of the stream, and the rest of the stream.
- **a** is the type of data this channel is designed to hold

It can be compared with the doubly linked lists in object oriented programming, where

- reading **MVar** is the read pointer, from where the consumer retrieves data
- writing **MVar** is the write pointer, from where the producer writes data
- **Item** is a linked list node, which holds the current node data value, as well as the pointer to the next node or we can say the rest of the list

It will expose the following interface :

```
newChannel ::  IO (Channel a)
putChannel ::  Channel a -> a -> IO()
getChannel ::  Channel a -> IO a
```

- **newChannel** creates and returns a new channel for type **a**
- **putChannel** creates a new **MVar** with value **a**, and updates the **write-MVar** of the channel to contain this new item. It also updates the old stream to have the new item
- **getChannel** reads the value in the first item of the stream in **read-MVar**, makes the **read-MVar** point to next item, and return the value of the first item

Looking at the complete implementation:

```
import Control.Concurrent

-- it is like double-linked list :
-- readVar -> start pointer,
-- writeVar -> end pointer,
-- stream -> list of items,
-- item -> (head, rest of the list)
type Channel a = (MVar (Stream a),
                  MVar (Stream a))
type Stream a = MVar (Item a)
data Item a = Item a (Stream a)

newChannel :: IO (Channel a)
newChannel = newEmptyMVar >>= \read ->
   newEmptyMVar >>= \write ->
      newEmptyMVar >>= \emptyStream ->
         putMVar read emptyStream >>
         putMVar write emptyStream >>
         return (read,write)

putChannel :: Channel a -> a -> IO()
putChannel (read,write) val =
   newEmptyMVar >>= \newItem ->
      takeMVar write >>= \oldStream ->
         putMVar write newItem >>
         putMVar oldStream (Item val newItem)

getChannel :: Channel a -> IO a
getChannel (read,write) =
   takeMVar read >>= \head ->
    takeMVar head >>= \(Item val restStream) ->
         putMVar read restStream >>
          return val

main = getChar >>= \c ->
 do { channel <- newChannel
    ; forkIO (
       do {
        threadDelay 2000000
      ; val <- getChannel channel
      ; print ("value␣read␣in␣fork", val)}) >>
      putChannel channel 'f' >>
      print "put␣first␣value␣-␣'f'" >>
      putChannel channel c >>
      print ("put␣second␣value␣-␣", c) >>
      do {
       threadDelay 3000000
       ; val <- getChannel channel
       ; print ("value␣left␣in␣channel", val)}}
```

Here, understanding the flow :

1. A character **c** is taken as input
2. A channel is created
3. A child process is forked. It first sleeps for 2 seconds, and then reads a value from the channel
4. First character **f** and then the input character **c** is written to the channel
5. Parent process sleeps for 3 seconds and then reads the value from the channel.

Again, the **print** statements help us in understanding the program flow. Given the input **k**, following output is generated :

```
"put␣first␣value␣-␣'f'"
("put␣second␣value␣-␣",'k')
("value␣read␣in␣fork",'f')
("value␣left␣in␣channel",'k')
```

Note that values are written to the channel regardless of whether they are consumed or not.

4 Software Transactional Memory

Now consider the case when multiple shared resources need to be accessed in a way that the intermediate stage should not be visible to any other process, or a scenario where multiple processes access the same shared resource. For instance, **transfer** function for a bank account. First, it needs to subtract the amount from the sender account, and then add it to the receiver account. Both of the accounts would be shared resources here, because multiple such transfers happen in a bank. In this scenario, we need to pitch in the idea of making both these steps into a transaction. As stated in [3], the key idea of a transaction is that a block of code, including nested calls, can be enclosed by an atomic block, with the guarantee that it runs atomically with respect to every other atomic block.

Transactions can be implemented using optimistic execution. Here, when an atomic transaction is to be performed, a transaction log is created. The idea being that, whenever a shared transactional variable is updated, instead of updating it actually, an entry in the transaction log is created. It has the variable's address and the new value. Similarly, when a read operation is to be performed within the transaction, first the log is searched for an updated value. If no entry is found, the actual variable is read and recorded in the log. Once all the steps of the transaction are done, final changes are committed to the actual variable. There might be multiple threads accessing the same variables, and one of them might end up updating the variable while our transaction is still running. So, before committing the transaction, a validation happens, where the initially recorded value of the variable is matched with the current value. If they do not match, transaction log is discarded and the transaction is performed again.

In Haskell, we have **STM**. It is a monad, just like **IO**. The difference is that within a **STM** block, side effects are only limited to transactional variables of type **TVar**.

Very similar to **MVar**, we have the following interface for **TVar**:

```
data TVar a
newTVar ::  a -> STM (TVar a)
readTVar ::  TVar a -> STM a
writeTVar ::  TVar a -> a -> STM ()
```

STM actions can be composed together using **do** notation, just like **IO** actions. All the **STM** actions composed together, can be made compatible with the **IO** actions using

```
atomically ::  STM a -> IO a
```

atomically takes in all the **STM** actions, perform them in a single transaction log, and finally commits the transaction log, thus performing an **IO** action. It also prevents from performing an **IO** action within this block by mistake. This helps to not repeat the **IO** action in cases of failed validation of transaction log, where the transaction has to be performed again.

Considering the bank transfer example from [4], we can implement it using **STM** as follows:

```
import Control.Concurrent
import Control.Concurrent.STM

type Account = TVar Int

newAccount :: Int -> IO(Account)
newAccount balance =
   atomically (do { acc <- newTVar balance
                  ; return acc })

withdraw :: Account -> Int -> STM ()
withdraw acc amount =
   do { bal <- readTVar acc
      ; writeTVar acc (bal - amount) }

deposit :: Account -> Int -> STM ()
deposit acc amount = withdraw acc (- amount)

getBalance :: Account -> IO Int
getBalance acc =
   atomically (do { bal <- readTVar acc
                  ; return bal})

transfer :: Account -> Account -> Int -> IO ()
transfer from to amount =
   atomically (do { deposit to amount
                  ; withdraw from amount })
main :: IO()
main = do { acc1 <- newAccount 10
          ; acc2 <- newAccount 3
          ; forkIO (transfer acc1 acc2 5)
          ; forkIO (transfer acc2 acc1 2)
          ; threadDelay 1000000
          ; bal1 <- getBalance acc1
          ; bal2 <- getBalance acc2
          ; print ("balance␣in␣acc1␣:␣", bal1)
          ; print ("balance␣in␣acc2␣:␣", bal2)}
```

Here, in **transfer**, we use **atomically** to perform **deposit** and **withdraw** within the same transaction. In the **main** function, we can see that the two processes try to access the same accounts, but none of the transactions affects the other.

4.1 Blocking

There might be a requirement where we wish to block a thread until certain condition is fulfilled. For instance, we might want to block the **transfer** until the sender account has sufficient balance. We can do this by using the following function described in [4]

```
retry ::  STM a
```

retry function simply blocks the thread if the given condition is not fulfilled and retries later after some time. Now, it makes sense only to **retry** after some changes have been made to the involved variables. Thus, **retry** is attempted only after a write happens to the variables enlisted in the transaction log of this transaction.

We also have a supportive function **check** which helps in checking whther the given condition is satisfied or not.

```
check ::  Bool -> STM ()
check True = return ()
check False = retry
```

As suggested in [4], we can thus modify our previous example as follows :

```
limitedDeposit :: Account -> Int -> STM ()
limitedDeposit acc amount =
    limitedWithdraw acc (- amount)

limitedWithdraw :: Account -> Int -> STM ()
limitedWithdraw acc amount =
    do { bal <- readTVar acc
       ; check (amount <= bal)
       ; writeTVar acc (bal - amount)}

limitedTransfer :: Account -> Account ->
                   Int -> IO ()
limitedTransfer from to amount =
 atomically (do { limitedDeposit to amount
                ; limitedWithdraw from amount})
```

4.2 Choice

Now, suppose we wish to have a secondary sender account. For example, we wish to transfer amount **x** from account **acc1** to account **acc2**. But if **acc1** does not have enough balance, instead of blocking the process, we can transfer from another account **acc3**. However, if both of them do not have sufficient balance, we can block until any one of them gets sufficient funds.

This can be achieved by using another primitive function of STM Haskell, **orElse**

```
orElse ::  STM a -> STM a -> STM a
```

OrElse a b takes two **STM** actions and combines them such that : perform **a**. If **a** retries, perform **b**. If **b** retries, perform the whole combined action again. Since here too, the internal blocking happens using **retry**, action would be retried only when at least one of the involved transactional variables is changed by any other process.

As suggested in [4], if we wish to accommodate this in our previous example, code would change as follows :

```
choiceWithdraw :: Account -> Account ->
                  Int -> STM ()
-- (choiceWithdraw acc1 acc2 amt)
-- withdraws amt from acc1,
-- if acc1 has enough money, else from acc2.
-- If neither has enough, it retries.
```

```
choiceWithdraw acc1 acc2 amt =
    orElse (limitedWithdraw acc1 amt)
           (limitedWithdraw acc2 amt)

choiceDeposit :: Account -> Int -> STM ()
choiceDeposit acc amount =
    choiceWithdraw acc acc (- amount)

choiceTransfer :: Account -> Account ->
                  Account -> Int -> IO ()
choiceTransfer from1 from2 to amount =
    atomically (
       do { choiceDeposit to amount
          ; choiceWithdraw from1 from2 amount})
```

5 Comparing STM With Locks

But why do we need STM at all? We have locking mechanism readily available. It is modular, and simple to execute using conditional statements.

Let us consider our previous bank **transfer** example referred from Beautiful Concurrency [4]. To ensure that the whole transaction of transferring the amount from one bank account to another happens atomically, locks can be implemented in some object oriented language, like Golang, as follows :

```
\\ Send money from acc1 to acc2
transfer(acc1, acc2 Account, amount int){
    acc1.lock(); acc2.lock()

    acc1.withdraw(amount)
    acc2.deposit(amount)

    acc1.unlock(); acc2.unlock()
}
```

This seems easy. However, it might result into deadlocks at high scale. For instance, another transfer happening at the same time in the opposite direction. Both the processes will acquire locks on one of the accounts, and will wait infinitely for acquiring a lock on the second account.

If we know that it is going to be only 2 locks to be acquired, we can put up some condition such that deadlock never happens. For instance, define a global order for lock acquisition, like

```
if acc1 < acc2 {
    acc1.lock(); acc2.lock()
} else {
    acc2.lock(); acc1.lock()
}
```

This becomes difficult as the function specifications increase, like, how do we tackle the situation discussed in Section 4.2. In cases of choice, what is the correct order to acquire locks to be able to avoid deadlocks? In cases of blocking, we will have to release the locks, while waiting on a process.

Long story short, locks pose multiple different unresolved issues :

- Taking too many locks which can block other processes, or taking too less locks which can lead to scenarios where two concurrent processes end up working on the same resource.
- Taking the locks in wrong order which can lead to deadlock, as discussed above.
- Simulating all the scenarios of lock acquisition combinations, and handling errors in all of them.
- Compositionality and modularity is compromised. For instance, in the above example, the **transfer** function needs to know about the locking mechanisms, which falls out of its scope if Single Responsibility Principle is followed.

STM handles all these issues with elegance. In STM, order of changes does not matter because changes occur in transaction log with optimistic execution. Also, compositionality is preserved by the monad type **STM()**. **atomically** makes sure the whole module is executed together with no intermediate state being visible to any other process.

6 Conclusion

We have described multiple ways to write concurrent programs in Haskell. We have discussed the approach to spawn concurrent processes, and use shared resources amongst them. We have discussed mechanisms for message passing among multiple concurrent processes. This is achieved using **forkIO** and **MVar**. However, it has one limitation. Updates happen directly at the variable location. This is good if the system has only one producer and one consumer. But it can cause inconsistencies in cases of multiple producers and multiple consumers. Thus, we narrowed down to Software Transactional Memory **STM**, implemented using transactional variables **TVar**. Here, optimistic execution approach is followed, and the whole module of transactions is performed only when the transaction log is validated. It is compared against the popular locking mechanism and the following guarantees have been developed from STM :

1. Logic for synchronisation need not be a part of the domain logic [4].
2. Multiples modules can be clubbed together, and the final transaction is executed as a whole
3. Within an **STM** transaction, no actual **IO** will be performed

As promising as it may seem, STM is also not a silver bullet. There is an inevitable overhead of maintaining the transaction log, and validating it, and may be redoing the the transaction in cases of failed validation. It proves to be highly memory and computation intensive in cases of complex systems with very high scale.

7 Future Work

We have not considered and compared the above stated tools of concurrency in Haskell with the available approaches in other languages, like context, channels and goroutines in GoLang, etc. This research can be further extended towards such comparisons, and may be drawing a parallel approach of concurrent programming amongst the two paradigms of sequential programming and object oriented programming.

References

[1] Arvind, P. S. Barth, and R. S. Nikhil. M-structures: Extending a parallel, non-strict, functional language with state. ACM Transactions on Programming Languages and Systems (TOPLAS), September 1991.

[2] Arvind, K. Pingali, and R. S. Nikhil. I-structures - data structures for parallel computing. ACM Transactions on Programming Languages and Systems (TOPLAS), October 1989.

[3] T. Harris, S. Marlow, S. P. Jones, and M. Herlihy. Composable memory transactions. Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP '05), Chicago IL, USA, June 2005.

[4] S. P. Jones. Beautiful concurrency. In *Beautiful Code*. O'Reilly Media, May 2007.

[5] S. P. Jones, A. Gordon, and S. Finne. Concurrent haskell. 23rd ACM Symposium on Principles of Programming Languages (POPL '96), St Petersburg Beach, Florida, January 1996.

[6] J. Launchbury and S. P. Jones. State in haskell. Lisp and Symbolic Computation, December 1995.

[7] B. Yorgey. Haskell basics. Introduction to Haskell - Credit Course in University of Pennsylvania, December 2014.

A Review of Scene Recognition Techniques Based on Convolutional Neural Networks

Alina Matei, Andreea Glavan

Abstract— Can computers tell the difference between a hotel room and a bedroom? The field of image recognition has greatly advanced during the past years, giving way to new challenges such as scene recognition. This paper discusses three approaches to the task of scene recognition: context based recognition, combining scene and object centric knowledge and spatio-temporal residual units for dynamic scenes. The approaches are based on convolutional neural networks (CNNs). The aim of the paper is to aid professionals in making an informed choice about which method is most suitable for the scene recognition task they are facing. For this purpose, the three approaches are detailed and compared quantitatively, as well as qualitatively. The comparison aims to provide an overview of the strengths and limitations of the methods. The results indicate that combining scene and object knowledge in CNNs is superior to only context based recognition. For dynamic scenes, spatio-temporal residual networks achieve state-of-the-art results. However, the approach should be chosen on a case-by-case basis, since no one method can generalize well to all scene recognition tasks and datasets.

Index Terms—Computer vision, Deep learning, Image recognition, Convolutional neural networks.

1 Introduction

During the past years, the field of visual recognition has greatly advanced such that nowadays state-of-the-art visual recognition techniques can nearly replicate the way a human would describe the contents of an image. Achieving this performance is possible through the use of deep learning methods, vast training times and large datasets. One domain of image recognition which still poses difficulty is the area of scene (or place) recognition.

Given two images, one of a forest and the other of a park or, alternatively, one depicting a restaurant and the other a dining room, humans are able to identify the different categories by using contextual clues. However, for a computer, these contextual clues are not obvious, therefore the challenge of distinguishing between scene categories with low variance remains. Previously, visual recognition techniques have made extensive use of object recognition when faced with such problems [1, 2]. The scenes would be classified based on exhaustive lists of objects identified in the scene.

With the development of deep learning, a new approach was introduced. Deep learning architectures, such as Convolutional Neural Networks (CNNs), focus on the overall background of the image and the presented location, thus attempting scene classification from a contextual perspective [3, 4, 5].

The classification of visual data depicting scenes or places can therefore be approached from two angles: on one hand, by object identification and on the other hand, by overall context recognition. These two approaches are complementary since object detection can provide additional semantic information that might be lost when only considering the contextual clues. By combining deep learning techniques with both methods, it is possible to extend the performance obtained individually by each method [6]. Furthermore, considering the scale at which objects appear in a scene can facilitate the classification process by targeting distinct regions of interest within the image. Since objects usually appear at lower scales, the object classifier should target local scopes of the image, while the places classifier should be aimed at the global scale, in order to capture contextual information [7].

Visual data available nowadays is not only presented in a static format, as images, but also dynamically, as video recordings. The problem of scene recognition can also be generalized to include dynamic data. However, this type of data employs an additional level of complexity since the inherent temporal aspect of video recordings must be considered: a video can capture scenes which suffer temporal alterations. In this case, the classification model should be powerful enough to account for the temporal changes and nonetheless give the correct scene classification. This issue has been addressed by literature [8, 9, 10].

In this paper, we present, analyse and compare state-of-the-art scene recognition techniques which rely on the deep learning framework. We also include a brief analysis of the datasets available which assisted the fast growing rate of advancement in the field. Our final contribution is presenting some applications of the techniques investigated. The aim of the paper at hand is to provide a guideline in the field of scene recognition, that could possibly assist professionals in choosing the most suitable approach for scene recognition on a case-by-case basis. While a wide variety of techniques are available, we will focus on analysing the following:

1. Scene recognition based on CNNs by combining scene-centric and object-centric knowledge at different image scales [7]
2. Scene recognition based on pre-trained CNN architectures on the Places365 database, an emerging dataset covering an exhaustive set of places categories [11]
3. Dynamic scene recognition using temporal residual networks [8]

The proposed methods are considered quantitatively, by judging their performances based on metrics such as accuracy (top-1 and top-5), as well as qualitatively, by analysing their performances and limitations in the context of the corresponding datasets. The implementation and the results achieved by the approaches cannot be judged independent of their experimental setup (i.e. dataset, architecture of the CNN), therefore the scope and limitations of the conducted experiments must also be considered.

This paper is structured as follows: in Section 2 we discuss the available datasets supporting scene and object focused recognition. Section 3 addresses the methodology of the state-of-the-art techniques and approaches discussed in the paper at hand. Furthermore, Section 4 presents a detailed comparison of the presented approaches. Finally, in Section 5, concrete applications of the scene recognition methods are presented, Section 6 offering the conclusions succeeding our investigation. Lastly, Section 7 suggests possible extensions for the research conducted in this paper.

2 Datasets

The aim of using deep learning CNNs for the task of scene recognition is to tackle the complexity and high variance of the the problem.

These aspects are addressed by training models to learn a large range of specific features of images depicting scenes. The latest advancements in deep learning methods for scene recognition are motivated by the availability of large and exhaustive datasets.

Most available datasets are focused on object categories, providing labels, bounding boxes or segmentations. One of the most conspicuous object dataset is ImageNet [12], which provides 3.2 million object images in total. The exhaustive list of object categories offered by this dataset follow a hierarchical structure based on the WordNet structure [1]. The dataset contains 80,000 noun synsets [2], each synset being illustrated by roughly 500-1000 images. Another notable object dataset is COCO (Common Objects in Context)[13], which provides a limited number of 80 object categories illustrated by a total of 1.5 million instances. Open Images V5 [3] covers 600 boxable object classes, offering a training set of roughly 1.7 million images with annotated bounding boxes, object segmentations as well as visual relationships between objects. Overall, the problem of object recognition has a vast support in terms of datasets which can be employed for training deep learning models.

As for datasets which are scene-centered, initial datasets did not offer a comprehensive collection of places categories. The 15 scene dataset [14] includes a limited number of images per scene (i.e. 200-400 images) collected from the COREL collection, personal images and Google Image Search. The MIT Indoor67 [15] offers a range of 67 classes of indoor scenes, while the more extensive SUN397 [16] contains 397 places organized in a tree-like structure, with the three main meta-classes being: indoor, outdoor natural, outdoor man-made. However, each class has a limited number of samples with a complete total of 108,754 images for the entire dataset. A significant step towards the improvement of scene recognition models was the public release of the Places Database [11]. The Places Database furthered the scene recognition research by providing an exhaustive collection of scene images; the range of classes includes categories such as 'airfield', 'kindergarten classroom', 'restaurant' and 'zen garden', providing examples of both common and more peculiar places. A total of 10 million images were gathered, out of which 365 scene categories were chosen to be part of the dataset. The dataset is available in the Places365-standard format (i.e. 365 categories, roughly 1 million images training set, validation set with 50 images per class and test with 900 images per class) and the Places365-challenge format which extends the training set to 8 million image samples in total. With a dataset of this magnitude, training of CNNs exclusively on scene data becomes feasible, such an approach is presented in Section 3.

Scene recognition also encloses dynamic scene data; due to the limited amount of available datasets which include such data, most of the research efforts in this sub-field also include gathering suitable experimental data. Literature [17] contributes the Maryland "in-the-wild" dataset, which includes 13 classes each amounting for 10 videos chosen for a high intra-class variance. The categories vary from 'avalanche' to 'smooth traffic' scenes. The YUPENN dataset [18] introduces 14 classes depicting natural dynamic scenes (i.e. 'beach', 'city street', 'water fall'), with an overall total of 420 image sequences. As opposed to the Maryland "in-the-wild" dataset, which contains a high degree of scale motion due to camera movement, the YUPENN dataset presents data captured by static cameras. Literature [8] extends the YUPENN dataset, introducing the YUP++ dataset which samples 20 scene classes and enhances the number of videos per class (i.e. three times as many videos compared to the YUPENN dataset). The additional categories are 'building collapse', 'escalator', 'falling trees', 'fireworks', 'marathon' and 'waving flags'. The scope of the categories that are being recorded amongst the three datasets presented is not nearly as exhaustive as in the case of the objects and places datasets mentioned above. This is an indicator of the incipient status of research in this particular area of scene recognition.

[1]The WordNet lexical database for the English language can be accessed here: https://wordnet.princeton.edu/

[2]Sets of synonyms that can be used interchangeably to refer to the same concept

[3]The dataset can be accessed here:
https://storage.googleapis.com/openimages/web/download.html

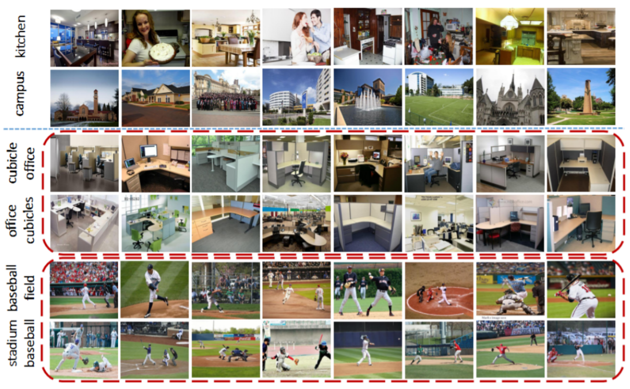


Fig. 1. Image examples from the Places Database. *Top Rows* present images from two classes (i.e. 'kitchen' and 'campus') with high intra-class variance. *Middle and Bottom Rows* present pairs of classes (i.e. 'cubicle office' and 'office cubicles', 'baseball field' and 'stadium baseball') with low inter-class variance.

3 Methodology

Various approaches have been proposed for the task of scene recognition; these methods vary in several aspects: from the supporting datasets used, the architectural models employed for feature extraction to the type of features of interest which are used to describe the scenes. In the section at hand, three different methodologies are detailed.

3.1 Static image based scene recognition

The inherent difficulty of the place recognition task is closely related to the nature of the images depicting a scene context. Literature [19] describes two major challenges: *visual inconsistency* and *annotation ambiguity*.

On the one hand, *visual inconsistency* refers to the high intra-class variance of scene categories. Demarcation of the categories is a subjective process which is highly dependent on the experience of the annotators, therefore images from the same category can showcase significant differences in context and appearance (see Figure 1, first two rows).

On the other hand, *annotation ambiguity* refers to low inter-class variance. Some scene categories can share similar visual appearances which creates the issue of class overlaps. Since images belonging to two different classes can be easily confused with one another (see Figure 1, four bottom rows), the class overlap cannot be neglected.

In order to account for these challenges inherent to static images, various approaches propose considering the semantic or contextual composition of the image as detailed below.

3.1.1 Places CNNs for scene recognition

Literature [11] introduces a new database benchmark for place recognition, specifically the Places Database (variations of this are the Places365-standard and Places365-challenge datasets described in Section 2). According to the authors of [11], the database encompasses a total of 434 scenes which account for 98% of the type of places a person can encounter in the natural and man-made world.

The approach proposed by the authors of literature [11] is to exploit the vast dataset at hand by training three popular Convolutional Neural Network (CNN) architectures (i.e. AlexNet [20], GoogLeNet [21], VGG16 [22]) on the available formats of the Places dataset. Furthermore, the ResNet152 [23] residual network architecture has been fine-tuned over the Places365-standard dataset. The choice for deep architectures is motivated by the complexity of the task: since the images are not described semantically (for example, as defined by the objects present in the scene) the models used are aimed at learning generic, contextual features of the scenes, which are captured by the high-level convolutional layers.

3.1.2 Combining object-centric and scene-centric knowledge at scale in CNNs

The approach presented by literature [7] entails effectively combining object-centric and scene-centric knowledge as an alternative for patch-based CNNs for scene recognition. Patch-based CNNs typically use a singular CNN model to analyse various patches extracted from the original input image; these patches have, consequently, various scales. This method addresses the issue of dataset biases which arise under different scaling conditions of the images. If the scaling operation is significant, the features of the data can drastically change from describing scene data to object data. Using an singular generic CNN model as a feature extractor from patches cannot overcome the bias introduced by the different scales of the image. However, networks trained on specific datasets (i.e. places or object specific datasets) are more suitable for different scale ranges, therefore alleviating the bias problem.

The proposed method for scene recognition involves a multi-scale model which combines various CNNs specialized either on object or place knowledge. The architectures are employed to extract features in parallel from patches, which represent the input image at increasingly larger scale versions. In order to aggregate the extracted features over the architectures used, simple max pooling[4] is adopted in order to downsample the feature space. The multi-scale model combines several AlexNet architectures [20]. The hybrid multi-scale architecture uses distinctive models for different scale ranges; depending on the scale range, the most suitable model is chosen from: object-centric CNN (pre-trained on the ImageNet dataset [12]), scene-centric CNN (pre-trained on the Places dataset [11]) or a fine-tuned CNN (adapted to the corresponding scale based on the dataset at hand). An overview of the proposed model is presented in Figure 2.

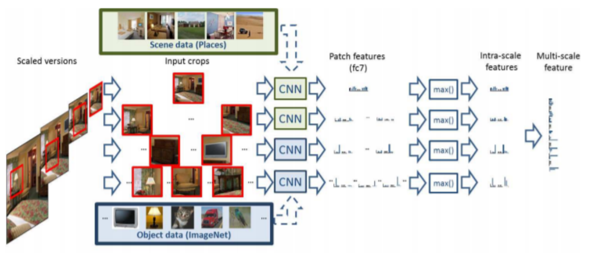


Fig. 2. Multi-scale hybrid architecture combining scale-specific networks as proposed by [7]. ImageNet and Places pre-trained CNNs are combined according to the corresponding scale of the image patch. This method adapts test data to the underlying train data of the networks to account for dataset biased. Scale specific features are obtained using max pooling at each scale, then concatenated into a final multi-scale feature.

The hybrid multi-scale model suggested can also be seen as a manner of combining and exploiting the training data available in the Places and ImageNet datasets. The knowledge learned from the two datasets is combined in a scale-adaptive way. In total, seven scales were considered; the scales were obtained by scaling the original images between 227x227 and 1827x1827 pixels. Variations of the architecture proposed include a single-scale architecture (for establishing a baseline), two-scale hybrid architecture (i.e. using one network pre-trained on ImageNet and one pre-trained on Places) and finally, the multi-scale hybrid architecture which employs seven networks (either pre-trained on ImageNet or Places) applied to seven different scales. For the final classification given by the multi-scale hybrid approach, the concatenation of the fc7 features (i.e. features exptracted by the 7th fully connected layer of the CNN) from the seven networks are considered. Principal component analysis (PCA) is used to reduce the feature space, the final feature vector has approximately 4096 combined dimensions.

[4]Max pooling is a pooling operation which computes the maximum value in each patch of a feature map; it is employed for downsampling input representations.

3.2 Dynamic image sequences based scene recognition

While early research in the field of scene recognition has been directed at single images, lately attention has been naturally drawn towards dynamic scene recognition. The task of dynamic scene recognition is closely related to action recognition [24, 25] as well as video-based recognition [26, 27]. CNNs have shown promising results for the general task of scene recognition in single images and have the potential to be generalized to video data as well [25, 28]. To achieve this generalization, the spatio-temporal nature of dynamic scenes must be considered: while static scenes (depicted as single images) only present spatial features, dynamic scenes also capture temporal transformations which affect the spatial aspect of the scene.

3.2.1 Temporal residual networks for dynamic scene recognition

Literature [8] introduces an original CNN architecture, T-ResNet, based on residual networks [29, 30]. T-ResNet employs transfer learning to adapt the spatial-centric residual architecture to a spatio-temporal-centric network. The transfer learning process implies maintaining and adapting the knowledge stored by the spatial residual network after the transformation to the spatio-temporal architecture.

The spatial CNN used as a starting point is the ResNet50 architecture [23], pre-trained on the ImageNet dataset [12]. The architecture is transformed into T-ResNet by replacing the last layer (i.e. the prediction layer); every first and third residual unit at each convolutional stage (i.e. conv2_x to conv5_x) that includes residual units are switched to the proposed spatio-temporal unit. The proposed spatio-temporal unit injects temporal information into residual blocks of the CNN through 1D temporal filtering (see Figure 3). The proposed temporal convolution block builds on the idea of inception: using spatio-temporal filters, the architecture is able to model the temporal characteristic of the features learned by the previous layer. The temporal filters are initialized at random. To complete the architectural structure, temporal max pooling is performed after the spatial global average pooling layer present in the ResNet50 model.

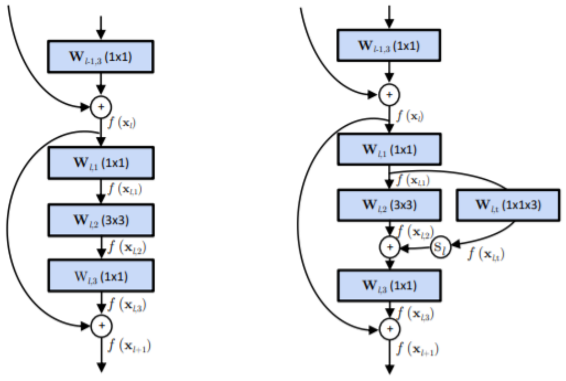


Fig. 3. Comparison between the original residual unit of the ResNet architecture (left) with the temporal residual unit of the proposed T-ResNet architecture [8] (right). The temporal residual unit extends the 'bottleneck structure' of the original residual unit with an additional temporal convolutional block and an affine scaling layer, s_l.

T-ResNet accepts 16 frame inputs. To guarantee transfer learning, the network is trained on the corresponding dataset (depending on the context in which the architecture is used) with a very low learning rate (i.e. 10^{-2}). The learning rate is decreased by an order of magnitude after the performance on the validation set stagnates. Batch normalization [31] is used with no dropout [5].

Given the spatio-temporal feature of the network, the training process can be accelerated by considering the two aspects involved: in the first stage, the network is trained in a spatial manner (training data is randomly sampled from the sets of frames, one frame per set), while during the second stage the residual units are transformed to the proposed spatio-temporal units and the training process is restarted (training data now contains the entire set of 16 frames).

4 Comparison and Discussion

The different methods presented in Section 3 have their strong and weak points. In order to give an overview of the cases for which each method is most suitable, the results obtained by each approach will be considered and discussed from a quantitative and qualitative point of view.

4.1 Quantitative comparison of the presented methodology

The quantitative comparison refers to the top-1 accuracy metric, which is defined as the percentage of total correct predictions out of the total number of data points in the test set.

The method described in Section 3.1.1 (places CNN) displayed the highest accuracy, 92.99%, on the SUN Attribute dataset [32] (used as a test set in this case). The architecture which obtained this score is VGG [22], pre-trained on the Places365 dataset. The method described in Section 3.1.2 (object and places CNN) obtained the highest accuracy of 95.18% on the 15 scenes dataset [14] with the dual hybrid architecture proposed (two-scale model consisting of ImageNet and Places pre-trained networks based on the scale). Considering the study presented by [33], both approaches obtain a significantly higher accuracy in contrast to a human expert which only achieves a percentage of 70.60% in accuracy.

To compare the performance of the above approaches for static scene recognition, their performance on the following datasets is considered: 15 scenes dataset [14], MIT Indoor 67 [15] and SUN 397 [16]. An overview of the quantitative results comparison is presented in Table 1. The obtained accuracies are presented below:

- 15 scenes dataset: the places CNN (see Section 3.1.1) obtains accuracy 91.97% with architecture VGG pre-trained on Places365, while the Hybrid1365 VGG (pre-trained on the 1365 merged classes of ImageNet and Places365) obtains 92.15% in accuracy. The seven-scale hybrid VGG architecture which combines object-centric and scene-centric data (see Section 3.1.2) is superior, with a 94.08% accuracy percentage.

- MIT Indoor 67: the seven-scale hybrid approach with AlexNet networks obtains a 80.97% accuracy, being again superior to the Places365 VGG architecture with a mere 76.53% accuracy score.

- SUN 397: the SUN 397 is a more complex dataset, therefore an overall decrease in performance can be observed; however, the seven-scale hybrid architecture with AlexNet networks is winning over the Places365 VGG model with accuracy 65.38% compared to 63.24%.

Moreover, the Hybrid1365 VGG architecture scores the highest average accuracy of 81.48% over all the experiments conducted for the place-centric CNN approach (has highest performance for 2 out of 3 comparison datasets as shown in Table 1). This, together with the quantitative comparison above, shows that the approach of combining both object-centric and scene-centric knowledge can potentially establish a new performance standard for scene recognition.

The results achieved by the T-ResNet architecture described in Section 3.2.1 will be presented individually, since there are no grounds for comparison with the previously mentioned methods due to the different characteristics of the corresponding data (static vs. dynamic). The YUP++ dataset [8] was used for proving the performance of the novel T-ResNet model. T-ResNet is mainly compared with the classical ResNet architecture as shown in Table 2. The superiority of the T-ResNet is evident: T-ResNet achieves the following percentages in accuracy: 92.41% on the YUP++ static camera partition, 81.50% on the YUP++ moving camera partition and finally 89.00% on the entire YUP++ dataset. Opposingly, ResNet achieves the respective percentages: 86.50%, 73.50% and 85.90%. The most significant difference worth of 8 percentage points is attained for the moving camera partition of the dataset, which presents a more complex challenge (in comparison with the dynamic data captured with a static camera). This demonstrates the superiority of the spatio-temporal approach.

	15 scenes	MIT Indoor	SUN 397
Places CNNs			
Places365 AlexNet	89.25%	70.72%	56.12%
Places365 GoogleNet	91.25%	73.20%	58.37%
Places365 VGG	91.97%	76.53%	**63.24%**
Hybrid1365 VGG	**92.15%**	**79.49%**	61.77%
Object and scene centric knowledge at scale			
7-scale Hybrid VGG	**94.08%**	80.22% (w/ FT)	63.19%*
7-scale Hybrid AlexNet	93.90%	**80.97%** (w/ FT)	**65.38%**

Table 1. An overview of the quantitative comparison between methods proposed in [11] and [7]. The winning performances are highlighted in bold; w/FT refers to the fine-tuning of the architecture on the corresponding dataset. *6 scales only, scale 1827x1827 not included due to the nature of the dataset.

	YUP++ static	YUP++ moving	YUP++ complete
Temporal residual network for dynamic scenes			
ResNet	86.50%	73.50%	85.90%
T-ResNet	**92.41%**	**81.50%**	89.00%

Table 2. Overview of the results achieved by the spatio-temporal residual network (T-ResNet) proposed in [8]. Literature [8] also proposes a new dataset (YUP++ complete). YUP++ static refers to the partition of the dataset capture with a static camera, while YUP++ moving to the partition with camera movement.

4.2 Qualitative comparison of the presented methodology

Addressing the task of scene recognition from the place perspective as presented in Section 3.1.1, the CNNs are expected to learn deep features that are relevant for the contextual clues present in the image. Literature [11] observers that the low-level convolutional layers detect low-level visual concepts such as object edges and textures, while the high-level layers activate on entire objects and scene parts. Even though the model has been previously trained on an exclusively places-centric dataset, the network still identifies semantic clues in the image by detecting objects alongside contextual clues. Therefore, CNNs trained on the Places Database (which does not contain object labels) could still be employed for object detection.

Another aspect arising from training the same architecture on datasets with different number of scene categories (i.e. Places205 and Places365) proves that having more categories leads to better results as well as more predicted categories (architecture AlexNet trained on Places205 obtains 0.572 top-1 accuracy, while the same architecture trained on Places365 obtains 0.577 accuracy) as stated in [11]. For the places CNN approach two main types of mis-classifications occur (see Figure 4): on one hand, less-typical activities happening in a scene context (e.g. taking a photo at a construction site) and on the other hand, images depicting multiple scene parts. A possible solution, as proposed by [11], would represent assigning multiple ground-truth labels in order to capture the content of an image more precisely.

[5]Dropout refers to ignoring a random number of units of the network at training phase to avoid overfitting.

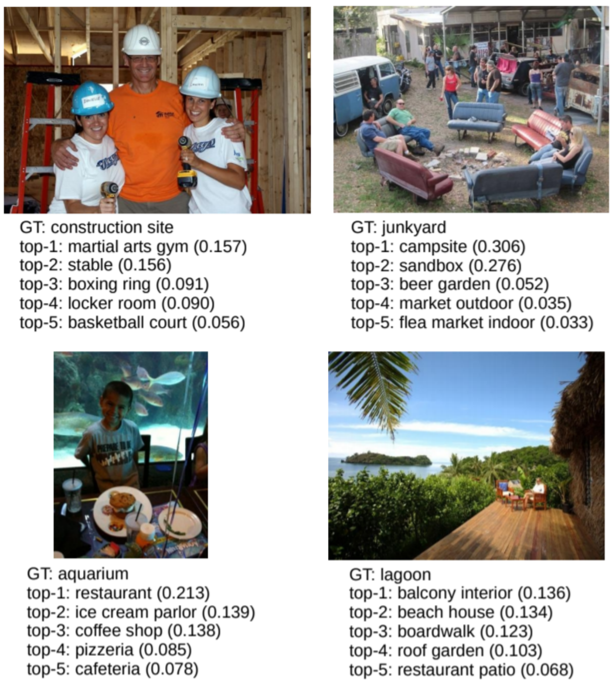

GT: construction site
top-1: martial arts gym (0.157)
top-2: stable (0.156)
top-3: boxing ring (0.091)
top-4: locker room (0.090)
top-5: basketball court (0.056)
GT: junkyard
top-1: campsite (0.306)
top-2: sandbox (0.276)
top-3: beer garden (0.052)
top-4: market outdoor (0.035)
top-5: flea market indoor (0.033)
GT: aquarium
top-1: restaurant (0.213)
top-2: ice cream parlor (0.139)
top-3: coffee shop (0.138)
top-4: pizzeria (0.085)
top-5: cafeteria (0.078)
GT: lagoon
top-1: balcony interior (0.136)
top-2: beach house (0.134)
top-3: boardwalk (0.123)
top-4: roof garden (0.103)
top-5: restaurant patio (0.068)

Fig. 4. Examples of validation set mis-classifications of the Places365-VGG architecture proposed by [11]. The ground truth label is referred to as GT. It can be observed that the top-5 predictions are, to some extent, related to the GT category, which would suggest introducing multi-label GTs.

One observation arising from both methods presented for static image based scene recognition (see Section 3.1.1 and 3.1.2) is that deeper CNN architectures such as GoogLeNet [21] or VGG [22] are not superior in all cases. For the hybrid multi-scale model combining scene-centric and object-centric networks in [7], experiments using VGG architecture for more than two-scales (two VGG networks) obtained disappointing results, inferior to the baseline performance achieved with one single scale (one network). Since the multi-scale hybrid model entails seven different scales, it can be inferred that VGG becomes noisy when applied on small input image patches.

The T-ResNet architecture, alongside the corresponding introduced dataset, established a new benchmark in the sub-field of dynamic scene recognition. The dataset introduced in [8] poses new challenges by introducing a larger variety of classes (i.e. 20 classes as compared to the previously largest dataset which contains only 16 classes) together with more complex data (i.e. videos with camera motion). T-ResNet obtains state-of-the art results, as per literature [8], for the scenes captured by a static camera; the model exhibits strong performance for classes with linear motion patterns (for example, classes 'elevator', 'ocean', 'windmill farm'), however, for scene categories presenting irregular or mixed defining motion patterns (for example, classes 'snowing' and 'fireworks') the performance is negatively impacted. The authors of [8] observed that, for data points which contain camera motion, T-ResNet presents a decrement in performance. The proposed model exhibits difficulties distinguishing intrinsic scene dynamics from the additional motion of the camera. Further research is required to account for this difference.

The results achieved by the T-ResNet model illustrate the potential of spatio-temporal networks for dynamic scenes. The transformation from a purely spatial network to a spatio-temporal one can succeed on the basis of a very small training set (i.e. only 10% of the YUP++ dataset introduced) as proven by [8]. Well-initialized spatial networks can be efficiently transformed to extract spatio-temporal features, therefore, in theory every network that performs well on static scenes could be easily adapted to dynamic scenes.

	UCF101	HMDB51
	State of the art	
Literature [36]	92.40%	62.00%
Literature [37]	92.50%	65.40%
Literature [38]	93.40%	66.40%
	T-ResNet [8]	
Appearance	85.40%	51.30%
Flow	89.10%	62.00%
Fusion	**93.90%**	**67.20%**

Table 3. Accuracy comparison of the T-ResNet architecture and state-of-the-art models on action recognition datasets (UCF101 [34], HMDB51 [35]) as presented in [8]. The temporal residual units are applied on the appearance and flow networks of a two-stream architecture [39]. The fusion of the two streams achieves the highest accuracy.

The authors of [8] conducted experiments to analyse the generalization ability of the proposed network on action recognition tasks. For the challenge of action recognition, optical flow is a discriminative feature. The proposed T-ResNet architecture presents a healthy performance improvement on two popular action recognition datasets (i.e. UCF101 [34], HMDB51 [35]) in contrast to previously achieved state-of-the-art results as shown in Table 3. The performance boost of T-ResNet, according to [8], is based on the added temporal filters which are 1x1 in spatial dimension and span only 3 instances in time, which come at a very low cost.

5 Applications

In order to illustrate the motivation behind the task of scene recognition, the section at hand addresses several real-life applications in which scene recognition assumes a central role.

5.1 Scene and event recognition

Literature [6] and [40] present the connection between scene and event recognition. Literature [6] presents a pipe-line approach combining object-centric and places-centric CNNs for event recognition. The approach borrows on the idea presented by the method described in Section 3.1.2, which proposes the combination of scene and object knowledge from existing datasets. The challenge exposed by [6] is the classification of 50 cultural events around the world; the events are captured by single images, however the contents of the images are highly detailed and dynamic (usually expressing movement). The authors of [40] propose a recognition tool for events in photo albums. They extract deep scene and object features directly from the image using a CNN architecture. Consequently, the extracted features are analysed with a probabilistic graphical model (PGM) which gives the event classification. The model integrates feature relevance in order to managed the increased complexity of multi-scene images.

5.2 Scene recognition in egocentric photo-streams from lifelogging

Lifelogging is the process of documenting one's daily activities by wearing a first-person narrative camera. The visual data capture is referred to as egocentric data. Egocentric data usually implies low image quality. Literature [41, 42, 43] deals with static and dynamic visual egocentric data for identification of personal scenes. Recognizing personal context has useful applications in stress monitoring, as well as detecting daily routine. Another closely related task is recognition of daily activities.

The task of scene recognition is furthered detailed in literature [44] which presents a hierarchical classification approach for food-scenes. The challenge in this case is represented by the high similarity between the food-scenes. This classification tool is able to categorize a taxonomy of 15 different food places (e.g. supermarket, restaurant, coffee shop, market outdoor). This can be potentially applied in nutritional medicine by aiding the process of discovering eating routines in a person's life. This could assist nutritionist in offering personalized treatment for a healthier nutritional lifestyle.

5.3 Scene recognition for robot localization

Indoor localization for mobile robots is one of the emerging application scopes of scene recognition. According to the authors of [45], in the following two decades, every household could own a social robot employed for housekeeping, surveillance or companionship tasks. Literature [46] and [45] propose the VGG CNN architecture, either with transfer learning or fine-tuning, as the means for scene recognition for service or social robots localization. Another complementary approach is presented in [47]. In this case, an object-centric approach is taken: based on the objects detected, a semantic map of the scene is created in order to facilitate the indoor localization. The authors of [48] introduce a novel, more compact CNN architecture, which achieves state-of-the-art classification results. The model shows a high level of robustness, since it accounts for the viewpoint and various lighting conditions which impact the quality of the scene image. Moreover, the compactness of the introduced CNN model facilitates its embedding in the robot's own architecture.

6 Conclusion

In this work, we compared three state-of-the-art techniques based on CNNs for scene recognition. Furthermore, we presented some of the applications of scene recognition to emphasize the importance of this topic.

For the task of static scene recognition, the combination of scene-centric and object-centric knowledge has proven superior to only considering the scene context. Moreover, the novel availability of large, exhaustive datasets, such as the Places Database, is offering significant support for further research for the challenge of place recognition.

Dynamic scene recognition reached new state-of-the-art performance through the approach of adapting spatial networks to the task, transforming the network to also consider the temporal aspect of the scenes. These emerging spatio-temporal networks are suitable for video data captured with a static camera, however it still faces difficulties in the case of added camera motion.

To conclude, we argue that the main factor to consider is the type of data on which recognition and classification is applied. Since the task of scene recognition is not entirely subjective due to the nature of the scene images and the scene categories overlap, no one particular method can be generalized to all scene recognition tasks.

We hope this paper will inspire and aid professionals in making an informed decision about which approach best fits their scene recognition challenge.

7 Future Work

The research presented in this paper is not exhaustive, analysing only three approaches to scene recognition. Moreover, the comparison of the methods is relative to each other, since currently there is no baseline method that can generalize to all scene recognition tasks. For future work, we suggest furthering the research to include more perspectives on the problem at hand. Some methods which have the potential of challenging the current state-of-the-art are presented in [49] [50] [51] [52]. Moreover, we think a clear demarcation between static and dynamic scene recognition should be made for more detailed comparisons.

Acknowledgements

The authors wish to thank Estefanía Talavera Martínez for providing her expert opinion on the topic of scene recognition.

References

[1] L.-J. Li, H. Su, Y. Lim, and L. Fei-Fei, "Objects as attributes for scene classification," in *European conference on computer vision*, pp. 57–69, Springer, 2010.

[2] A. Bosch, X. Muñoz, and R. Martí, "Which is the best way to organize/classify images by content?," *Image and vision computing*, vol. 25, no. 6, pp. 778–791, 2007.

[3] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 487–495, Curran Associates, Inc., 2014.

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2012.

[5] M. P. Kumar and D. Koller, "Efficiently selecting regions for scene understanding," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3217–3224, IEEE, 2010.

[6] L. Wang, Z. Wang, W. Du, and Y. Qiao, "Object-scene convolutional neural networks for event recognition in images," *CVPR, ChaLearn Looking at People (LAP) challenge*, 2015.

[7] L. Herranz, S. Jiang, and X. Li, "Scene recognition with cnns: objects, scales and dataset bias," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 571–579, 2016.

[8] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Temporal residual networks for dynamic scene recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4728–4737, 2017.

[9] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Dynamic scene recognition with complementary spatiotemporal features," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 12, pp. 2389–2401, 2016.

[10] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Bags of spacetime energies for dynamic scene recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2681–2688, 2014.

[11] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[13] 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 *European conference on computer vision*, pp. 740–755, Springer, 2014.

[14] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2169–2178, IEEE, 2006.

[15] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 413–420, IEEE, 2009.

[16] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3485–3492, IEEE, 2010.

[17] N. Shroff, P. Turaga, and R. Chellappa, "Moving vistas: Exploiting motion for describing scenes," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1911–1918, IEEE, 2010.

[18] K. G. Derpanis, M. Lecce, K. Daniilidis, and R. P. Wildes, "Dynamic scene understanding: The role of orientation features in space and time in scene classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1306–1313, IEEE, 2012.

[19] L. Wang, S. Guo, W. Huang, Y. Xiong, and Y. Qiao, "Knowledge guided disambiguation for large-scale scene classification with multi-resolution cnns," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 2055–2068, 2017.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

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

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[24] M. Marszalek, I. Laptev, and C. Schmid, "Actions in context," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2929–2936, IEEE, 2009.

[25] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals,

R. Monga, and G. Toderici, “Beyond short snippets: Deep networks for video classification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4694–4702, 2015.

[26] D. Park, C. L. Zitnick, D. Ramanan, and P. Dollár, “Exploring weak stabilization for motion feature extraction,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2882–2889, 2013.

[27] Y. Poleg, A. Ephrat, S. Peleg, and C. Arora, “Compact cnn for indexing egocentric videos,” in *2016 IEEE winter conference on applications of computer vision (WACV)*, pp. 1–9, IEEE, 2016.

[28] Z. Xu, Y. Yang, and A. G. Hauptmann, “A discriminative cnn video representation for event detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1798–1807, 2015.

[29] M. Thorpe and Y. van Gennip, “Deep limits of residual neural networks,” *arXiv preprint arXiv:1810.11741*, 2018.

[30] A. Veit, M. J. Wilber, and S. Belongie, “Residual networks behave like ensembles of relatively shallow networks,” in *Advances in neural information processing systems*, pp. 550–558, 2016.

[31] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[32] G. Patterson and J. Hays, “Sun attribute database: Discovering, annotating, and recognizing scene attributes,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2751–2758, IEEE, 2012.

[33] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva, “Sun database: Exploring a large collection of scene categories,” *International Journal of Computer Vision*, vol. 119, no. 1, pp. 3–22, 2016.

[34] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.

[35] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, “Hmdb: a large video database for human motion recognition,” in *2011 International Conference on Computer Vision*, pp. 2556–2563, IEEE, 2011.

[36] X. Wang, A. Farhadi, and A. Gupta, “Actions˜ transformations,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2658–2667, 2016.

[37] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Convolutional two-stream network fusion for video action recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1933–1941, 2016.

[38] R. Christoph and F. A. Pinz, “Spatiotemporal residual networks for video action recognition,” *Advances in Neural Information Processing Systems*, pp. 3468–3476, 2016.

[39] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in neural information processing systems*, pp. 568–576, 2014.

[40] S. Bacha, M. S. Allili, and N. Benblidia, “Event recognition in photo albums using probabilistic graphical models and feature relevance,” *Journal of Visual Communication and Image Representation*, vol. 40, pp. 546–558, 2016.

[41] A. Furnari, G. M. Farinella, and S. Battiato, “Recognizing personal contexts from egocentric images,” in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 1–9, 2015.

[42] A. Furnari, G. M. Farinella, and S. Battiato, “Recognizing personal locations from egocentric videos,” *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 1, pp. 6–18, 2016.

[43] A. Furnari, G. M. Farinella, and S. Battiato, “Temporal segmentation of egocentric videos to highlight personal locations of interest,” in *European Conference on Computer Vision*, pp. 474–489, Springer, 2016.

[44] E. T. Martinez, M. Leyva-Vallina, M. K. Sarker, D. Puig, N. Petkov, and P. Radeva, “Hierarchical approach to classify food scenes in egocentric photo-streams,” *IEEE journal of biomedical and health informatics*, 2019.

[45] K. M. Othman and A. B. Rad, “An indoor room classification system for social robots via integration of cnn and ecoc,” *Applied Sciences*, vol. 9, no. 3, p. 470, 2019.

[46] P. Wozniak, H. Afrisal, R. G. Esparza, and B. Kwolek, “Scene recognition for indoor localization of mobile robots using deep cnn,” in *International Conference on Computer Vision and Graphics*, pp. 137–147, Springer, 2018.

[47] D. Chaves, J. Ruiz-Sarmiento, N. Petkov, and J. Gonzalez-Jimenez, “Integration of cnn into a robotic architecture to build semantic maps of indoor environments,” in *International Work-Conference on Artificial Neural Networks*, pp. 313–324, Springer, 2019.

[48] H. Baumgartl and R. Buettner, “Development of a highly precise place recognition module for effective human-robot interactions in changing lighting and viewpoint conditions,” in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[49] H. Seong, J. Hyun, H. Chang, S. Lee, S. Woo, and E. Kim, “Scene recognition via object-to-scene class conversion: end-to-end training,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, IEEE, 2019.

[50] L. Chen, W. Zhan, W. Tian, Y. He, and Q. Zou, “Deep integration: A multi-label architecture for road scene recognition,” *IEEE Transactions on Image Processing*, vol. 28, no. 10, pp. 4883–4898, 2019.

[51] V. Peltonen, J. Tuomi, A. Klapuri, J. Huopaniemi, and T. Sorsa, “Computational auditory scene recognition,” in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. II–1941, IEEE, 2002.

[52] A. A. Rafique, A. Jalal, and A. Ahmed, “Scene understanding and recognition: Statistical segmented model using geometrical features and gaussian naïve bayes,” in *IEEE conference on International Conference on Applied and Engineering Mathematics*, 2019.

Deep learning in oncology for predicting cancer radiotherapy treatment outcome – A survey

Jeroen G. S. Overschie and Hichem Bouakaz

Abstract— A commonly used treatment for cancer is radiation therapy. However, patients potentially diagnosed with cancer undergoing radiotherapy suffer from numerous unwanted side-effects. Therefore, it is at all times desired to apply only the minimally required radiation dosage. A treatment outcome prediction can be made by analyzing pre-treatment medical imaging to infer a suitable radiation dosage. But this is a difficult task: every individual tumour is different and hence a one-size-fits-all solution is not sufficient. In conventional methods, experts analyzed medical data manually, which does not always yield optimal results due to human- bias or imprecision.

Instead, systematic analysis can yield better results. Deep Learning is a branch of machine learning, in which a multitude of neural network layers are applied to make image-analysis and classification possible. This technique can be used to analyze tumour CT imaging, with a prevalent implementation known as a Convolutional Neural Network (CNN).

In this survey, various methods of applying Deep Learning, CNN's specifically, in the field of oncology are explored to predict radiotherapy treatment outcome for cancer patients. Three methods are examined: (1) a CNN for H&N cancers, (2) a 3D rCNN for H&N cancers and (3) a 3D CNN for lung cancers. Opportunities are present in automating segmentation, 3D CNN and the use of more data sources. Limitations arise accordingly with higher demands for processing power and larger dataset size. Overall, the use of Deep Learning to better predict cancer radiotherapy treatment outcomes has promising results.

Index Terms—Cancer, Oncology, Radiation therapy, Treatment Outcome Prediction, Prognosis, Machine Learning, Deep Learning.

1 Introduction

Cancer remains to be one of the leading causes of death in the modern world [20]. Along with Heart Disease, this disease is the most prevalent cause of death to ageing populations. Therefore, good treatment strategies should be developed, which often involve radiotherapy treatment. Radiotherapy treatment, however, has many undesired side effects. Depending on the body region to which radiotherapy treatment is applied, patients experience skin problems, fatigue, hair loss and sickness [5]. Therefore, to minimize undesired side effects as much as possible, radiotherapy treatment dosage should be kept to a minimum level, while still retaining treatment effectiveness. For this reason, thorough pre-treatment patient analyses should be conducted, to determine exactly what dosage of radioactivity is sufficient for the patient. The pre-treatment analysis consists out of assessing how a specific tumour will react to the radiotherapy treatment, to then be able to determine an individualized dosage best suitable for the patient. This process is known as radiotherapy treatment outcome prediction.

Many methods for predicting radiotherapy treatment outcomes have been proposed, e.g. by focusing primarily on the use of image biomarkers to determine an effective dosage [21] or by calculating risk and survival estimates using logistic regression [17]. With advances in computer processing power, new possibilities arose to do more thorough processing on image-modality output. For example, CT-scan outputs produce *voxels* (3-dimensional, volumetric pixels) with a resolution that is such, that many possibly interesting features remain hidden from the human eye to distinguish [16]. Thus lie the opportunity for machine processing and machine learning from the images.

Recent advancements in the machine learning field led to the development of *Deep Learning*, where, in contrast to other machine learning methods, the entire image with all its raw pixel values is used as input to the network. Alternative methods for processing images would first perform a feature extraction step, to reduce the dimensionality of the data drastically. In Deep Learning, however, the raw input data is sent through multiple layers before performing a final prediction. In this way, more complex and previously unseen image features can be constructed, opening up opportunities to make better treatment outcome predictions and advance the field of oncology.

The primary research goal of this survey is to construct an overview of applying Deep Learning in the field of Oncology and to assess effectiveness of multiple methods of using Deep Learning for predicting cancer radiotherapy treatment outcome. Three methods are thoroughly examined, to then be able to assess their strengths and weaknesses. Given enough analysis, a comparison can then be made between the three methods. To our knowledge, the three state-of-the-art methods have not been compared before - therefore identifying its strengths and weaknesses could help guide future researchers to advance the topic. The survey is focused on a technical perspective, highlighting design choices related to the deep learning implementation. The paper is written with background information on deep learning included, such that, given some familiarity with machine learning, clinicians and oncologists in the medical field are also able to understand the relevance of applying Deep Learning to Oncology.

The structure of this survey is outlined as follows. First, the reader is provided with background on the subject and given an overview of computer-based methods used for medical images analysis in Section 2. The methods used in the compared papers are explained and an overall overview of each method is discussed in Section 3. The same methods are then further assessed in terms of advantages and disadvantages in Section 4. Further more, an overall comparison is provided in Section 5. In Section 6 we will elaborate on the findings of the survey. Finally, in Section 7 future work is considered.

2 Background

Disease diagnosis accuracy using medical imaging depends on both image- acquisition and interpretation. Image acquisition has improved over the years, but image interpretation has only recently begun to benefit from the advancement in computer technologies [7]. Moreover, image interpretation by humans is both labour-intensive and may be subject to human bias. Therefore, computerized solutions such as Image Processing and Deep learning are required to improve the diagnosis of diseases.

Many studies have been conducted to make use of advancements in computer technologies to improve cancer diagnoses. Below a brief overview is provided of common methods used for analysing and classifying medical images.

• *J.G.S. Overschie at University of Groningen, MSc student Data Science and Systems Complexity, E-mail: j.g.s.overschie@student.rug.nl.*
• *Hichem Bouakaz at University of Groningen, MSc student Software Engineering and Distributed Systems, E-mail: h.bouakaz@student.rug.nl.*

2.1 Statistical Methods

Statistical methods are based on probability distribution models and generally include two categories; unsupervised and supervised. The unsupervised methods use clustering algorithms such as K-means and fuzzy clustering methods. The supervised approach needs training data, test data, and data labelling. Probabilistic methods like nearest neighbour classifier, decision trees and Bayesian classifier are included in this category [15] [9].

2.2 Support Vector Machines

The *Support Vector Machine* (SVM) is an algorithm for data classification and regression introduced by Vapnic in 1995. It is connected with the statistical learning theory. The SVM algorithm is a learning machine; therefore it is based on training, testing and performance evaluation [12].

2.3 Radiomics

Radiomics is a method that aims at extracting features from medical images using data characterization algorithms through three steps: (1) initial image pre-processing, which uses a variety of reconstruction algorithms such as edge enhancement or contrast stretching, (2) image segmentation by creating areas of interest in the images, (3) feature extraction which includes two types of features; semantic features used to describe areas of interest such as shape, location and agnostic features used to capture heterogeneity through quantitative mathematical descriptors [1].

2.4 Deep Learning

Deep Learning emerged in the computer vision field in late 2012 and became very popular after a Deep Learning approach based on a CNN won in a computer vision competition known worldwide [10]. Deep Learning can be defined as a subset of machine learning: a Deep Learning network is a form of an *Artificial Neural Network* (ANN), with hidden layers between the input- and output layer. The methods of deep learning are representation-learning methods that contain multiple levels of representation, obtained by creating simple but non-linear modules that each transform the representation at one level into a representation at a higher abstract level. This allows for complex functions to be learned. A schematic overview of a typical neural network can be seen in Figure 1.

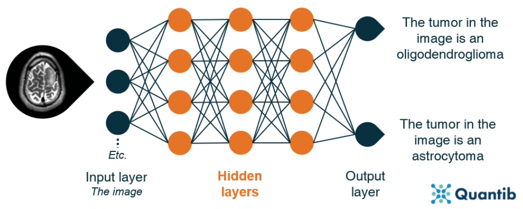

The tumor in the image is an oligodendroglioma
The tumor in the image is an astrocytoma
Etc.
Input layer
The image
Hidden layers
Output layer
Quantib

Fig. 1. Deep neural networks are an extension of "regular" neural networks. In contrast to simple or shallow neural networks they use hidden layers before passing the results to the output layer [2].

In the medical field deep learning is becoming the methodology of choice when it comes to classifying radiological data [13]. The most successful type of model for image classification and analysis to date is a *Convolutional Neural Network* (CNN), which is due to its architecture - making the network both easy to train and more efficient [4]. The size of the CNN network varies, depending on two factors: the learning task at hand and dataset characteristics. There are numerous existing architectures that can be built upon, of which the most commonly used in medical imaging are AlexNet, VGG, ResNet and Inception [13].

3 Methods

Three proposed methods of applying deep learning to improve oncology were selected [11][14][6]. In the following section, each method is explained from a technical perspective, focusing on how each method applies Deep Learning to make an analysis.

3.1 Deep Profiler

Authors Lou et al propose "Deep Profiler" in their paper [11], a multi-task deep neural network. It consists of three fundamental parts: (1) an **encoder** that uses a 3D convolutional neural network for extracting imaging features and constructing a task-specific unique signature, (2) a **decoder** for assessing handcrafted radiomic highlights and (3) a **task-specific neural network** that generates picture signature for treatment result prediction. See Figure 2.

The data used in the study contains 849 tumours corresponding to patients from an internal study cohort and 95 tumours from an independent validation cohort. To assess the performance of the Neural Network, five-fold cross-validation was used, where the dataset was split as 80% for training and 20% for testing with no overlap between the folds. The model was then fine-tuned based on the performance of the validation set. A total of 365 3D radiomics features were extracted from the gross tumour volume.

The features can be grouped into intensity, geometry, texture, and wavelet features. A nested five-fold cross-validation experiment was also used for the examination of the performance of handcrafted radiomics to predict local failure. The performance of the extracted features is computed in the training set individually using the concordance index (C-index) and only the best feature from each of the four groups was selected. The features were combined in a multi-variable model for predicting local failure. The complementary effect of the image score with other clinical risk factors such as *biological effective dose* (BED) and histological subtypes were also assessed. Fine and Gray regression modelling was used to examine the effect of all factors on local failure. First, uni-variate analysis was used to confirm the significance of each individual factor. The model was fit to a subset of the data containing adenocarcinoma and squamous cell carcinoma patients only to evaluate the effect of histological sub-type between the two. This model was then used in combination with the Deep Profiler score and BED, to predict failure and calibrate radiation dose for modulating the risk of local failure.

3.2 3D rCNN model for xerostomia prediction

In the paper from Men et al [14], a model is proposed to accurately predict xerostomia using a 3-dimensional residual convolutional neural network. The 3D rCNN model uses computed tomography planning, 3-dimensional dose distributions and contours as inputs. The model outputs a toxicity probability.

The study included 784 patients enrolled in the Radiation Therapy Oncology. The pre-processing of the data was done using a deep learning-based auto-segmentation approach to contour the parotid and submandibular glands in all patients that are included in this study. Left-right flipping, auto-scaling, and random cropping techniques were used to augment the data to avoid over-fitting. Both 3D images and the corresponding 3D dose distributions were resampled to the size of 2.00×2.00 millimeters and cross-sectional thickness of 5.0 millimeters. A 3D rCNN model was trained for xerostomia prediction where the input is 3D CT Images, 3D dose distribution, and contours (parotid and submandibular glands) and the output is a binary value that represents the prediction of xerostomia. The architecture of the 3D rCNN framework is depicted in Figure 3 and contains 23 layers with the adoption of a residual network to improve the performance. To avoid the early merging of longitude features, various layers follow the last convolutional layer; a pooling layer, a fully connected layer and finally, a softmax loss layer. The neural network contains approximately 20 million weighted parameters.

To train the model, 80% of the data was used. Then, for validation, the study used 10% of the data, with the remaining 10% used to test and assess the performance of the proposed model. Model parameters were fine-tuned during the training.

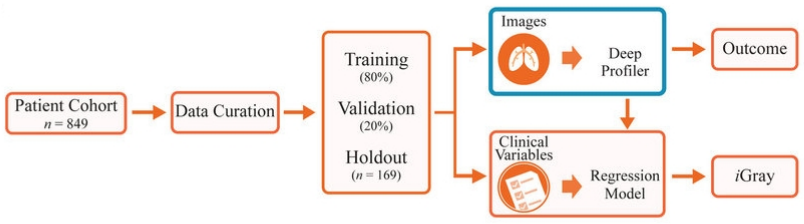


Fig. 2. Deep Profiler study design and neural network architecture. Both a CNN and a regression are used and in combination produce iGray [11].

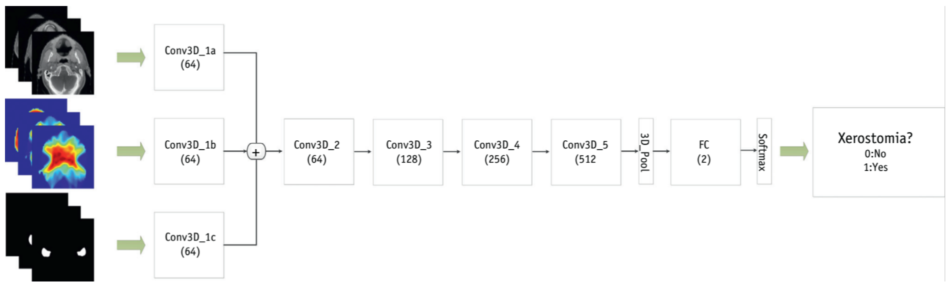


Fig. 3. The flowchart for predicting xerostomia using the 3-dimensional residual convolutional neural network model [11].

3.3 CNN for H&N cancer outcome prediction

In the work from Diamant et al [6], the authors propose the use of a Convolutional Neural Network as the main architecture decision to create treatment outcome predictions in head & neck (H&N) cancers. The CNN used is divided up in four separate parts: (1) **convolution**, (2) **non-linearity**, (3) **pooling** and (4) **classification**. Combining these several operations, a layered structure is obtained: data passes through the pipeline in a predictable manner, where the first (convolutional) layer receives the raw image input.

The convolutional layer will learn and extract features from the given input data. The network uses a certain number of *filters* in the first layer, which will apply a 'sliding window' to the input data. This process will cause the 2-dimensional input data to take on another dimension, which contains the convoluted versions of the image. The number of filters in this first network layer can be of various sizes (for example, one could use 64 filters), influencing the amount of image features the network is able to extract from novel images.

Secondly, the non-linearity layer is applied. A choice for a desirable operation is to be made in order to 'activate' the data produced by the convolutional layer-mapping it to suitable output values to model real-world data accurately. This paper uses a *parametrized rectified linear unit* (PReLU), which slightly differs from the popular *rectified linear unit* (ReLU) in the way that it allows for some fraction of negative input values to pass through the network.

Thirdly, the pooling operation is applied. Pooling allows for reducing the overall size of its input. Because the given input data can take a great spatial size after the convolutional operation, in order to retain computational efficiency, a filter is applied to reduce the high dimensionality. Also, image features are still bound to some specific location on the data - pooling helps make the data location-invariant. This paper uses a method called
enquotemax-pooling, specifically. It works by taking the maximum value of every small 4 by 4 input data patch and replacing the entire patch by this maximum value.

Finally, the last classification layer is applied. The constructed feature maps that are output from the previous layers are taken and are combined using some sigmoidal activation function [8]. The classifier is then able to generate a single output value, which in the case of this paper is an estimation of pre-treatment risk in cancer patients undergoing radiotherapy treatment. The entire CNN network architecture is depicted in Figure 4.

The way the CNN was designed promotes generalization because of its simple architecture. As well, the risk of over-fitting is reduced by not making the model overly complex. A notable remark is to be made of the input layer size, which is 512×512, meaning that some CT image of standard format can be processed by the network without large amounts of pre-processing. In the subsequent layers more convolutional blocks are presented, with each block containing a convolutional layer, max-pooling layer and a PReLU layer. Every added block makes for more ability in the network to uncover more complex features - which is exactly the power of a CNN compared to a human: the CNN can delve 'deeper' into the image, discovering features that are invisible to a human. After these three convolutional layers two fully connected layers are presented, with lastly a PReLU.

4 Strengths and Weaknesses

Various strengths and weaknesses of every method are explored and the effectiveness of the method as described in the paper itself; often assessed in a paper by comparing its own results to competing existing methods using the same dataset.

4.1 Deep Profiler

The results obtained from the study show that Deep Profiler can accurately predict treatment failures across diverse clinical settings and distinct CT scanners. The multivariate models that included Deep Profiler and clinical variables proved to be superior to the classical radiomics which disregards image information outside of the gross tumour volume and the 3D volume, with C-index score of 0.71 (95% CI 0.67-0.77) compared to the classical radiomics regularisation with C-index score of 0·68 (0·63–0·74) and 3D volume with C-index score of 0·66 (0·60–0·71). Detailed results are depicted in Table 1. The agreement between the observed outcomes and the multivariable model with iGray and BED was examined by calculating calibration curves. The curve was obtained by plotting the average predicted probability at 1, 2, or 3 years after radiation treatment versus cumulative incidence curve estimates of the actual outcome. The resulting curve shows that the model can accurately predict the outcome of the treatment. As shown in Figure 5.

The advantages of the method include implementing a deep neural network, Deep Profiler, that combines both CT images and clinical data to predict failure for patients treated with radiotherapy. The derived iGray can be used to provide a patient-specific dose that reduces the probability of treatment failure to below 5%. Deep Profiler is also

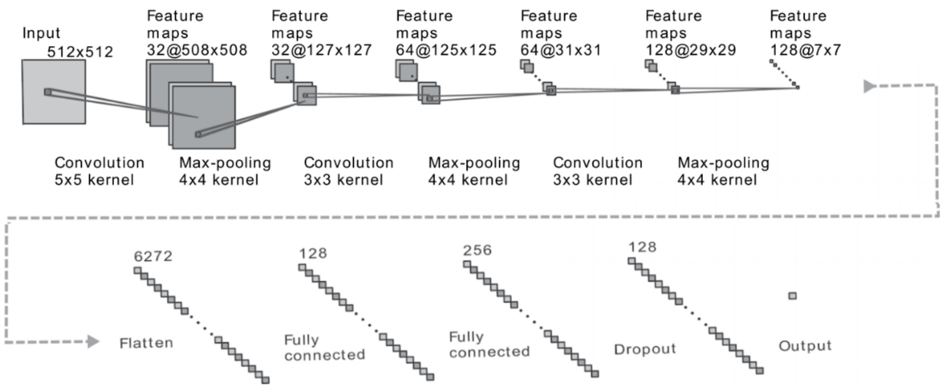


Fig. 4. CNN network architecture for making a pre-treatment risk assessment, given some imaging modality input image, e.g. a Computed Tomography image [6].

	C-index	p value
Deep Profiler	0.71	1 (ref)
2-D CT size	0.61	2.05×10^{-22}
Maximum 3D diameter	0.66	1.06×10^{-20}
Three-dimensional volume	0.67	9.78×10^{-24}
Classical radiomics (feature selection)	0.65	4.23×10^{-25}
Classical radiomics (regularisation)	0.68	1.18×10^{-10}

Table 1. Deep Profiler prognostic performance on various models. Metric used is the *Concordance-index*, which is a generic index for validating prediction ability of some survival model. Compared predictions all use the same patient cohort. The proposed learning-based Deep Profiler framework is superior to classical radiomics features, with a C-index score of 0.71, compared to lower values [11].

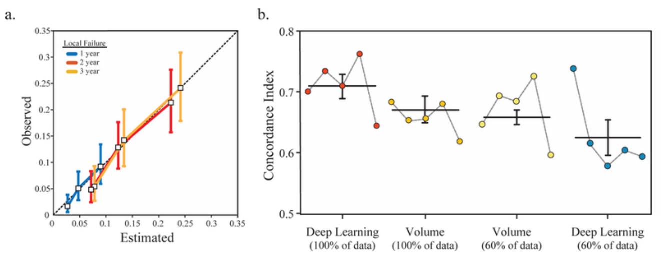


Fig. 5. Model performance and the scalability of the deep neural network. Figure (a) on the left shows the estimated vs observed local failure, where estimations were done using iGray and BED (biologically effective dose). A black line represents a reference perfect score. The length of any bar depicts the 95% CI level. Figure (b) on the right shows model performance using the full 100% of the data and using only a subset of 60%, illustrating potential model scalability [11].

superior to deep learning used alone due to the combination of CT-images and clinical data to derive an accurate iGray which estimates the biologically effective dose required to achieve local control. It became apparent in the study that voxels which are most deterministic for treatment failure are located within the physician-contoured volumes (gross tumour volumes or planning target volumes). Such voxels are completely ignored in the classical radiomics. The authors of the paper have concluded that predictive features can be learned from CT-Images. Additionally, the obtained results have many other clinical implications. Deep Profiler could find image-distinct subpopulations with differential sensitivity to radiotherapy. It provides an integrated method that combines images and established clinical variables to individualize radiation dose.

However, the study has limitations. Firstly, the dataset presents several limitations; (1) the small size of the dataset used in the study and (2) the lack of stratification of the dataset into more homogeneous populations such as cancer subtypes, clinical stage and the use of systemic adjuvant therapy. Secondly, manual annotation is used which could bias feature extraction. Thirdly, the potential causes of bias that are not fully accounted for and the explicit population heterogeneity in the datasets. Finally, the study did not take into account normal tissue toxicity.

4.2 3D rCNN model for xerostomia prediction

The aim of the paper is to check whether using a 3-dimensional (3D) residual neural network (rCNN) can be used to guide radiation therapy through predicting xerostomia to reduce toxicity that results in most of the times Xerostomia. To check the effectiveness of the combined model the results were compared to both a 3D rCNN without contour, a 3D rCNN model without CT-images, a 3D rCNN model without the dose. Additionally, the model was compared to two classical logistic regression models where one used clinical variables and the other did not. The results showed the superiority of the combined rCNN model AUC score 0.84 (0.74-0.91) to the 3D rCNN models without one of the three inputs (CT-images 0.78 (0.67-0.88), contours AUC score 0.82 (0.72-0.90), dose AUC score 0.70 (0.58-0.80)). The combined rCNN model was also superior to both logistic regression model without clinical data (AUC score of 0.84 vs 0.68) and logistic regression model with the clinical data(AUC score of 0.84 vs 0.74). Table 2 shows the results in detail.

Method	Acc	F-score	AUC (95% CI)
3D rCNN	0.76	0.7	0.84 (0.74-0.91)
3D rCNN without contour	0.74	0.68	0.82 (0.72-0.90)
3D rCNN without CT	0.73	0.69	0.78 (0.67-0.88)
3D rCNN without dose	0.65	0.56	0.70 (0.58-0.80)
LR without clinical vars	0.56	0.57	0.68 (0.56-0.80)
LR with clinical vars	0.64	0.6	0.74 (0.64-0.84)

Table 2. Results of xerostomia prediction obtained after applying different methods on the test set. Abbreviations: 3D rCNN Z 3-dimensional residual convolutional neural network; AUC Z area under the curve; CI Z confidence interval; CT Z. computed tomography; LR Z logistic regression [14].

Advantages of the method proposed in the paper include the aim to introduce a fully automatic framework. Compared to other studies that

	Method	DL architecture	Cohort size	Tumor location	Segmentation	Parameters	Model output
Diamant et al. (2019) [6]	DL	2D CNN	784	Head and Neck	Pre-segmented	CT images	Probabilities for DM, LRF, OS
Men et al. (2019) [14]	DL	3D rCNN	300	Head and Neck	Automatic	CT images, Contours, Radiotherapy Dose Distribution	Toxicity probability
Lou et al. (2019) [11]	Radiomics + DL + RM	3D CNN	849	Lungs	Hand-crafted	CT images, Clinical variables	Local failure probability, recommended dosage

Table 3. Comparison of the methods used in the three papers. Abbreviations used: DL = *Deep Learning*; RM = *Regression Model*; DM = *Distant Metastasis*; LRF = *Loco-Regional Failure*; OS = *Overall survival.*

use time-consuming handcrafted feature extractors, a deep-learning-based prediction model that can extract features in an efficient way was used. Using auto-segmentation and contouring has another advantage which is the potential to reduce inter- and intra-observer variations in the delineation. The study also provided a comparison between using 3D CT planning images 3D rCNN without contour, 3D rCNN without CT and 3D rCNN without dose, this comparison shows that high radiation therapy dose is the main cause of xerostomia since the model without the dose performed the worst 0.70 AUC score (0.58-0.80). Compared to the LR model the 3D rCNN model could extract both radiomic and dosiomic features automatically.

Disadvantages of this study include the need for large datasets to train deep neural networks. Additionally, the study does not include clinical variables which can play an important role in the toxicity profile.

4.3 CNN for H&N cancer outcome prediction

In this paper, a hypothesis is set that using CNNs could enhance the performance of traditional radiomics. The paper compared its method with a traditional radiomic framework applied to the same dataset or ’patient cohort’, both in predicting Distant Metastasis. Given the same prediction target, the discussed conventional method [19] achieves an AUC score of 0.86 whilst the newly proposed method using a CNN achieves a score of 0.88. Better results are obtained when combining the newly proposed method with the conventional method. An AUC score of 0.92 is achieved. The combined model was only implemented for *DM*, which is because of the inability for the traditional radiomic approach to find strong individual features for *LRF* and *OS*. See all results in Table 4.

	AUC (Area Under the Curve)				Combined model AUC
	Central slice	Superior slice	Inferior slice	Vallières et al.	
DM	0.88	0.88	0.88	0.86	0.92
LRF	0.65	0.63	0.64	0.50	-
OS	0.70	0.68	0.67	0.65	

Table 4. Results on validation set from Diamant et al [6]. Both Vallières et al [19] and Diamant et al tested on the same patient cohort. Left column abbreviations represent, respectively, *DM = Distant Metastasis*, *LRF = Loco-regional failure*, *OS = Overall survival.*

Advantages of this method are said to be the fact that the network is trained *de novo*. This means that no secondary machine learning algorithm is used and thus no need for using transfer learning is required. Also, little feature engineering is required in the model; the algorithm is given the full set of un-altered pixel data of the tumour. In this way, no human is telling the algorithm which features are of relevance, rather, the algorithm figures this out itself and reports back to the user. Also, the algorithm is able to explicitly recognize radiomic features, whilst still keeping the model relatively easily interpretable.

Disadvantages of the method include the fact that CT images could have been cropped in such a way, that only the tumour itself would be in the main field of view. Such, many irrelevant voxels are removed. However, this directly opposes a primary method advantage: adding extra preprocessing steps would add much complexity to the framework, contradicting the advantage of keeping the pipeline relatively simple and free of complex feature selection steps. Also, more information could be added. The dimensionality of the CT-scan output imaging was possibly not exploited to its full potential: CT imaging contains 3 dimensions, which, if considered in its entirety, might reveal extra information. Another potentially useful source of information to add is a patient’s *positron emission tomography* (PET) image. Vallieres et al [19] found PET information to be of additional predictive power, so considering this information might improve prediction performance. Another disadvantage mentioned is the fact that the framework only considered the pre-segmented gross tumour volume as input, instead of including all tissue including the surrounding tissue, removing the need for location invariance. Taking the surrounding tissue into consideration could also be of additional value with respect to the prediction. Finally, the combination approach was quite simple: logistic regression was performed to combine the traditional radiomics approach with the newly proposed CNN method. More sophisticated methods (transfer learning) could have been explored to combine the two different approaches, possibly improving performance further.

5 COMPARISON

In previous sections, information about each paper’s method and its strengths and weaknesses was given. In this section, this information is put together to distil essential findings. Main differences and similarities are summarized in Table 3.

As depicted in the table, the selected methods have different model outputs. Whereas Men et al estimates the toxicity level to predict xerostomia, Diamant et al predict the outcome of radiotherapy through estimating the probability of Distant Metastasis, Loco-Regional Failure and Overall Survival. Both Men et al and Diamant et al prediction is for Head and Neck cancer. Lou et al, contrarily, predicts the outcome of radiotherapy for Lung cancer through estimating the probability of Local Failure and computes a recommended radiotherapy treatment dosage using a Regression Model, as depicted in the ’Method’ column of the table.

Segmentation methods vary across the examined papers. Diamant et al use a dataset where the tumour volume is already segmented, but Lou et al uses data that depended on human experts to segment the volumes manually. As discussed in their paper, handcrafted segmentation is labour intensive and subjected to human bias. The automatic segmentation proposed in Men et al, however, is mentioned as an improvement over handcrafted segmentation.

Different Deep Learning architectures were used; Diamant et al used a 2-dimensional CNN approach, whereas the other two use 3-dimensional approaches. Choosing between a 2D or 3D CNN architecture, one should consider the classical trade-off between accuracy and speed: considering more dimensions requires more computational power to train the model. Whilst Diamant et al only considers three 2-dimensional slices of the tumour volume - which results in considerably fewer data to process than the entire 3-dimensional volume. However, Lou et al mention that considering the entire 3-dimensional volume could have extra predictive power, resulting in possibly more accurate predictions.

Model parameters are constructed using the method's data sources, the main one being CT images for all papers. The additional parameter *Radiotherapy Dose Distribution*, used in Men et al, is primarily used to complement the process of predicting toxicity probability, which is its model output. In Lou et al, clinical variables are used to assist recommended dosage prediction. Men et al, however, no clinical variables are used, but an argumentation according to their results is provided, that adding in clinical variables could improve accuracy.

All studies conducted their experiments on different patient cohorts, as can be seen in the table. Accordingly, all paper mentioned future work improvements related to their dataset. Lou et al mentioned their dataset to be complete and of sufficient size, but their validation cohort limited. Men et al argued that their 3-dimensional CNN approach requires large datasets to function optimally, which is a common challenge in Deep Learning.

6 Conclusion

Given various methods, applying Deep Learning to predict cancer radiotherapy treatment outcome has promising results compared to traditional methods. Opportunities lie in the use of automated segmentation, the use of the full 3-dimensional volume using 3D CNN and mixing in more data sources alongside the CT image, like clinical data. Limitations are increased demands for data processing power and large dataset sizes, which not all institutions possess. Therefore, critical to further improving prediction power is the willingness of researchers and institutions to openly share large datasets.

7 Future work

Several steps can be taken for further improvements based on this research. Relevant points of improvements are discussed below.

First of all, a larger synthesis of research material could be made. In this paper, three methods for applying deep learning in the field of oncology are described and eventually compared. A possible improvement would be to include more sources: since a primary goal of our research is to outline the latest developments of utilizing deep learning in this field, it is reasonable to include more methods available to get a better overview on the subject matter. Supporting this, there indeed exist many more papers in which similar topics were explored [3] [18].

Secondly, a comparison could be made of work that is more related to each other. The three chosen methods all have different research goals and aim to make a different prediction using their networks. This makes the methods hard to objectively compare; primarily, a comparison is made between each method's individual effectiveness given its problem context. An improvement would be to compare papers that use the same dataset and attempt to predict the same attribute.

Thirdly, validation could have been performed on the paper results. This could have been done by using the same dataset for the three papers and compare the results. In such a way, one is able to reproduce the same charts that were supplied in the papers, thereby validating its scientific integrity.

Lastly, another possible improvement is to further complete the usefulness of Deep Learning in Oncology by comparing the novel methods of prediction to more traditional ways of performing the same predictions: by using conventional Machine Learning or some form of assessment using only clinical data. Only then, by also examining a large spectrum of methods including Deep Learning, can the effectiveness of Deep Learning be truly assessed effectively.

Acknowledgements

We would like to thank E. (Estefanía) Talavera Martínez for her help in finding suitable source papers and her constructive feedback in the review - we also like to thank the additional peer reviewers for their time and energy.

References

[1] Radiomics. `https://radiopaedia.org/articles/radiomics`. Accessed: 2020-03-15.

[2] The ultimate guide to ai in radiolog. `https://www.quantib.com/the-ultimate-guide-to-ai-in-radiology`. Accessed: 2020-02-20.

[3] J.-E. Bibault, P. Giraud, and A. Burgun. Big data and machine learning in radiation oncology: state of the art and future prospects. *Cancer letters*, 382(1):110–117, 2016.

[4] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[5] D. P. Dearnaley, V. S. Khoo, A. R. Norman, L. Meyer, A. Nahum, D. Tait, J. Yarnold, and A. Horwich. Comparison of radiation side-effects of conformal and conventional radiotherapy in prostate cancer: a randomised trial. *The Lancet*, 353(9149):267–272, 1999.

[6] A. Diamant, A. Chatterjee, M. Vallières, G. Shenouda, and J. Seuntjens. Deep learning in head & neck cancer outcome prediction. *Scientific reports*, 9(1):1–10, 2019.

[7] A. Hosny, C. Parmar, J. Quackenbush, L. H. Schwartz, and H. J. Aerts. Artificial intelligence in radiology. *Nature Reviews Cancer*, 18(8):500–510, 2018.

[8] B. Karlik and A. V. Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[9] T. A. Lashari and R. Ibrahim. A framework for medical image classification using soft set. 2013.

[10] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[11] B. Lou, S. Doken, T. Zhuang, D. Wingerter, M. Gidwani, N. Mistry, L. Ladic, A. Kamen, and M. E. Abazeed. An image-based deep learning framework for individualising radiotherapy dose: a retrospective analysis of outcome prediction. *The Lancet Digital Health*, 1(3):e136–e147, 2019.

[12] I. G. Maglogiannis and E. P. Zafiropoulos. Characterization of digital medical images utilizing support vector machines. *BMC Medical Informatics and Decision Making*, 4(1):4, 2004.

[13] M. A. Mazurowski, M. Buda, A. Saha, and M. R. Bashir. Deep learning in radiology: An overview of the concepts and a survey of the state of the art with focus on mri. *Journal of magnetic resonance imaging*, 49(4):939–954, 2019.

[14] K. Men, H. Geng, H. Zhong, Y. Fan, A. Lin, and Y. Xiao. A deep learning model for predicting xerostomia due to radiation therapy for head and neck squamous cell carcinoma in the rtog 0522 clinical trial. *International Journal of Radiation Oncology* Biology* Physics*, 105(2):440–447, 2019.

[15] E. Miranda, M. Aryuni, and E. Irwansyah. A survey of medical image classification techniques. In *2016 International Conference on Information Management and Technology (ICIMTech)*, pages 56–61. IEEE, 2016.

[16] V. Ratnaparkhe, R. Manthalkar, and Y. Joshi. Texture characterization of ct images based on ridgelet transform. *ICGST-GVIP*, 8(5):43–50, 2009.

[17] W. Schroth, L. Antoniadou, P. Fritz, M. Schwab, T. Muerdter, U. M. Zanger, W. Simon, M. Eichelbaum, and H. Brauch. Breast cancer treatment outcome with adjuvant tamoxifen relative to patient [...] genotypes. *Journal of Clinical Oncology*, 25(33):5187–5193, 2007.

[18] S. Trebeschi, J. J. van Griethuysen, D. M. Lambregts, M. J. Lahaye, C. Parmar, F. C. Bakers, N. H. Peters, R. G. Beets-Tan, and H. J. Aerts. Deep learning for fully-automated localization and segmentation of rectal cancer on multiparametric mr. *Scientific reports*, 7(1):1–9, 2017.

[19] M. Vallières, E. Kay-Rivest, L. J. Perrin, X. Liem, C. Furstoss, H. J. Aerts, N. Khaouam, P. F. Nguyen-Tan, C.-S. Wang, K. Sultanem, et al. Radiomics strategies for risk assessment of tumour failure in head-and-neck cancer. *Scientific reports*, 7(1):1–14, 2017.

[20] J. Xu, K. D. Kochanek, S. L. Murphy, and B. Tejada-Vera. Deaths: final data for 2007. *National vital statistics reports: from the Centers for Disease Control and Prevention, National Center for Health Statistics, National Vital Statistics System*, 58(19):1–19, 2010.

[21] A. Yaromina, M. Krause, and M. Baumann. Individualization of cancer treatment from radiotherapy perspective. *Molecular oncology*, 6(2):211–221, 2012.

An overview of methods used for automatic detection of social interaction in visual material.

Alessandro Pianese and Tanja de Vries

Abstract— Automatic detection of social interaction is a popular field of research due to the broad range of applications. From mental health to surveillance systems. Moreover, it is used to automatically detect the special moments that should be kept in a day long personal video. This paper goes over the main common methods used in scientific literature to detect social interactions in visual material. We have found that the main structure of an automatic recogniser is usually very similar between models, however, they differ a lot in the specific implementation.

Index Terms—Computer vision, social interaction detection, recurrent neural network.

1 Introduction

Engaging in social interactions is one of the basic needs of individuals and one of the few activities that we tend to perform on a regular basis. In this process we interact and react to the people (and the environment) that surrounds us. The first scientific definition of this process was given by Erving Goffman. [12] Automatic detection of social interaction has a lot of applications, and as a result there is an increasing scientific interest in the topic. [11]

One of the earliest applications of understanding social behaviour in computer science were automated surveillance videos. These systems have to detect, as soon as possible, threatening behaviour. This is however a very difficult task for which a lot of social information is needed. [9]

Another kind of application is in the medical field. The frequency and diversity of social interactions can say a lot about the physical and mental health of the people partaking in it. [16] Thus, an application is for example the analysis of a series of pictures from the life of a person with a mental disorder. This could help psychologist to get a better understanding about the mental disorder. Moreover, it could be the basis of a method to monitor progress of a treatment.

Lastly, people generally want to keep pictures of special moments. However, taking pictures all day distracts from the special moment. Filming the whole day captures the special moments, but also a lot of boring moments. Therefore, research is done on separating the special moments from the moments one does not want to keep. Fathi et all. believe that a frame containing a social interaction is a good indicator for the frame to be worth keeping. [10]

1.1 State of the field

In this subsection an overview is given of the developments in the field of automatic detection of social interaction.

Setti et al.[15] did research on detecting social interactions in images with a similar point of view as surveillance cameras. An example picture is given in Figure 1. Since the camera has a static position, a lot of parameters, such as illumination, are constant. This gives a relatively easy setting to recognize and track individuals and groups. However, due to the camera being external, the people in the picture might be partially hidden behind other people. The researchers used the F-formation to determine interacting groups. The F-formation assumes that people that are close and facing each other are interacting with each other, more information about this method is given in Section 2.4.1. The researches focused on differentiating between different kind of groups with this technique. However, due to the nature of the data, this method was not able to gain insights of the social interaction pattern of a single person.

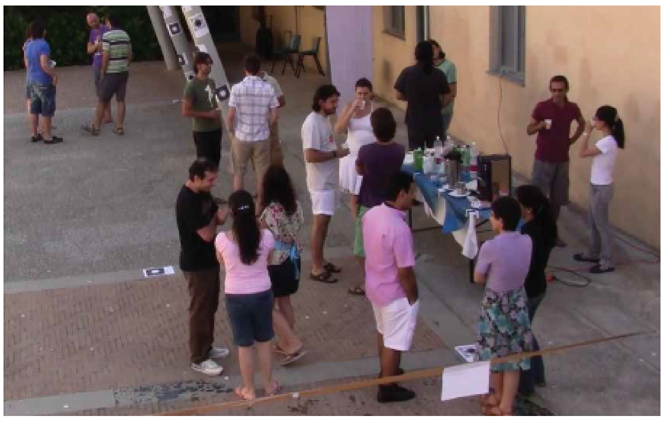


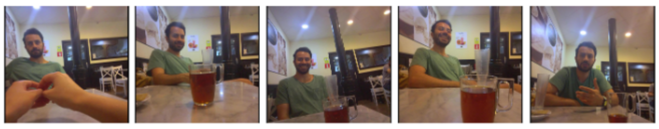
Fig. 1. A picture from an external point of view used in [15]. There are multiple groups of socially interacting people visible.

In 2012, Cristani et al. [9] did research on images from surveillance cameras. The authors discuss the idea of social signal processing techniques on surveillance footage to identifying threatening behaviours. They report that the elements that usually deliver the most useful information are, among others, gestures and postures, and face and gaze direction. Thus, one of their conclusions is that social interaction detection should not only be approached from a computer vision and image processing point of view but also from a social signal processing perspective.

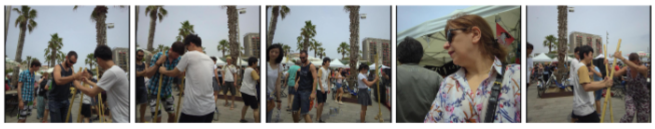
With the recent increasing availability of wearable cameras the option to revisit the problem of detecting social interaction has drawn the attention of researchers.[10] Detection of social interaction in egocentric pictures is very different from the external surveillance cameras approach. This is mostly due to the fact that interactions with the wearer have to be found while the wearer itself is not in the image. A more elaborate discussion on the different aspects of point of view of data is given in in Section 2.1.

Aghaei et al. made a model based on the F-formation, using egocentric photo series. [4] An example of the data they used is given in Figure 2. They found this model did not perform well enough in crowded places where people stand too close to one another without interacting such as in amusement parks. Therefore, they improved their model using emotions recognition. [1] In this model a neutral expression decreases the probability of an individual partaking in a social interaction with the camera wearer. This can be seen in Figure 3.

- *Alessandro Pianese is a CS master student at the RUG, E-mail: a.pianese@student.rug.nl.*
- *Tanja de Vries is a CS master student at the RUG, E-mail: t.r.de.vries.2@student.rug.nl.*

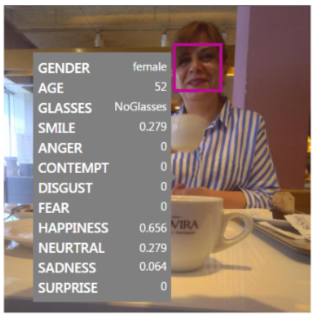


(a) Social interaction

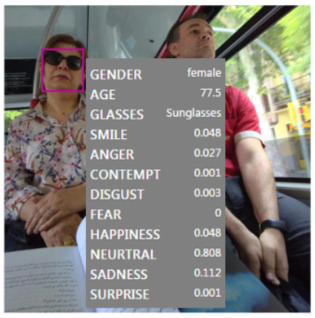


(b) No social interaction

Fig. 2. This is an example of an egocentric photo series, from the EgoSocialStyle data set[2], displaying a social interaction at the top and no social interaction at the bottom.



(a) Social interaction



(b) No social interaction

Fig. 3. This is an example where emotion is used for the detection of social interaction, by Aghaei et al. [2]

1.2 Overview of paper

In this paper we discuss a selection of the approaches that are used in the field. We have tried to give a broad overview of the available methods. Most papers use the same basic structure, so in Section 2 the methods are split in the 5 basic steps. For each step the process is explained and some examples of specific methods are given. In Section 3 a discussion is given on the combination of methods and some of the results are shown. Lastly, in Section 4 we give some concluding statements.

2 Methods

In general an automatic social interaction detector consists of 4 main steps: person recognition, person tracking, group detection and classification. However, the specific methods used to implement these steps differ greatly between research groups. It depends on the goal of the project what methods give the needed information. Moreover, the kind of data that is used has a large impact on the process. In this section we describe some of the methods that are used in this field. We first discuss the available data sets and then we discuss the methods that are used for each main step.

2.1 Data set

The choice of data determines a lot about the methods used. Moreover, it depends on the goal of the project what data is best to use. We already shortly mentioned the difference between pictures from external and first person view. Another point of attention is the amount and quality of frames, varying from one picture to high quality video. In this section we go over the choices that should be made.

As explained in the introduction, in first-person view the interactions with the wearer have to be found even though the wearer itself is not in the image. However, users in a social interaction naturally are oriented towards the others in the interaction. Therefore, the orientation of the wearer is known, but it might be more uncertain than the orientation of the other persons. Moreover, the data will usually contain the interacting people in focus and with more details than in the external view data. An egocentric viewpoint also brings new challenges due to the camera moving. These sometimes abrupt movements can result in quick changing illumination and noise.

When we look at literature regarding this topic, there are two possibilities for first person camera placing, either on the chest or on the head. A chest mounted camera is less distracting for the people in the interaction and has less issues with abrupt movement, since the head is usually changing orientation more often. However, a head mounted camera captures the facing direction of the head instead of body, which might give better results when the other persons are to the side rather than in front of the wearer.

Lastly, the data ranges from a single picture to high quality video. A high quality video gives the most information, but more information usually includes more noise as well. So a higher quality does not automatically give better results. For recording a longer time, a day or even a month, it is more suitable to use a low frame rate photo series.

2.1.1 Available data sets

In this section we will go in more detail for the specific data sets that are mentioned in this paper.

Firstly, A pair of data sets that is publicly available for these operations is the ECO-Group and ECO-HPE data set. This is a first-person view data set that was produced to be employed in [6] as a testing asset. The ECO-Group data set contains high frame video of social interactions in various environments. All footage has been produced from a head mounted camera and, as a results, produced video with substantial background clutter and noise. The ECO-HPE data set, on top of the previous one, adds head pose estimation for all faces found in the frames.

Another first-person view dataset is the EgoSocialStyle set employed in [1]. The aim of this dataset is different since they replaced high quality videos with low frame rate videos. The authors strived to be able to automatically recognise people partaking in social interactions with only a subset of the information previously used. This time the camera was programmed to take pictures once every thirty seconds and was attached to the chest of the person.

Lastly, the authors in [10] made a data set in an amusement park. A subset of 8 of a group of 25 people was wearing a first-person head-mounted camera to record 3 days. This results in a 15 fps video of more than 42 hours. Thus, this dataset contains a lot of noise. Moreover, the frames usually consist of a lot of people.

2.2 Person recognition

The first step needed for the detection of social interaction in streams of images is the recognition of people in said images. This problem can be traced back to object recognition. As described in [19], some of the most common issues in the field are:

- The loss of information due to the 2D projection on the image plane of the 3D world.
- Changes in the scene illumination
- Abrupt changes in the motion of the object
- The usual noise in images

Moreover, these points do not allow for real-time processing of images. Usually for frameworks to obtain results in a reasonable amount of time some assumptions are made. For example, one of the most common assumption is to expect the image in which the object is represented to have an even illumination.

The object recognition pipeline start from identifying contours in images. This can be done by several techniques from the image processing field such as segmentation or point detector. In segmentation, an image is divided into perceptually similar region[19] based on

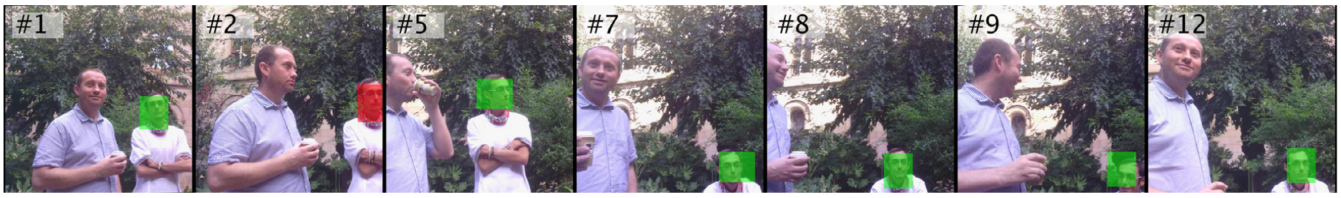


Fig. 4. These are example tracklets found in a series of frames by Alletto et al. [6]. The red box displays the seed and the corresponding boxes are shown in green.

shapes and intensity levels of the image. This way, objects that stick out from the background can be easily picked up. Another approach is to use point detectors. With points detectors we want to identify certain points in images. These points should have some specific features such as constant illumination and invariance to the camera viewpoint. Comparing two images with said technique will return a number of matches that stands for the number of similar points found in both images. Points detectors may be used to recognize shapes in both images and video.

2.2.1 Deformable model

One of the methods for automatic object recognition is deformable models. With the term *deformable models*, we indicate a group of algorithms and techniques that aims to model the variability of a certain class of objects[5], heads and/or human bodies in our case. In [7], the authors give examples of how these deformable models are usually employed. One of the most widespread type of models, is the Point Distribute Model. The PDM represents shapes as a collection of landmark points that are deemed important. This means that certain pixels define a shape that is capable of describing the object we want to recognize. Deformable models are usually characterized by certain parameters that dictate how the algorithm has to approach the model. In [18], the authors propose an algorithm that produce a collection of pixels, called snake, that enclose the object of interest. In this use case, the parameters employed define the tension, rigidity and external forces of the snakes.

2.3 Person Tracking

If more than one picture are used to detect social interaction, the tracking of faces along the different frames is a fundamental step in detecting social interaction. It is relatively easy to do so in a video from an external camera with a static field of view. However, when using an egocentric camera that gives abrupt changes in both field of view and illumination, the problem is much harder to solve. There are multiple methods available to track persons, we go in dept with one of these methods by Aghaei et al.[3]

These authors constructed a tracker that can be used on the EgoSocialStyle dataset. The challenge of this dataset is especially the low frame rate of 2 fpm. Even in the case of no movement noise, having 30 seconds between every picture has as a result that the people in the picture have enough time to be moving around. This increases the possibility that persons in a picture are in a very different spot, then they were in the previous picture.

Since the pictures are in first person view, the people captured in the picture are close. Therefore, the authors decided to track persons only based on their face rather than on their whole body which might not be in the picture. They perform a 4 step tracking and identification process.

Seed and tracklet generation. A convolutional neural network is used to divide the data into segments based on global features. Each segment is individually evaluated for relevance, it is checked to contains trackable persons. In fact, a certain segment is kept if the number of trackable faces is at least half of the number of frames in the segment. After selection, bounding boxes surrounding the faces, called seeds, are collected. For each seed a deep matching network is used to find the corresponding boxes in the other pictures. For this the already found boxes are matched, but it also searches for faces in the neighbourhood of the seed. Thus, it is possible that a face missed in the last stage is nonetheless found at this step in the process.
In Figure 4 an example segment is shown with a seed in red and the corresponding boxes in green. This set of boxes together is defined to be a tracklet. In fact, a tracklet is defined as $T^i = \{t^i_b....t^i_s....t^i_e\}$ where t^i_k is the bounding box corresponding to the seed i and frame k. The seed is found in frame s and the sub-indexes b and e give the frames where it appears first and last, respectively.

Grouping tracklets into bag-of-tracklets. There are multiple tracklets belonging to the same person. Therefore the next step is to group the tracklets and construct one unique extended-bag-of-tracklets (eBoT), denoted by $\mathbb{T}$, for every person in the segment. This is done by computing the similarity of a tracklet to the eBoT. If they are found to be similar the tracklet is added to the eBoT. In the end all tracklets in an eBoT should correspond to the same person. However, there can be some unreliable eBoTs that do not correspond to a person. To minimize the unreliable eBoTs, an eBoT is removed if the ratio between tracklets in the eBoT and frames in the sequence is too low.

Prototype extraction. Since a tracklet is constructed from every found seed there is a lot of overlap in the bounding boxes in an eBoT. Therefore, in this step, eBoTs are used to construct prototype tracklets that should represent all tracklets in the eBoT. The prototype tracklet consists of one bounding box per frame. So, for each frame, the bounding boxes in all tracklets in the eBoT are collected. The prototype bounding boxes are then chosen such that they have maximum intersection with the bounding boxes in the collection. The computed prototype $\hat{T} = \{\hat{t}^i_b....\hat{t}^i_s....\hat{t}^i_e\}$ should be able to identify the face in all frames. However, it also selects a best bounding box when the face is not present in the frame.

Occlusion treatment. In the last step, occlusions and unreliable detections are identified and removed. To do so, the authors define a function Λ that associates the deep matching score of each bounding box to their respective seed. This function is used to calculate the confidence level of a frame k as

$$C_k = \frac{1}{|\mathbb{T}|}\sum_{i=1}^{|\mathbb{T}|}\Lambda(t^i_s, t^i_k),$$

where t^i_s is the seed of the ith eBoT tracklet. The variable t^i_k is the bounding box at frame k and $|\mathbb{T}|$ is the number of frames in the tracklet. This function correctly shows a confidence drop by frames with missing faces and occlusions. A threshold is set on the confidence level. Whenever the confidence of a frame is below this threshold, the frame is removed.

2.4 Groups identification

Once people are identified in streams of images, we can move on to actually make use of this information and find the potential groups in an image. In this sub section we will reports some of the most used

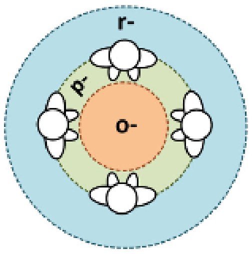


Fig. 5. The three spaces in F-formation. The interaction persons are positioned in the p-space. [15]

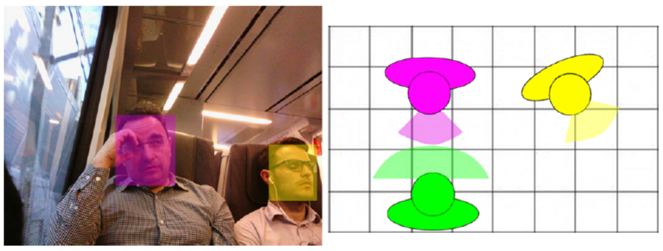


Fig. 6. This is an example of F-formation by Aghaei et al.[4]. The bird-view model at the right shows the F-formation of the scene at the left. From this formation we conclude that the wearer is in a social interaction with the person in pink.

methods in the field starting with the Facing Formation which is often the starting point for all of these methods.

2.4.1 The Facing Formation

Once involved in a social interaction, there is a natural formation people will take position in. Kendon was the first to make a model of this formation, defined to be the Facing Formation, abbreviated to F-formation [13]. Humans in a social interaction all look inwards a void in the middle of the group. In other words, their orientation is towards the other people in the interaction. Moreover, they will have an intuitive distance between each other.

More specifically, the F-formation consists of three spaces as shown in Figure 5. Firstly, the p-space, which is the space consisting of the interacting people. Everyone participating in the interaction has easy and equal access to the empty space between them, called the o-space. Lastly, the interacting people stand with the back to the outside world. People not interaction with the group are not accepted in the p-space, they are positioned in the r-space.

Kendon [14] defines the F-formation as follows:

An F-formation arises whenever two or more people sustain a spatial and orientational relationship in which the space between them is one to which they have equal, direct, and exclusive access.

The F-formation is used to detect groups that are socially interacting. More specifically, it is assumed that interacting people and people standing in F-formation are more or less interchangeable. So, if a group is found to stand in F-formation one can conclude they are interacting. Often, as mentioned before, researchers use the F-formation as a first selection and build their research upon it. To find if a group of people are in F-formation the orientation of each person and the distance compared to the group are computed. If they are oriented towards each other and they are close together, it is concluded that the group is in F-formation.

The F-formation is used by Aghaei et al.[4]. In this article orientation is based on the face orientation instead of the whole body. The face pose is determined using an algorithm by Zhu et al. [20]. The result is shown in Figure 6.

2.4.2 Markov random field

Another example of group identification is given by the authors in [10]. They calibrate the camera by making pictures of people with known orientation and position with respect to the first person. This information is then used to make a formula to compute the 3D location of each person from a 2D image. Some parameters are for example the height of the head and the image coordinates. Moreover a first estimate of the orientation is computed of each person. The orientation is assumed to have a higher probability to be towards another person and also to have a higher probability to be towards the same spot as the orientation of another person. This information is all combined in a Markov random field to determine the final orientations. The persons that are looking at each other or to the same object are then said to be in a group.

2.4.3 Structured groups of people

Choi et al. in [8], propose a new approach to groups and interaction classification. They introduce the concept of people forming *structured groups* while partaking in social interactions. Examples of these groups can be a line of people in front of a shop, two persons sitting on a bench or a group of people sitting side by side on multiple rows like we would observe during a lecture. Their idea is to combine these groups to identify the type of interaction taking place.

A line of people behind a group of two people, in which one is paying, implies that people are shopping. Also a line of people sitting side by side and one person in front, speaking, means that some kind of lecture is taking place.

This technique is applied to picture of groups taken by an external perspective and, therefore, is sensitive to the usual issues of obstruction, changes in perspective and many more.

2.5 Classification

Now that we know how to find and track persons and how to find potential groups, we need a method to combine all the collected information and give a final decision on whether a social interaction is present in the data. Since the data consists of time series, a Recurrent Neural Network is a natural choice. However, we have also find researchers to use other types of classifiers. We first describe one method using a Recurrent Neural Network, then we discuss another method using a Support-Vector Machine.

2.5.1 Recurrent Neural Network

Aghaei et al. in [4] employ recurrent neural networks (RNN) to perform classification of the feature vectors extracted from photo frames. Specifically, they used a Long Short Term Memory (LSTM) network to take advantage of the temporal evolution of the scene. This is due to the recurrent structure of the network where hidden units work as memory cells and there are cycling connections within nodes. An LSTM network was preferred since an RNN isn't capable of learning task with a substantial temporal gap between events due to the presence of only short term memory cells.
Being able to recognise and classify social interactions with time gaps inside the sequences is not a trivial operation since the LSTM has to learn to ”protect the memory cell contents from against even minor internal state drift” [4]. Also, hidden units use the information stored inside other memory cells to decide whether or not to access the information stored inside them.
For performing the actual training of the network, they employed a backpropagation function with a time element (BPTT) to extend the series of derivatives calculated with the chain rule in the standard backpropagation method.

2.5.2 Support-Vector Machine

Alletto et al. in [6] defined a new way to determine the relation between two individuals that may be engaging in a social interaction. They present the relation between two persons r and t as $\phi_{rt} = (d, o_r, o_t)$ where d is the distance between the individuals, o_r and o_t are, respectively, the rotation needed to look at the other person.

After having defined these correlations, they explore the idea of using a simple correlation clustering algorithm to perform the grouping.

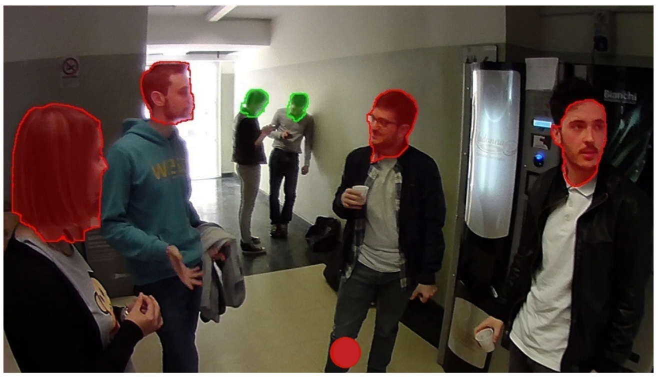


Fig. 7. This is an example of group formation by Alletto et al.[6]. One group is shown in green and the group of the camera wearer (red dot) is depicted by red.

They employ a correlation matrix W where $W_{rt} > 0$ if the two individuals are engaging in a social interaction and $W_{rt} < 0$ if they are not. To identify a social interacting group using this correlation matrix we simply have to look for the subset of individuals $y = r_1, r_2, ..., r_n$ that has the biggest sum of all related $W_{r_i r_j}$ where $i \neq j$.

To reduce the complexity of the problem however, they subsequently modified the representation of the data to be a structured graph instead of a matrix. On top of this they employed a structural SVM to learn the discriminant function that can match all the pairs (r,t) of individuals with the right group formation y. In this process, assigning two non interacting individuals to a determined group, will increase the error used for measuring. Due to the relevance that the error possesses, choosing the right loss function is critical.
The authors found the MITRE Score[17] as an appropriate scoring measure since its application is, according to the authors, in many ways similar to the classification of social interaction. An example of the result of this procedure can be seen in figure 7. Here we can easily observe how the faces of people partaking in different social interaction are highlighted by different colors. The red circle at the bottom represents the person wearing the camera.

3 Discussion

Up to this point, we have presented methods for the detection of people and the classification of groups using images. Most of them considerably overlap and together they can be combined in a very powerful framework. However, these methods have a computational load that have to be taken into account. Another method we mentioned, the one used by [4] with 2 frames per minute video, took that into account and tried to recover the same information as in other experiments while using less data. Now we will try to highlight how these methods may interact among themselves to provide valuable information on streams of images.

We have seen that the F-formation is widely used by a range of papers, to identify individuals partaking in the same social interactions. It is an easy method to implement because the only needed variables are the orientations and relative distances of the subjects. However, the method is a simplification of a much more complex model. It is not always correct to assume that people in F-formation are interacting. An easy counterexample is people in a public transport context: they will often face each other and stand close together, also inside the p-space, even though they are not interacting. On the other hand, people can also interact without showing a clear F-formation. For example having a conversation while walking or sitting next to one another is easily missed.

An addition to this model, that could keep it simple, may be to consider the emotion of faces in the scene. Recognising people with neutral emotions may really help to correctly classify people as not interacting while being in the F-formation. An example of this method by [1] is shown in Figure 3. However, this also adds another layer of computation to the pipeline so it is not always employed.

To reinforce the F-Formation model, we could also add probabilities extracted from a Markov random field constructed as described in sub-section 2.4.2. With a Markov random field, it was also possible to study groups from five to ten people and to gain some information on the interaction. In fact it is possible to understand what kind of interaction is going on by checking who and how often people speak. This. however, would add another layer of complexity, if added on top of the F-Formation, but it would help with false negatives such as people far away from the group or cases where the o-space, and therefore the p-space, is harder to define. An example could be a person that is considered to be outside of the p-space and, therefore, out of the interaction. However, when we consider its gaze, we could discover that he is looking in the same direction as the others, partaking in a type of interaction.

Another attempt at classifying social interactions by looking at groups of people was presented in sub-section 2.4.3. The approach starts from the same premises of the other, such as face detection and tracking, but goes on to something different. Their objective is to be able to identify the type of social interaction happening by knowing which kind of groups are partaking in it, as explained in the relative sub-section. The idea behind this is that social interactions always have recurrent structures. A lecture usually has rows of students sitting while listening to a professor who is standing in from of them. While this approach could offer simpler classification methods, it could return false results. The assumption that doesn't have to be violated is that determined social interactions either never change their structure or that they have different ways they can be composed. This last part also implies that a particular structures must uniquely identify a type of social interaction.

In [6] the authors approach is to simplify the process of identification and tracking as much as possible without loss of generality. Before advancing to the clustering procedure assisted by structured SVM, as previously discussed, they estimate the relative position of individuals for each frame. Only two variables from the 3D scene are used for the reconstruction of the model by assuming that faces are on the same plane. This plane is divided into cells with which the depth information of the subjects is estimated.

Another key point of the above mentioned paper is the employment of a novel head pose estimation procedure. To remove as much noise as possible for obtaining more precise results, they perform various methods to achieve robustness against scale and lighting settings. These include contrast normalization, background subtractions and resizing.

Results for their methods are reported in Figure 8. As we can observe and as the authors report, some training scenarios perform better than others. When the algorithm trains on the laboratory setting, it achieves a good generalization by keeping a low standard deviation on the error. However is also noticeable how the training on the coffee settings and the party setting tend to push the model towards overfitting.

The type of footage used in the previous example is high quality video. With high quality we mean a resolution which is usually 1920x1080 or 1280x720 with a 15 or 30 frame per seconds capturing rate. The employment of this type of data resulted in an high storage and computational complexity. In [1] the authors experimented on the adoption of low frame rate video. Obtaining pictures every 30 seconds results in a drastic decrease of data. Their approach to cope with less data is with the employment of an LSTM recurrent neural network. As hidden units behave like memory cells, their algorithm is able to remember the previous social interactions and identify when such events start and finish. To also help improve the social interaction recognition, the authors propose four available settings:

SID1: This utilize information on the distance of the individuals and their head rotation.

SID2: This setting is built upon the previous one by adding also the faces pitch and roll. This way we are estimating the complete

	Training: Laboratory			Training: Coffee			Training: Party			Training: Outdoor			Training: All		
Test scenario	**Error**	**Precision**	**Recall**	**Error**	**Precision**	**Recall**	**Error**	**Precision**	**Recall**	**Error**	**Precision**	**Recall**	**Error**	**Precision**	**Recall**
Coffee	11.55	82.99	97.17	11.69	79.17	100.00	18.90	69.44	100.00	6.75	92.62	94.06	6.50	88.80	99.46
Party	9.33	100.00	83.63	0.00	100.00	100.00	0.00	100.00	100.00	10.92	100.00	80.34	3.15	96.27	98.05
Laboratory	11.91	91.68	85.79	14.75	74.67	99.43	14.43	74.81	100.00	27.75	72.60	72.81	19.97	74.32	88.05
Outdoor	10.97	87.39	95.09	11.31	81.25	100.00	11.31	81.25	100.00	29.76	100.00	58.93	15.83	83.93	89.17

Fig. 8. Results from training and testing the method described in [6] on the ECO-Group dataset.

head orientation and therefore achieving higher precision in social interaction detection.

SID3: This is also built upon the first setting but instead of adding pitch and roll, an estimate of emotion is calculated by the facial expression and used classification.

SID4: Here the authors combine all the information previously mentioned obtaining a more complex model as well as a more precise one.

As one could expect the SID4 setting yielded the highest result for both social interaction detection and classification. This showed how less information can be captured and employed if we are willing to accept a higher model complexity as a trade off.

4 Conclusion

As we have seen, scientific literature presents various approaches to identifying and classifying social interactions. The common ground they all share is comprised of the face detection and tracking techniques employed. However, the differences between papers lies into the implementation details. This is, for example, noticeable in the techniques employed for classification. From clustering to Support-Vector Machines and to LSTM Recurrent Neural Networks, they were still able to obtain significantly good results.

The method that however impressed us the most is described in [4]. Here the footage available possesses only one frame for 30 seconds. The model they proposed, however, is obviously more complex due to estimating the complete head position and orientation of the subject. However, the accuracy with which they can identify and classify social interaction range from 85% to 95%, which is a remarkable result with such few images.

As a future development it would be beneficial for the field to identify and recognise subtle body gestures as described in [9]. They are often carried out unconsciously and could offer a greater insight into social interactions.

Acknowledgements

The authors wish to thank Estefania Talavera Martinez and Alina Matei for reviewing this paper.

References

[1] M. Aghaei, M. Dimiccoli, C. C. Ferrer, and P. Radeva. Social style characterization from egocentric photo-streams. *arXiv preprint arXiv:1709.05775*, 2017.

[2] M. Aghaei, M. Dimiccoli, C. C. Ferrer, and P. Radeva. Towards social pattern characterization in egocentric photo-streams. *Computer Vision and Image Understanding*, 171:104–117, 2018.

[3] M. Aghaei, M. Dimiccoli, and P. Radeva. Multi-face tracking by extended bag-of-tracklets in egocentric photo-streams. *Computer Vision and Image Understanding*, 149:146–156, 2016.

[4] M. Aghaei, M. Dimiccoli, and P. Radeva. With whom do i interact? detecting social interactions in egocentric photo-streams. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2959–2964. IEEE, 2016.

[5] T. Albrecht, M. Lüthi, and T. Vetter. *Deformable Models*, pages 210–215. Springer US, Boston, MA, 2009.

[6] S. Alletto, G. Serra, S. Calderara, F. Solera, and R. Cucchiara. From ego to nos-vision: Detecting social relationships in first-person views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 580–585, 2014.

[7] H. Buxton. Learning and understanding dynamic scene activity: a review. *Image and vision computing*, 21(1):125–136, 2003.

[8] W. Choi, Y.-W. Chao, C. Pantofaru, and S. Savarese. Discovering groups of people in images. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 417–433, Cham, 2014. Springer International Publishing.

[9] M. Cristani, R. Raghavendra, A. Del Bue, and V. Murino. Human behavior analysis in video surveillance: A social signal processing perspective. *Neurocomputing*, 100:86–97, 2013.

[10] A. Fathi, J. K. Hodgins, and J. M. Rehg. Social interactions: A first-person perspective. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1226–1233. IEEE, 2012.

[11] D. Gatica-Perez. Automatic nonverbal analysis of social interaction in small groups: A review. *Image and vision computing*, 27(12):1775–1787, 2009.

[12] E. Goffman et al. *The presentation of self in everyday life*. Harmondsworth London, 1978.

[13] A. Kendon. The f-formation system: The spatial organization of social encounters. *Man-Environment Systems*, 6(01):1976, 1976.

[14] A. Kendon. *Conducting interaction: Patterns of behavior in focused encounters*, volume 7. CUP Archive, 1990.

[15] F. Setti, C. Russell, C. Bassetti, and M. Cristani. F-formation detection: Individuating free-standing conversational groups in images. *PloS one*, 10(5):e0123783, 2015.

[16] D. Umberson and J. Karas Montez. Social relationships and health: A flashpoint for health policy. *Journal of health and social behavior*, 51(1_suppl):S54–S66, 2010.

[17] M. Vilain, J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman. A model-theoretic coreference scoring scheme. pages 45–52, 01 1995.

[18] C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on image processing*, 7(3):359–369, 1998.

[19] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13–es, 2006.

[20] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2879–2886. IEEE, 2012.

Comparison between the Dropout and DropConnect regularization schemes

Ludger Visser, Ariadna Albors Zumel

Abstract— Deep learning neural networks are very powerful, but likely to overfit on a training dataset. In this paper, we look at two state-of-the-art regularization methods for dealing with the overfitting problem of a deep neural network. First, we look at Dropout, a technique where we drop units with all their connections from the network randomly by a chosen probability. This reduces the co-adapting in units, which results in less overfitting. Second, we consider DropConnect a generalization of Dropout. In DropConnect, we randomly drop only weights by a probability. We evaluate both regularization schemes on a range of datasets in shallow architectures. Finally, we compare the accuracy performance of the two schemes on different datasets and conclude that Dropout performs slightly better than DropConnect despite the latter being more stable through different rates.

Index Terms—Neural Networks, Deep Learning, Regularization, Dropout, DropConnect.

1 Introduction

Deep neural networks are popular machines because they can be trained to learn complex relationships between input and output. In order to be capable of this, neural networks have many parameters, sometimes as many as millions. With that many parameters, deep neural networks are known to try and interpret patterns in the noise of its training data. To reduce the aforementioned overfitting, several methods have been developed. This includes methods that stop training as soon as the performance on a test set deteriorates. For example by weight penalties, L1 and L2 regularization or weight sharing [10]. Without early stopping the training, a popular alternative method to prevent overfitting is by use of an ensemble of models. With an ensemble of model, each model is trained on different data and for testing one takes the average result of the models' output for data input. This works decently for small neural networks [16], but it is not the perfect solution for all neural networks. Often one neural network model requires already a long training time by itself. Thus multiple models for an ensemble require more training time than desired. Besides the long training time, there is also the extra effort to maintain many different models needed for an ensemble.

Dropout [13] is a regularization method where during training some units are dropped from the neural network. In each training iteration, different units are dropped to get a different neural network at each training iteration. This prevents hidden units in neural network layers from co-adapting too much, which would result in overfitting. DropConnect [15] is another regularization method. DropConnect drops weights of units during the training phase while Dropout drops whole units with all their weights. Both Dropout and DropConnect are similar to an ensemble having different neural networks during the training phase, but not the additional inconvenience and computational effort of maintaining different models.

Both Dropout and DropConnect have shown successful results[6] with an Extreme learning machine, which is a neural network consisting of a single hidden layer where the input-to-hidden weights are randomly initialized and only the output weights are determined by backpropagation. DropConnect has also shown promising results with a learning vector quantization algorithm implementation [11]. Both papers [6, 11] show promising results with an increasing classification performance compared to using neither generalization scheme.

- *Ludger Visser is an MSc Computing Science student of the University of Groningen, E-mail: l.e.f.visser@student.rug.nl.*
- *Ariadna Albors Zumel is an MSc Computing Science student of the University of Groningen, E-mail: a.albors.zumel@student.rug.nl.*

In this paper, we perform four experiments with Dropout and DropConnect to further investigate their performance on classification tasks using neural networks. We used different drop rates and different datasets. In our experiments, we found that it is not necessarily always beneficial to use Dropout or DropConnect on a neural network since only a few specific rates improve its performance. We also observed that DropConnect is more stable than Dropout, although Dropout usually achieves higher accuracies.

The following part of the paper is organized as follows: in Section 2 we give more background information about how Dropout and DropConnect work. In Section 3 we explain our experiment setups in detail, including a description of the datasets used. In Section 4 we show the results of the experiments performed on each dataset. Lastly, in Sections 5 and 6 we discuss and draw conclusions from our results.

2 Background

In the following subsections we introduce both Dropout and Dropconnect separately. We look at how they are applied to a neural network schematically and the mathematical background of both methods.

2.1 Dropout

In Dropout, we drop some units in the neural network with all their output weights [13]. This can be seen in Figure 1. The choice of which units to drop is random and each unit is retained by a probability p and dropped by probability $1-p$. Dropout can be described in mathematical terms. For a single hidden unit with input vector $v = [x_1, x_2, ..., x_n]$ the output of the unit can be described as,

$$H = f(\sum_i W_i x_i) \tag{1}$$

In Eq. (1), the unit's local threshold is left out for simplicity. Here f is the activation function and W is the weight vector $[w_1, w_2, ..., w_n]$. Applying Dropout to this single hidden unit, would result in:

$$H = p \cdot f(Wx) \tag{2}$$

m is a Bernoulli variable with probability p to be 1 and $1-p$ to be 0. For $m = 1$, we retain the unit and Eq. (2) is the same as Eq. (1). For $m = 0$, the output is 0 and we drop the unit. Now instead of a single hidden unit, we have a layer consisting of d hidden units, where $d \in \mathbf{N}$. In this case, the weights W then become a matrix of size $n \times d$. Without implementing Dropout, the output vector of the units can be described by r with:

$$r = f(Wx) \tag{3}$$

with which Dropout becomes:

$$r = M \star f(Wx) \tag{4}$$

where $\star$ is the element-wise product. M is a vector of d independent Bernoulli random variables each of which has a probability p of being value 1 and otherwise 0. M is called the mask and it determines which units are dropped. A new mask is drawn for every example during training.

The Dropout probability is a hyperparameter and should be a value between 0 and 1. A value of 0.5 seems close to the optimal for a wide range of tasks and networks [13]. Before every training iteration, we randomly drop units starting from the full original neural network again. Following this approach, there is very little chance to have exactly the same network of active units in different training iterations. This prevents the units from co-adapting too much whereby the neural network tries to read patterns in the noise of the training data.

During testing, we do not actually drop units. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in Figure 2. This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time [13]. The intuitive idea behind Dropout is that it takes the averages of different models by dropping units randomly. The same way an ensemble uses multiple different models, but without having to train all the models separately. By dropping units we thin the neural network, therefor it makes sense to start with a bigger network than one would without Dropout. To do so we divide the size of the network without Dropout by the retain probability of the hidden units. That should be the size of our neural network where we will apply Dropout. Following this approach, the expected network size after dropping units will be the same as the other neural network without applying Dropout.

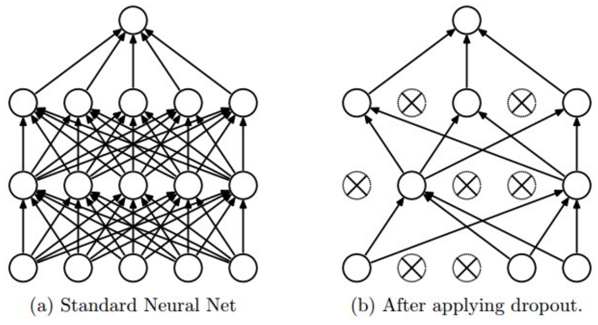


Fig. 1. Dropout Neural Net Model. **Left**: A standard neural network with 2 hidden layers. **Right**: An example of a thinned network produced by applying Dropout to the network on the left. Crossed units have been dropped. Figure taken from [13].

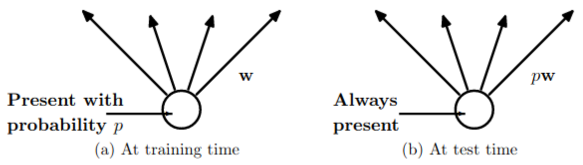


Fig. 2. Neural network hidden unit **Left**: At train time a unit is dropped with a probability $1-p$. **Right**: At test time all output weights of a unit are multiplied by probability p. Figure taken from [13].

2.2 DropConnect

The second method we will discuss is DropConnect[15]. Just as in Dropout, DropConnect has a different model for each training iteration. The difference is that in Dropout we randomly drop all the weights of some units, whereas in DropConnect we randomly drop single weights. In a neural network, there are always more weights than hidden units, so there are more ways to thin the model with DropConnect. DropConnect can also be described in mathematical terms. From Eq. (3), where we described a layer without Dropout or DropConnect, the equation of a layer with DropConnect follows as:

$$r = f((M \star W)x) \tag{5}$$

where DropConnect mask M is a matrix of $r \times d$ independent Bernoulli random variables, each of which has a probability p of being value 1 and otherwise 0. The mask M is also drawn again for every example during training.

In Figure 4 Dropout is applied to the neural network with 1 hidden layer from Figure 3. In Figure 5 DropConnect is applied to the same neural network for comparison. DropConnect is a generalization of Dropout because with DropConnect one can drop all the weights from a hidden unit as is the case with Dropout. This can be seen in Figure 5 on the right, as the second hidden unit is effectively dropped. As with Dropout, DropConnect prevents the units from adapting too much and read patterns in the noise of the training data. Figure 6 shows another perspective of Dropconnect in a neural network.

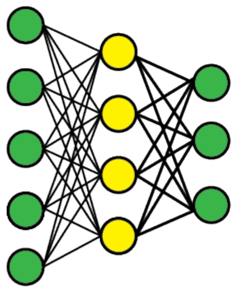


Fig. 3. A neural network with 1 hidden layer. Green circles are the input and output units and yellow circles are the hidden units. No Dropout or DropConnect applied. Figure taken from [1].

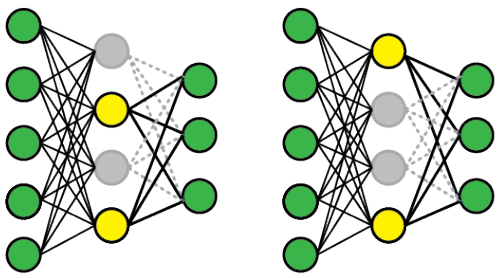


Fig. 4. A neural network with 1 hidden layer. Green circles are the input and output units, yellow circles are the hidden units and grey circles are dropped units. Left and right are two different iterations of training, both after applying Dropout. Figure taken from [1].

3 Experiments

In order to compare the performance of Dropout to DropConnect, we built four neural networks to perform classification tasks on two different datasets. The neural networks were built using a high-level neural network API for Python called Keras [7, 8].

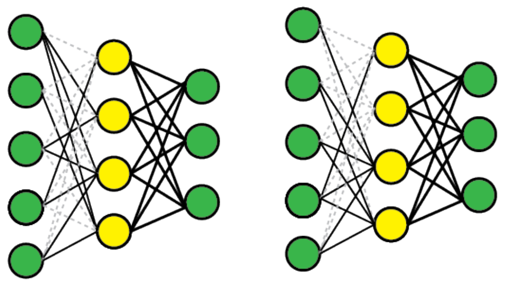


Fig. 5. A neural network with 1 hidden layer. Green circles are the input and output units. The grey faded edges are dropped weights and the clear grey edges are retained weights. Left and right are two different iterations of training after applying DropConnect. Figure taken from [1].

This section is divided into two parts. Firstly, we introduce the two datasets used. Afterwards, we describe the neural network architectures used for each of the four experiments.

3.1 Datasets

3.1.1 CIFAR-10 dataset

The dataset CIFAR-10 [4] is a balanced dataset, which consists of 60,000 color images of 32x32 pixels of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). From these images, 50,000 were training images and the 10,000 remaining were testing images. In order to make this classification task differ more from our second experiment, we choose only two classes (horse, dog), resulting in a binary classification task. Therefore, we used a total of 10,000 images for training and 2000 for testing.

Before using this dataset, we preprocessed the data by normalizing the pixel values to the [0,1] range and also performed one-hot-encoding. One-hot-encoding transforms the training labels into binary n-dimensional vectors (where n is the number of classes) with zeros everywhere except at the position of the corresponding class, which has a one [2]. It is the most common target encoding strategy [12].

We performed two experiments with this dataset; one with only dense layers and another with also convolutional layers. The architectures of the corresponding neural networks are explained in the subsections 3.2.1 and 3.2.2 respectively.

For both experiments, we trained and evaluated the performance of our neural network on ten different Dropout/DropConnect rates ranging from 0 to 0.9 with steps of 0.1. The evaluation for each rate was repeated five times in order to take into consideration the randomness in the Dropout/DropConnect schemes as well as in the optimizers.

3.1.2 MNIST

The second dataset used was the MNIST Database of Handwritten Digits. It contains a total of 60,000 training and 10,000 testing greyscale images of 28x28 pixels [3]. The images consist of handwritten digits with values between 0 and 9. Again, before training the neural network, we applied the same preprocessing as for the CIFAR-10 dataset.

Similarly, as with the CIFAR-10 dataset, we performed two experiments; one with convolutional layers and one with only dense layers (an explanation of the neural network architectures can be found in the subsections 3.2.3 and 3.2.4).

Once more, we trained and then tested the performance of the neural network of both experiments on ten different Dropout/DropConnect rates ranging from 0 to 0.9 with steps of 0.1 and the evaluations were performed five times for each rate.

3.2 Network architectures

3.2.1 CIFAR-10 with dense layers

The first experiment with this dataset contained four dense layers and a Dropout/DropConnect layer between the first and the second dense layers. The model was then compiled using the Stochastic gradient descent (SGD) optimizer [14] and Categorical Cross-Entropy loss (also known as Softmax loss) [5].

3.2.2 CIFAR-10 with CNN layers

The neural network used on the second experiment with this dataset consisted of two convolutional layers and one dense (fully-connected) layer as the output layer. Between them, we also applied a layer of batch normalization, which normalizes the activation of the previous layer [7], and a layer of max pooling, which is the most common type of pooling layer and is used to reduce overfitting through spacial reduction. Then we had the Dropout/DropConnect layer, depending on the experiment, situated after the first max pooling layer. The compilation was done using the same loss function and optimizer as in the previous experiment.

3.2.3 MNIST with dense layers

The neural network for the first experiment with this dataset was composed of three dense layers and a Dropout/DropConnect layer between the second and the third aforementioned dense layers. The compilation of the model was done using the Adam optimizer [9] and Categorical Cross-Entropy, as mentioned earlier, for the loss function.

3.2.4 MNIST with CNN layers

We performed a second experiment, where we slightly modified the architecture of the neural network used in the previous experiment and added a convolutional layer as the first hidden layer. In order to make the experiments more comparable, we also kept the same optimizer and loss function.

4 Results

The results from the first experiment can be found in Figure 7, where we show the average accuracy (for both the training and the testing data) over the five measurements for each rate. Firstly, we can observe that there is indeed overfitting since the training accuracies are higher than the testing accuracies (the testing accuracies will be referred to as “accuracies” in the remaining of the paper).

It can also be observed that in this experiment, DropConnect outperforms Dropout (we always discuss the performance with respect to the testing data). In addition to Dropout drastically decreasing on accuracy for high rates, it also does not achieve a significant improvement on accuracy compared to the base accuracy (the accuracy at rate 0, which does not use any regularization schema). Contrarily, DropConnect shows stable results (despite having a slightly higher standard deviation) and achieves maximum performance at the rate of 0.8, which meant an almost 0.02 increase in performance compared to the base accuracy.

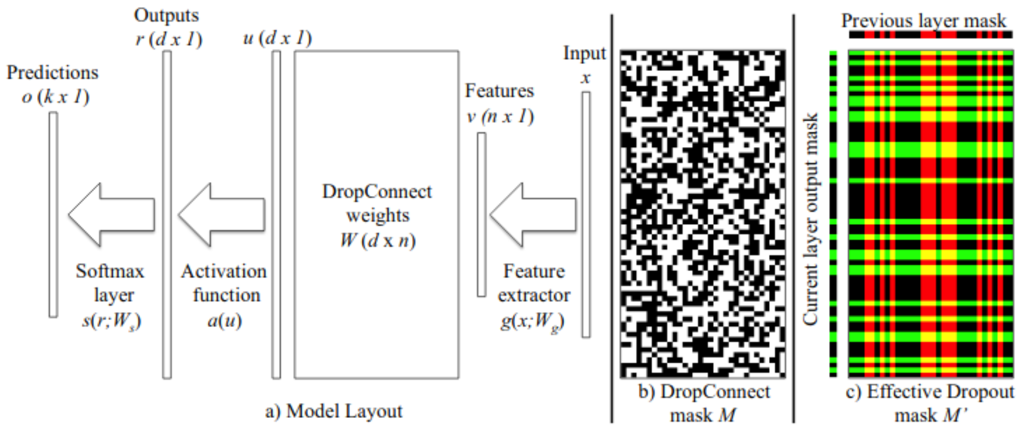


Fig. 6. as Li Wan et al. writes in [15] ”The masked weights are multiplied with this feature vector to produce u which is the input to an activation function a and a softmax layer s. For comparison, (c) shows an effective weight mask for elements that Dropout uses when applied to the previous layer's output (red columns) and this layer's output (green rows). Note the lack of structure in (b) compared to (c).” This is a neural network structure with DropConnect implemented. Figure taken from [15]

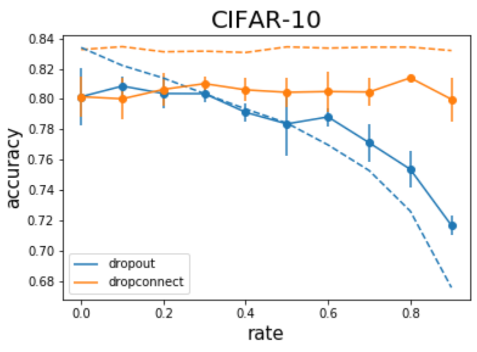


Fig. 7. Accuracies for a binary classifier using data from the CIFAR-10 dataset. The solid lines are testing accuracies and the dashed lines are training accuracies. The standard deviation for the training data are of the order of 0.001 for both regularization schemes.

In Figure 8, there are the results obtained from the experiment performed on the same dataset, the CIFAR-10 dataset, but this time using a convolutional neural network.

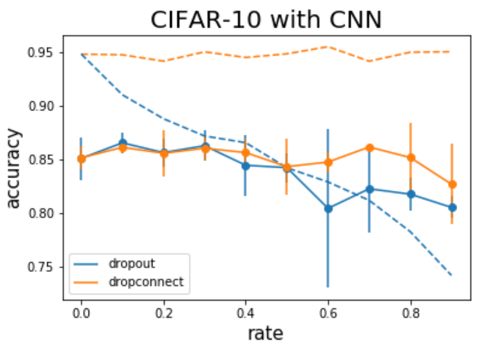


Fig. 8. Accuracies for a binary classifier using data from the CIFAR-10 dataset on a CNN. The solid lines are testing accuracies and the dashed lines are training accuracies. The standard deviation for the training data are of the order of 0.001 for both regularization schemes.

Again, we can see that that the neural network is overfitting, meaning that using a regularization schema is needed. It is also observable that most measurements of the accuracies using DropConnect have a small standard deviation, meaning that this schema is quite stable. It also showed an increase in performance for most regularization rates, although the improvement in accuracy was only of 0.01 at the maximum point. On the other hand, the results obtained using Dropout show more significant standard deviations, indicating that this regularization schema is less stable than DropConnect.

We can also see that the rates of 0.1 and 0.3 slightly improved the performance of the neural network, while the other rates decreased its performance. The results also show that Dropout is more affected by significant rates compared to DropConnect, which has improved performance for rates as high as 0.7.

The results obtained from the MNIST dataset, without convolutional layers, can be found in Figure 9. Despite the small difference between training and testing accuracies, it can still be observed that the neural network is overfitting. Moreover, unlike in the previous experiment, Dropout was the regularization schema that performed better in this case, with a maximum increase of performance achieved at a rate of 0.4. On the other hand, DropConnect shows a slightly higher standard deviation than Dropout, although the latest has more difference between the performance for each rate. Again, it can also be observed that Dropout performs worse than DropConnect for higher rates.

It should also be noted that for the MNIST dataset, the increased accuracy achieved with either regularization scheme was small, with a maximal increase in accuracy of 0.001 achieved with the aforementioned 0.4 rate on Dropout. This was probably due to the neural network not overfitting much, which meant that the regularization schemes could not have a significant contribution to improving the performance of the neural network.

Lastly, from the results obtained for the experiment on the MNIST dataset using a convolutional neural network (see Figure 10), it can be observed that DropConnect, despite showing a slightly higher standard deviation than Dropout, gives more stable results and shows better performance for high rates. On the other hand, Dropout is the regularization schema that achieves the most significant improvement in accuracy at the rate of 0.3, which is close to the optimal rate for Dropout obtained on the previous experiment (MNIST dataset without convolutional layers on the neural network). Again, we observed the neural network was overfitting, which made the use of a regularization schema relevant.

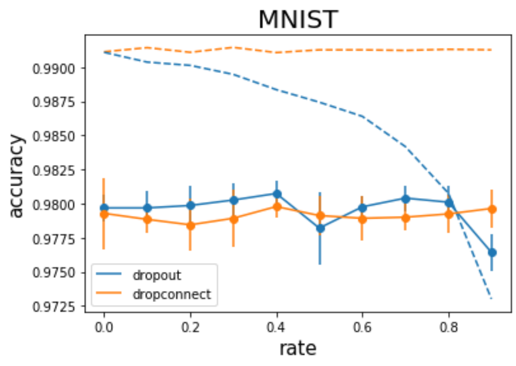


Fig. 9. Accuracies for a multi-class classifier using data from the MNIST dataset. The solid lines are testing accuracies and the dashed lines are training accuracies. The standard deviation for the training data are of the order of 0.0001 for both regularization schemes.

Another notable result was the drastic decrease in accuracy present in the experiment with Dropout at a rate of 0.9. This behavior was also observed in our first and third experiments.

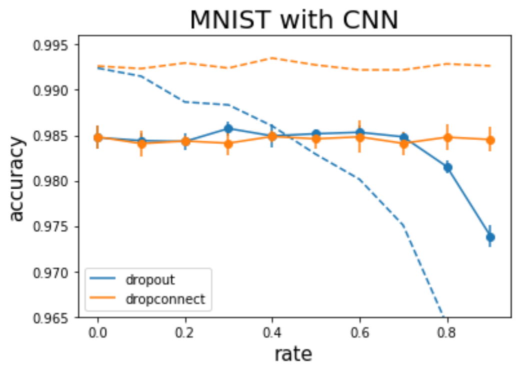


Fig. 10. Testing accuracies for a multi-class classifier using data from the MNIST dataset on a CNN. The solid lines are testing accuracies and the dashed lines are training accuracies. The y-axis was chopped in order to see the difference between the testing accuracies of the two schemes better. The standard deviation for the training data are of the order of 0.0001 for Dropout and 0.001 for DropConnect.

The results for all four experiments are summarized in Tables 1 and 2. The former contains the results for all the Dropout experiments while the latter contains the same information but for the DropConnect experiments.

Experiment	base acc.	max. acc.	avg. acc.	avg. std
CIFAR-10	0.802	0.809	0.784	0.009
CIFAR-10 w. CNN	0.851	0.865	0.838	0.023
MNIST	0.979	0.981	0.979	0.001
MNIST w. CNN	0.985	0.986	0.984	0.0009

Table 1. Summary of accuracies and standard deviations obtained for each experiment using **Dropout** as regularization schema.

Experiment	base acc.	max. acc.	avg. acc.	avg. std
CIFAR-10	0.802	0.814	0.805	0.010
CIFAR-10 w. CNN	0.851	0.861	0.852	0.016
MNIST	0.979	0.979	0.979	0.002
MNIST w. CNN	0.984	0.985	0.985	0.0012

Table 2. Summary of accuracies and standard deviations obtained for each experiment using **DropConnect** as regularization schema.

5 Discussion

The paper aimed to compare two regularization schemes, Dropout and DropConnect, which are used in order to deal with overfitting in neural networks. To perform such a comparison, we conducted four experiments, which we showed that were overfitting since the training accuracies were higher than the testing accuracies. From the results obtained (see Tables 1 and 2), it can be concluded that DropConnect is a more stable regularization schema since it showed more consistent results over the different rates (despite having a higher standard deviation in three out of the four experiments) and performed well also on higher rates. On the other hand, unlike in [15], the maximum improvement in accuracy was achieved using Dropout as regularization schema in all experiments except the first one, which was unexpected. This implies that if the optimal rate for Dropout is known, this should be the regularization method used in order to achieve the most favorable results. Despite this, we also observed that the minimum accuracies were also obtained using Dropout with a rate of 0.9 in all the experiments performed.

Our experiments did not show any clear preference between Dropout or DropConnect, depending on whether the neural network had convolutional layers or not. Still, this does not discard a possible correlation between the neural network architecture and the optimal regularization schema to be used.

It should be noted that the number of experiments performed was limited, which means that our results and corresponding conclusions should only be taken as starting points for future research, in particular with respect to the standard deviations. Still, it provided us with a first intuition on the core differences between Dropout and DropConnect and indicated that further research should be done to investigate the idea that DropConnect might be a more stable regularization schema than Dropout.

Therefore, to achieve a deeper understanding of the advantages of using either Dropout or DropConnect as regularization schemes, more experiments should be performed. These experiments should explore new datasets as well as variations of the presented neural network architectures for CIFAR-10 and MNIST, such as using different optimizers, activation functions, amount of layers or amount of neurons per layer.

6 Conclusion

To summarize, in this paper we have performed four experiments on two different datasets in order to compare the Dropout and DropConnect regularization schemes. From our results, we conclude that DropConnect is more stable than Dropout, although Dropout usually performs better for specific rates. These results provide a starting point for future research, where more experiments should be performed to get a deeper understanding of the advantages of each regularization schema.

Acknowledgements

The authors wish to thank Michael Biehl for his availability and guidance.

References

[1] Machine learning questions: How to prevent overfitting. Retrieved from `https://ztlevi.github.io/Gitbook_Machine_`

Learning_Questions/docs/General/How_to_prevent_overfitting.html, last accessed on 27/02/2020.

[2] H. Jaeger. Machine Learning. Lecture Notes, 2020. Master Program in Artificial Intelligence Rijksuniversiteit Groningen, Bernoulli Institute.

[3] Y. LeCun, C. Cortes, and C. Burges. The mnist database of handwritten digits. Retrieved from http://yann.lecun.com/exdb/mnist/, last accessed on 23/02/2020.

[4] CIFAR-10. The cifar-10 dataset. Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html, last accessed on 23/02/2020.

[5] I. Goodfellow et al. *Deep Learning*. MIT Press. Retrieved from http://www.deeplearningbook.org, last accessed on 11/03/2020.

[6] A. Iosifidis, A. Tefas, and I. Pitas. Dropelm: Fast neural network regularization with dropout and dropconnect. *Neurocomputing*, 162:57–66, 2015.

[7] Keras Documentation. Keras: The python deep learning library. Retrieved from https://keras.io/, last accessed on 23/02/2020.

[8] Keras Team. Keras. Retrieved from https://github.com/keras-team/keras, last accessed 22/02/2020.

[9] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec. 2014.

[10] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[11] J. Ravichandran, S. Saralajew, and T. Villmann. Dropconnect for evaluation of classification stability in learning vector quantization. *ESANN 2019 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.*, 2019.

[12] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera. Beyond One-hot Encoding: lower dimensional target embedding. *arXiv e-prints*, page arXiv:1806.10805, June 2018.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[14] T. Hastie et al. *The elements of statistical learning: Data mining, inference, and prediction*. Springer, 2001.

[15] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[16] H. Y. Xiong, Y. Barash, and B. J. Frey. Bayesian prediction of tissue-regulated splicing using rna sequence and cellular context. *Bioinformatics*, 27(18):2554–2562, 2011.

An Analysis of Neural Network Pruning in Relation to the Lottery Ticket Hypothesis

M.J. Havinga R.S. Sawhney

Abstract—In this paper, we analyze the novel “lottery ticket” approach for pruning neural networks and compare it to traditional techniques on dense feed-forward neural networks by the means of identifying “winning tickets”. We explore the precise conditions required to identify and make use of these winning tickets and explore perspectives on the hypothesis from other work. We conclude that there is demonstrable evidence in support of the hypothesis, but revealing winning tickets in any neural network remains a hard task. We then discuss the continued research in this topic and suggest some research directions of our own.

Index Terms—neural networks, lottery ticket hypothesis, network pruning, network compression, architecture search

1 Introduction

Neural network pruning refers to the process of removing weights from (dense) neural networks with the goal of improving efficiency in terms of energy and memory consumption, as well as to simplify the model [5, 21]. An early example of neural network pruning is named *Optimal Brain Damage* and was proposed in 1990 by LeCun et al. [14]. This method takes a reasonably sized network and selectively deletes half of the weights based on a special metric, *saliency*, in order to come up with a sub-network which could perform just as well if not better than the main network. Many more methods for pruning have since been proposed as illustrated in Sec. 2.1. The common goal of these methods is the reduction of the number of weights in the model, while sustaining an approximately equal model accuracy. As we will see later, pruning can also improve the accuracy. Figure 1 shows an illustration of pruning in general.

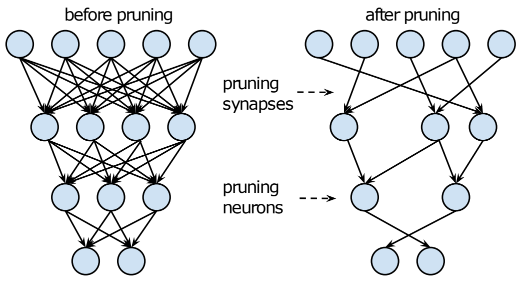


Fig. 1: Illustration of unstructured pruning. Weights are removed by the pruning algorithm based on some metric resulting in a sparser network. Taken from [5], page 3.

In general, pruning is performed *while* training the network. The resulting model is already trained at that point and reduced in the number of parameters. An open question remains whether it is possible to train a network from scratch, once it has already been pruned to a sparse architecture [15, 5]. So far, It has been shown that training a pruned neural network again from a new random initialization is a hard task. When re-initializing such a sparse network with random weights it generally achieves a lower accuracy than the original trained network. For example, Li et al. noticed this in [15] and claimed a “difficulty of training a network with a small capacity”. As can be seen in Table 1 in [15], pruned models that have been retrained from scratch perform systematically worse than their pruned and unpruned original model.

M.J. Havinga and R.S. Sawhney are students at the University of Groningen, Computing Science, Master programme.

If such a sparse network is hard to train, why has it proven successful to train an over-parameterized network and prune it until a sparse network remains? Han et al. explain this in [5] through the existence of “fragile co-adapted features” in the network as described in [24]. Essentially, it is implied that the configurations of the weights co-depend and can not be trivially re-initialized and retrained. Following up on these findings, Frankle and Carbin proposed the *lottery ticket hypothesis* and an accompanied conjecture, stated as follows.

Hypothesis. *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that–when trained in isolation–it can match the test accuracy of the original network after training for at most the same number of iterations.*
Conjecture. *Stochastic gradient descent seeks out and trains a subset of well-initialized weights. Dense, randomly-initialized networks are easier to train than the sparse networks that result from pruning because there are more possible subnetworks from which training might recover a winning ticket.*

Frankle and Carbin (2019) in [3]

From this point forward, we will refer to both the hypothesis and the accompanied conjecture as “the lottery ticket hypothesis”. In this paper, we explore this hypothesis. We task ourselves with investigating the following questions.

1. What is the precise meaning and implication of the lottery ticket hypothesis?
2. Can we find experimental evidence of the lottery ticket hypothesis and if so under which conditions?
3. How can we explain the evidence supporting the lottery ticket hypothesis and with which mechanisms?

To be able to adequately answer these questions, we will provide some background information in Sec. 2. After this, we will discuss the methods and experimental results found by Frankle and Carbin [3], Liu et al. [16] and ourselves in Sec. 3 and 4. In our discussion in Sec. 5, we implicitly answer our research questions. Finally in Sec. 6, we suggest some directions for future research in this topic.

2 Background

As briefly explained previously, pruning refers to process of removing weights from a (dense) neural network. Christopher Bishop [2] describes pruning as a method for finding optimal network architectures, as a counterpart to network growing which adds weights rather than remove them. On the other hand, Simon Haykin [6] describes pruning as a regularization method for neural networks, i.e. to improve the model’s generalization abilities. Sec. 2.2 further analyzes pruning in this context. We will see that it is reasonable to regard pruning as both a method of architecture search and regularization. There are many different pruning methods using several completely different approaches. This large and diverse family of methods is illustrated in the next section.

2.1 Pruning methods

Pruning methods can be classified as either pre-defined by the user or automatically derived by the network. Another division of methods can be made based on whether the pruning algorithm takes the network architecture into account. Methods that do this are called structured methods, while methods that only consider individual weights are called unstructured methods.

Unstructured pruning

Unstructured pruning methods find individual weights that are least relevant and can be pruned. This method was used by Frankle and Carbin to perform their experiments for the lottery ticket hypothesis [3]. There is an important distinction between one-shot pruning and iterative pruning.

One-shot pruning. This is the basic type of unstructured pruning method which prunes the network as as function of the percentage of the weights [3]. It involves the following steps.

1. Randomly initialize a neural network.
2. Train the network.
3. Set p% of weights from each layer (based on some metric) to 0.

Iterative Pruning. This is similar to the one-shot pruning method. The major difference is that the weights are trained and pruned over n rounds. In each round, $p^{\frac{1}{n}}\%$ of the weights are removed. This is the main method used in [3].

To find winning tickets, Frankle and Carbin propose to reset the weights of the pruned network to their original random initialization [3]. If the found model is indeed a winning thicket, it can then be retrained to gain an equivalent accuracy as the unpruned model.

Pre-defined structured pruning

The methods which employ structured pruning involve removing the weights in the channels while keeping the target architecture in mind before performing the pruning. The way in which pruning is performed is set by the user and the manner is dictated by the pruning algorithm itself. These methods were employed by Liu et al. to counter the unstructured pruning methods performed by Frankle and Carbin [16]. This technique of pruning is briefly explained in Figure 2. The different pre-defined structured pruning methods as used in Liu et al.'s work are given below.

L1-norm based Filter Pruning. One of the earliest pruning methods on channel pruning for convolutional neural networks. In this method, the L1-norm for the filters is considered in each layer. A pre-defined percentage of filters with smaller L1-norm is eliminated. [15]

ThiNet. This method is greedy as compared to the L1-norm. It prunes the channel that has the least effect on next layer's activation values [17].

Regression Based Feature Reconstruction. The pruning is performed by reducing the error during reconstruction of feature map of the next layer. The optimization problem in this method is solved with the help of LASSO Regression. [8]

Automatic Structured Pruning

Unlike the pre-defined structured pruning methods, here the target architectures for the sub-network are derived automatically. Liu et. al. argue that training these models from scratch can lead to comparable

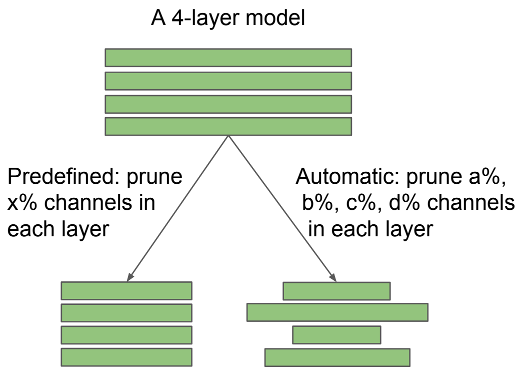

A 4-layer model
Predefined: prune x% channels in each layer
Automatic: prune a%, b%, c%, d% channels in each layer

Fig. 2: A visualization for the pre-defined and automatic structured pruning method. Figure 2 taken from [16].

or even better performance, concluding that the architecture derived in these methods is much more important than the weights themselves. The different automatic pruning methods are given below.A brief methodology for this technique of pruning is explained in Figure 2.

Network Slimming. As described in [16], "imposes L1-sparsity on channel-wise scaling factors from Batch Normalization layers during training, and prunes channels with lower scaling factors afterward."

Sparse Structure Selection. This method is a generalization of the network slimming method as it also makes use of the sparsified scaling factors to be used for pruning. It has been shown to be applicable on the residual blocks in ResNet if required. [10]

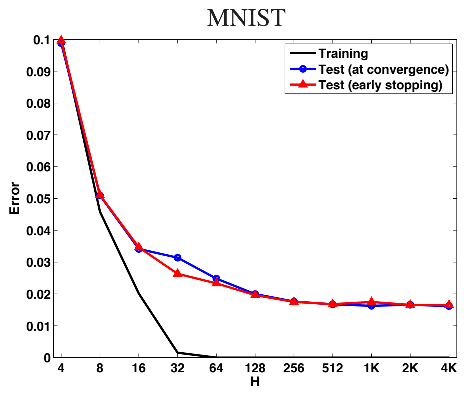


Fig. 3: Relationship between the training/test error and the size of a 2-layer neural network trained by momentum stochastic gradient descent. Here, H represents the network size in terms of the number of hidden units. Figure 1 taken from [20].

2.2 Pruning in relation to the generalization error

Neural network pruning has been observed to improve the generalization capabilities of the model. However, it is still debated how pruning achieves this [1]. Historically, the network's size in terms of the number of free parameters has been used as a metric of the complexity of a network [14]. Since increased complexity implies an increased risk of overfitting, it has been conjectured that the shrinking of a neural network reduces this risk and improves the generalization capabilities of the network. This theory however, has not been generally proven. In fact, contradictory evidence has been found for specific network architectures which implies that larger networks can generalize better than smaller networks [20]. As can be seen in Figure 3, a larger network always improves both the training and test error for this specific experiment. It has been suggested that it is the pruning method itself from which the improved generalization arises. This theory, as postulated by Bartoldson et al. [1], states that the “destabilizing effects” of the pruning method allow the learning algorithm to leave potentially sub-optimal local minima.

3 Methods

To answer our research questions, we have evaluated the results found by Frankle and Carbin in support of their hypothesis. We also analyzed evidence found by others in relation to the hypothesis, specifically by Liu et al. [16]. In their work, Liu et al. make certain arguments against the findings of Frankle and Carbin. As we will see, the results of the experiments depend strongly on the precise conditions of the experiment. To help clarify this, we add results from our own experiments as extra evidence.
In most of the experiments, four neural network models can be distinguished.

1. The original network with dense fully connected layers.
2. The network after training and pruning down to a $p\%$ density. This is where classical pruning ends.
3. The pruned sparse network, re-initialized with new random weights and trained again.
4. The pruned sparse network, re-initialized with its original weights and trained again. This is the method proposed by Frankle and Carbin to reveal winning “lottery tickets”.

In many of the experiments by Frankle and Carbin, Liu et al. and ourselves, the MNIST [13] and CIFAR10 [12] data sets are used. MNIST is a data set consisting of 60,000 training images (grayscale of handwritten digits of 28x28 resolutions. CIFAR10 consists of the same number of images, but of a slightly higher resolution (32x32), in color, and of 10 different real-life object classes such as cats and airplanes. CIFAR10 is arguably a larger and more complex data set than MNIST, which is important to take into account in these experiments.
To find winning tickets, Frankle and Carbin make use of convolutional neural network models with fully connected layers. They use optimization strategies such as stochastic gradient descent (SGD), momentum SGD and Adam [11] also making use of dropout [23, 9] and weight decay. The used pruning technique is mainly iterative unstructured pruning such that the winning tickets are sparse.
In [16], Liu et al. compare the performance of the pruned and the non-pruned network with the help of different methods of pruning as discussed in Sec. 2.1. They employ different techniques such as *Scratch-B* and *Scratch-E* which involve training the small pruned models for the same number of epochs (Scratch -E) or training over the same amount of computation budget (Scratch-B). If pruning the model decreases the number of FLOPs[1] by at least a factor of 2, the number of epochs is doubled. Note that since Liu et al. use automatic structured pruning methods, their approach differs fundamentally from Frankle and Carbin’s approach.
In our own experiments, we use a small convolutional neural network applied to the MNIST data set. Our network is based on LeNet [13] and most notably contains three fully connected layers, two of which together make for nearly 95% of the network’s parameters[2]. The model is trained using Adam [11]. With this, we recreate and train the four network models as mentioned before for different pruning factors p. We use iterative magnitude-based weight pruning to find our sparse networks. This simple method, contributed by Zhu and Gupta [26], prunes the weights of the lowest magnitudes until a desired sparsity is reached. All our experiments are implemented with the use of the Python programming language[3] using the libraries TensorFlow[4] and Keras[5] for implementing our neural networks.

[1]Floating point operations, a measure of computational load.
[2]These fully connected layers consist of 48,120 and 10,164 weights respectively. The full network consists of 61,706 weights.
[3]https://www.python.org/
[4]https://www.tensorflow.org/
[5]https://keras.io/

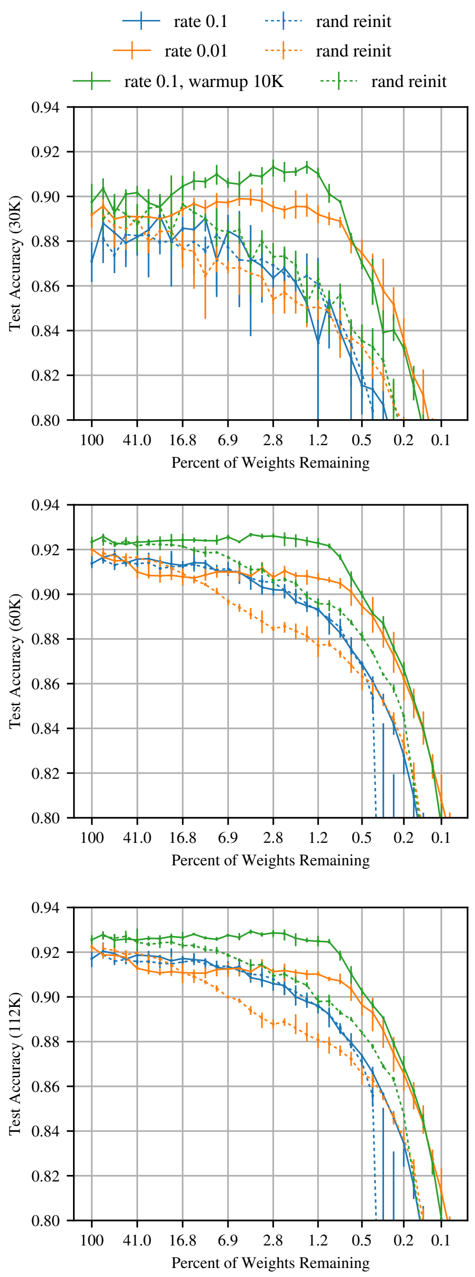


Fig. 4: Adapted from [3], page 7, Figure 7: “Test accuracy (at 30K, 60K, and 112K iterations) of VGG-19 when iteratively pruned.”

4 Results

In this section, we give an overview of some of the results from different sources, as described in Sec. 3. We highlight only some results which we think are important, relevant or exemplary. Equipped with this knowledge, we will discuss our research questions in Sec. 5.

4.1 Experiments by Frankle and Carbin

The pruning technique performed on LeNet [13] is a simple layer-wise pruning heuristic. A certain percentage of weights with the lowest magnitudes in each layer is removed. Connections to output are pruned at half the rate compared to the rest of the network. Here, the MNIST data set is used for training on a LeNet 300-100 architecture. From these experiments, it is concluded that in the case of a fully connected network such as LeNet for a relatively small data set (such as MNIST), the initialization is significant for the performance of the winning ticket.

Using iterative pruning, the winning tickets are found to learn faster than the original network. The winning tickets are pruned iteratively to various extents. It is observed that a "winning ticket" with 51.3% of the weights remaining gives a higher test accuracy than the original network. When the density goes down to 21.1% it achieves this higher test accuracy faster, in other words the training early-stops earlier. At the point where the density becomes 3.6%, the winning ticket performs on par again with the original network.
It is to be noted that winning tickets optimize more effectively but do not generalize better in the case of an early stopping criterion. However at 50,000 iterations, the test accuracy still seems to improve, while the training error is already down to 0. This implies that winning tickets do generalize better.
When the model is randomly re-initialized, it is observed that for a pruning density between 51.3% and 21% the winning tickets learn at a slower rate in the case of random re-initialization and lose test accuracy after a little pruning, which supports the lottery ticket hypothesis.
One-shot pruning helps to identify the winning tickets without repeated training. It is observed from the experiments that the iteratively pruned tickets learn faster and reach higher test accuracy at smaller network sizes. Frankle and Carbin emphasize the need for iterative pruning for their experiments on fully connected networks.
Moving on to slightly larger convolutional neural networks, Frankle and Carbin use scaled down VGG and VGG-19 networks [22] as well as ResNet [7] for experiments on the CIFAR10 data set [12]. The basic experiment is the same as in the case of LeNet. As the network is pruned, it tends to learn faster and the test accuracy is improved. Iterative pruning and random re-initialization for different layers of the ConvNet was performed. The results can be found in [3], page 6, Figure 5. The winning tickets seem to have a higher test accuracy. It is found that the gap between the test and training error is smaller for winning tickets implying that they generalize better. As in the case of LeNet, in ConvNets, the test accuracy is worse with random reinitialization of the winning tickets. However, the test accuracy at early stopping remains steady and may even improve for some layers in the network.
It is noted by Frankle and Carbin that using dropout [23, 9] helps increasing the test accuracy of the network. Here, dropout refers to updating only a random part of the network for each training iteration.
When training on larger VGG and ResNet networks, Frankle and Carbin found the architecture to be sensitive to the learning rate, and warmup was required to find the tickets at a higher learning rate. When using warmup, the learning rate is initially built up from 0 to the initial learning rate over the course of a specified number of iterations. In the case of a lower learning rate, the winning tickets seem to learn faster than the original network in the beginning. This however soon slows down due to the lower initial learning rate. On the other hand, they perform faster when randomly reinitialized. This can be improved by the introduction of the warmup strategy which in turn helps finding winning tickets at a higher learning rate.
Figure 4 shows an overview of results on VGG-19 when iteratively pruned and re-initialized, either randomly or with its original weights. From these graphs, it can be seen that the lottery ticket approach at an initial learning rate of 0.1 and with a 10K iterations warmup performs consistently better than the other approaches, and is able to maintain the same level of accuracy down to approximatly 1.2% of the weights remaining.

4.2 Experiments by Liu et al.

Liu et al. [16] claim that it is not necessarily better to reuse the original weights of the network. Instead, random re-initialization of the weights of the winning tickets can be enough to perform on par with the original unpruned network and the "lottery ticket" configuration
The main difference according to Liu et al. is the result when comparing unstructured pruning on CIFAR. For structured pruning, using either higher or lower learning rates, the winning ticket approach does not outperform the randomly re-initialized network.
In order to prove this, Liu et al. compare the models with Frankle and Carbin's approach as well as randomly re-initializing weights using initial learning rates of 0.1 and 0.01, also using a step-wise delay schedule and momentum SGD. These are then used on the CIFAR and ImageNet datasets, the latter of which is a larger dataset. The pruning methods used here are iterative pruning and one-shot pruning as used by Frankle and Carbin and comparing them with L1-norm based filter pruning. The values for the experiment are shown in Table 8 in [16].

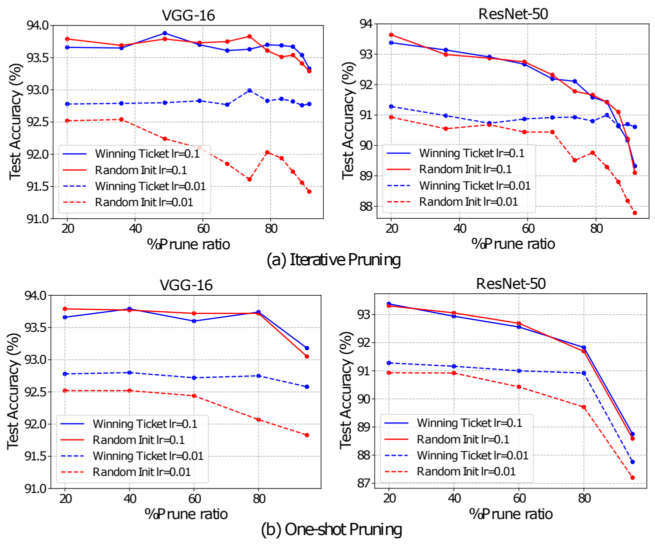


Fig. 5: From [16], page 11, Figure 7: "Comparisons with the Lottery Ticket Hypothesis [3] for iterative/one-shot unstructured pruning [5] with two initial learning rates 0.1 and 0.01, on CIFAR-10 data set."

From Figures 4 and 5 we observe that in the case of unstructured pruning, the winning ticket method of assigning the original weights to the subnetwork works better in comparison to the randomly reinitialized subnetwork if the learning rate is small. In the case of structured pruning, the winning ticket performs on par with the randomly reinitialized subnetwork.
Liu et al. argue that the reason why the winning ticket approach is helpful at low learning rate may be that the weights of the final trained model are not too far from original initialization due to the small parameter step-size.

4.3 Additional experiments

In our own experiments, we put the two main re-initialization approaches to the test. The test (validation) accuracies of our LeNet-based model as described in Sec. 3 are plotted in Figure 6. The accuracy of the fully connected model, i.e. without pruning whatsoever, is shown in the background as reference for the methods under investigation. The pruned network that was used to potentially find winning "lottery ticket" configurations is also regarded a model for reference here as it represents classical pruning. We see that for all pruning methods, the accuracy improves when pruning down to any density of 5% or higher. At 2% and 1% the network seems be too sparse to keep up with the accuracy of the original dense model. As for finding "lottery tickets", we see that only between densities 10% and 2% this approach clearly stands out from the rest. Presumably, a network that is still quite dense can easily find a good solution again with a new random initialization. "Aggressive" pruning seems to be required to make the winning lottery tickets reveal themselves. At the other extreme however, we see that at 1% density the difference between the two methods fades entirely. Both methods under-perform strongly here, which is not surprising because this level of pruning almost guarantees that information from the input is lost to some extent. Another interesting observation is that the pruning network seems to outperform all other models at the higher densities. We think this might be

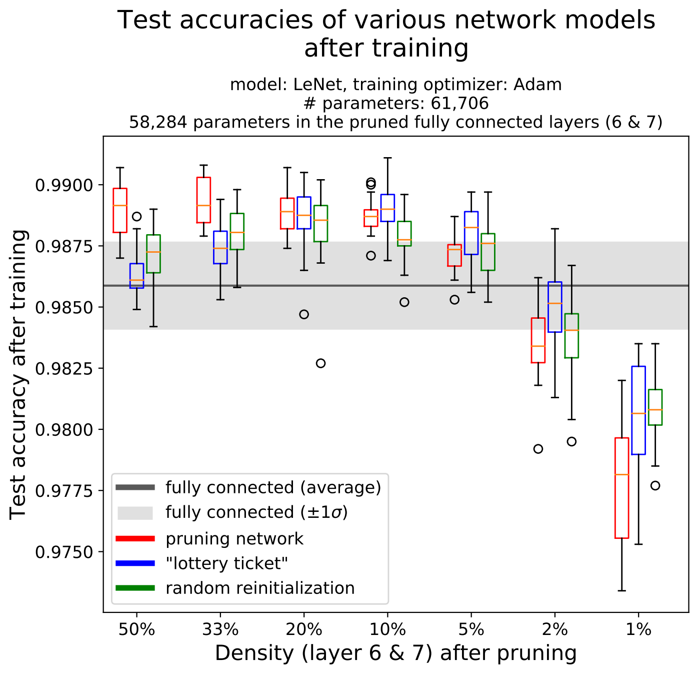
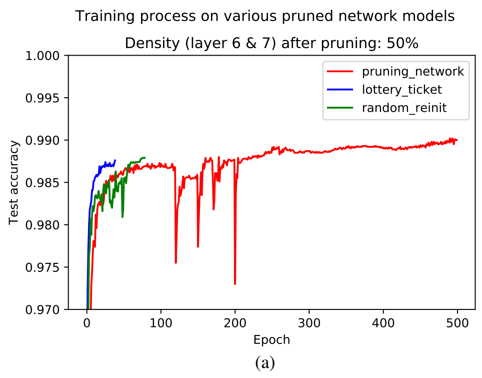


Fig. 6: Results of training and retraining our model for different levels of sparsity after pruning. The measured test accuracies are displayed as boxplots, where the box encompasses 50% of the measurements, with a line denoting the median within. The "whiskers" represent the full range of the data, except for outliers which are shown as separate circles.



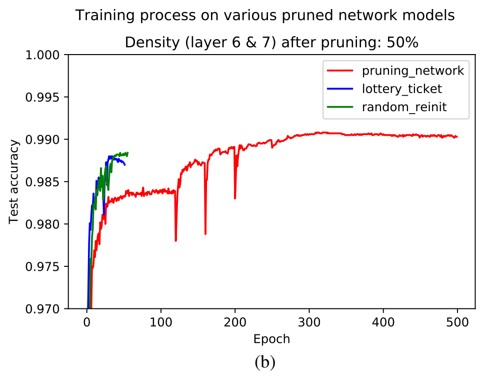


Fig. 7: Training process of the different pruned neural network models. Two instances of training a 50% pruned network were selected which showed interesting features as described in Sec. 4.3.

because the pruning network is not allowed to early-stop, while the re-initialized models are. The extra training on the pruning network may improve the test error, even though the training error is stable at 0. A similar observation was made by Frankle and Carbin (see Sec. 4.1) and we already discussed this general concept in Sec. 2.2. We can substantiate this claim by looking at the training curves in terms of the validation accuracy over the course of the training epochs. As can be seen in Figure 7a, it is possible for the test accuracy to increase in the pruning network over the course of many more epochs, while the re-initialized models are not able to do this because they early-stopped. In other cases, we observe that the pruning itself contributes to an improved test accuracy. As can be seen in Figure 7b, the pruning network benefits from the temporary disruption of the pruning iteration and reaches higher accuracies than it would have if no pruning would have been applied. This observation that pruning and the instability it generates may contribute to training was made earlier in [1] as mentioned in Sec. 2.2.

5 Discussion

Frankle and Carbin [3] observed that their approach of re-initializing weights with their original weights was more effective than re-initializing with random weights when using unstructured pruning. For large models and higher learning rates, the warmup technique was necessary to get better results than with random re-initialization. In case of the lower learning rates, the pruned network learnt faster initially but slowed down over the course of learning due to the small learning step.

Liu et al. [16] performed experiments to gain more evidence and observed that the accuracy achieved when the subnetwork is randomly reinitialized actually performs similar to that of the winning tickets. Thus it is implied that the structure of the network is important rather than the weights. They conjecture that this may occur because in Frankle and Carbin's experiments, the data set was small and the target weights were closer to the values in the original network. Liu et al. argue that the weights may introduce a large bias over the selection of the winning ticket.

Drawing conclusions from both authors and our own experiments, we can substantiate Frankle and Carbin's hypothesis with the found evidence. However, it is a challenging task to find the lottery tickets as described. For various neural network models of different sizes and complexities, more sophisticated learning methods are required. It is unresolved whether it is always possible to find winning tickets in any dense neural network, and if they exist as is claimed by Frankle and Carbin. As was claimed by Frankle and Carbin, it is essentially only possible to find lottery tickets beyond a certain level of sparsity. In other words, networks that remain too dense after pruning will not reveal lottery tickets. This is something that we also saw in our own experiments in Sec. 4.3.

So far, we have only seen evidence of applicability in relatively small networks and when using a relatively low learning rate or a warmup strategy. Indeed, applying the lottery ticket approach to larger and deeper networks has proven harder than for their smaller and more shallow counterparts. Frankle et al. recently published research [4] in which they are able to create well-initialized sub-networks in a deep Resnet-50 network, but only by re-initializing the weights to their values after they had already been trained for a short while. The required training to get the weights in this desired state is admittedly quite little, but it shows that it is significantly harder to apply the hypothesis to these networks.

The lottery ticket hypothesis has already generated a decent amount of follow-up research. This has helped to understand better by which mechanisms the observed effects of the winning tickets work. A notable finding by Zhou et al. [25] is that the process of pruning itself should be considered learning. This is well illustrated in Figure 8, where we see that a "lottery ticket" that was found with Frankle and Carbin's method performs far above the statistical expectation, even before any training. Merely selecting the lottery ticket from the overarching dense network is therefore a form of training. Zhou et al. name these untrained lottery tickets "Supermasks". Malach et al. [18] worked on a similar theory: "a sufficiently over-parameterized neural network with random weights contains a subnetwork with roughly

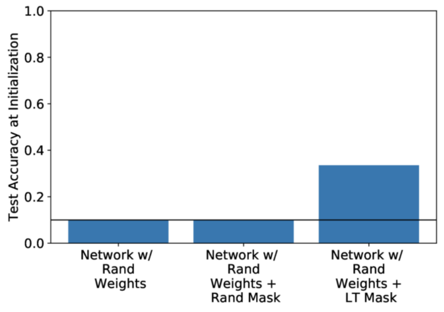


Fig. 8: Networks tested on the MNIST data set. The networks are untrained, i.e. have random weights, but a pruning mask is applied to the second and third model. Taken from `https://eng.uber.com/deconstructing-lottery-tickets/`, a web page about [25].

the same accuracy as the target network, without any further training." This can be thought of as a stronger version of the lottery ticket hypothesis, and Malach et al. claim to prove this stronger hypothesis in their paper. Finding such a perfect subnetwork however, is a computationally hard problem and only works in very over-parameterized models. This problem can therefore be interpreted as a different take on neural network training.

6 Suggestions for future work

The models on which the hypothesis has been applied are quite limited. The concepts contributed by Frankle and Carbin can be extended and applied to many other situations. An interesting example of this is the work by Rahul Mehta [19], who adapted the lottery ticket hypothesis to transfer learning, creating the "Ticket Transfer Hypothesis".
An open question remains the applicability of the hypothesis. In a practical setting, it is usually not necessary to re-initialize the network. Simple pruning and fine-tuning is sufficient to compress the model and achieve the associated benefits. It would be interesting to see more practical applications of the findings of Frankle and Carbin and those inspired by it.
As discussed in Sec. 5, we can interpret pruning as a novel form of neural network training. An important direction for future work should be to see how we can find ways to make use of both types of learning (i.e. "regular" weight optimization and pruning) in the most effective way to train models more efficiently and come up with better architectures.

Acknowledgements

The authors wish to thank Michael Biehl and Allessandro Pianese for reviewing this paper. We would like to thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster.

References

[1] B. R. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher. The generalization-stability tradeoff in neural network pruning. *CoRR*, abs/1906.03728, 2019. URL `https://arxiv.org/abs/1906.03728`. Accessed: February 2020.

[2] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA, 1995.

[3] J. Frankle and M. Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL `http://arxiv.org/abs/1803.03635`. Accessed: February 2020.

[4] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. The lottery ticket hypothesis at scale. *CoRR*, abs/1903.01611, 2019. URL `http://arxiv.org/abs/1903.01611`. Accessed: March 2020.

[5] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[6] S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, Inc.,, Upper Saddle River, New Jersey 07458, USA, 3rd edition, 2009.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `https://arxiv.org/abs/1512.03385`. Accessed: February 2020.

[8] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *CoRR*, abs/1707.06168, 2017. URL `https://arxiv.org/abs/1707.06168`. Accessed: February 2020.

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `https://arxiv.org/abs/1207.0580`. Accessed: February 2020.

[10] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. *CoRR*, abs/1707.01213, 2017. URL `https://arxiv.org/abs/1707.01213`. Accessed: February 2020.

[11] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations*, 2015.

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, Toronto, 2009.

[13] 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.

[14] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990. URL `http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf`. Accessed: February 2020.

[15] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. URL `https://arxiv.org/abs/1608.08710`. Accessed: February 2020.

[16] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *CoRR*, abs/1810.05270, 2018. URL `http://arxiv.org/abs/1810.05270`. Accessed: February 2020.

[17] J. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *CoRR*, abs/1707.06342, 2017. URL `http://arxiv.org/abs/1707.06342`. Accessed: February 2020.

[18] E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need, 2020. URL `https://arxiv.org/abs/2002.00585`. Accessed: March 2020.

[19] R. Mehta. Sparse transfer learning via winning lottery tickets. *CoRR*, abs/1905.07785, 2019. URL `https://arxiv.org/abs/1905.07785`. Accessed: March 2020.

[20] B. Neyshabur, R. Tomioka, and N. Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[21] R. D. Reed and R. J. Marks. *Neural smithing: supervised learning in feedforward artificial neural networks*, chapter 13: Pruning Algorithms. MIT Press, Cambridge, Massachusetts, 1999.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014. URL `https://arxiv.org/abs/1409.1556`. Accessed: February 2020.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. URL `http://arxiv.org/abs/1411.1792`. Accessed: February 2020.

[25] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *CoRR*, abs/1905.01067, 2019. URL `https://arxiv.org/abs/1905.01067`. Related web page: `https://eng.uber.com/deconstructing-lottery-tickets/`. Accessed: March 2020.

[26] M. Zhu and S. Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview, 2018. URL `https://openreview.net/pdf?id=Sy1iIDkPM`, Accessed: March 2020.

An Overview of Workflow Scheduling Algorithms in Cloud

Nivin Pradeep Kumar and Siddharth Mitra

Abstract— Modern scientific applications exploit data from multiple sources. Most of these applications use workflows to model the behavior of the application. Each activity of the workflow is mapped to a particular resource in a particular order to increase the overall performance. The problem of scheduling these activities to the resources is referred to as workflow scheduling. It is an NP-complete problem.
This paper focuses on solving the problem of efficiently running workflow scheduling algorithms in an IaaS based platform. The scheduling algorithms consider various parameters which include, the pay-per-use billing service, satisfying QoS requirements of the users, user-set deadlines and the data transfer costs between resources. We provide an overview of four workflow scheduling algorithms - Particle Swarm Optimization (PSO), Evolutionary Multi-objective Optimization (EMO)-based algorithm, IaaS Cloud Partial Critical Paths (IC-PCP) and a two-phase algorithm - IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). While the first two are meta-heuristic approaches, the latter two are heuristic approaches. These algorithms work according to different scheduling objectives. We will dive deeper into these algorithms in this paper.

Index Terms—Workflow Scheduling, Cloud Computing, Infrastructure As a Service(IaaS)

1 Introduction

Over the past decade, we can observe a steady incline in the number of large-scale applications hosted on cloud based platforms. What we also observe, is an increase in the complexity of these applications. In many cases, the applications involve distributed data sources as well as data and computationally intensive activities. It is a common practice to represent the applications as scientific workflows.

Traditional grid networks provide support for scheduling the workflows but since the advent of cloud computing, running the applications in a cloud-based environment has been proven to be more beneficial. Over the years, cloud computing has been gaining popularity and has become a platform for hosting large scale programs.

A workflow management system (WfMS)[6] is used to manage these workflows when they're made to run in a distributed environment. A few grid based workflow management systems include PEGASUS[2], ASKALON[3] and GrADS[4]. Three main reasons why cloud based platforms are preferred to grid based environments are:

- The ability to dynamically release and acquire resources on demand. The customers are generally charged using a pay-as-you-go-model. This helps the WfMS systems to scale resources according to the budget and deadline requirements of the workflow.
- In cloud based platforms, resources can be directly allocated to tasks. This serves as an advantage when compared to the classic *best effort methods* used to provide resources in grid based environments.
- The cloud provides an illusion of unlimited resources. Hence, the user is allocated a resource as and when required (a high possibility of allocation).

Service Level Agreement (SLA) provided by cloud services allow customers and cloud service providers to agree upon certain standards to maintain minimum levels of service. With the help of SLAs, cloud service providers define the Quality of Service(QoS) for the services provided. QoS requirements focus on a variety of *user-defined* parameters such as budget constraints, workflow deadlines, security, etc. When designing a workflow scheduling algorithm, these QoS requirements are taken into account. The cloud services generally charge the customers by their QoS requirements and execution time. Apart from trying to satisfy user defined QoS requirements, scheduling algorithms also focus on scheduling objectives such as optimizing overall execution time or cost. Hence, workflow scheduling in an Infrastructure as a Service (IaaS) based cloud environment is a challenging task since it has a multi-objective nature.

In this paper, we provide an overview of four cloud - based heuristics : Particle Swarm Optimization (PSO), Evolutionary Multi-objective Optimization (EMO)-based algorithm, IaaS Cloud Partial Critical Paths (IC-PCP) and a two-phase algorithm - IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). We study the previous work done on these algorithms from three main sources, [1],[5], and [7]. All four algorithms have different scheduling objectives. While the EMO-based algorithm focuses on optimizing the makespan and cost of executing the workflow on the cloud, the PSO based algorithm focuses on minimizing both computational cost and data transmission cost. Both IC-PCP and IC-PCP2 scheduling algorithms focus on minimizing the total execution time by satisfying a user-defined deadline.

The remaining portion of the paper is organized as the following. Section 2 describes the general workflow scheduling problem. Section 3 provides a brief explanation of the four scheduling algorithms. This also includes the pseudo codes for the various algorithms. Section 4 compares the four algorithms on the basis of optimization strategy, Scheduling objectives and time complexity.The paper is concluded with section 5. All pseudo codes, equations and mathematical expressions have been referenced from the above mentioned papers.

2 Workflow Schedule Problem

A workflow can be represented as a Directed Acyclic Graph (DAG). Each node of the graph represents a task in the workflow. The directed edges of the graph represent control or data dependencies between the tasks. Hence, a workflow(W) is a DAG with a finite set of tasks T = $\{T_0,T_1,....,T_n\}$ and a finite set representing the control/data dependencies, D = $\{(T_i,T_j)—T_i,T_j \in$ T$\}$.

$$\min F = (makespan, cost)^t \tag{1}$$

$$makespan = FT(T_{exit}) \tag{2}$$

Equation 1 shows the multi-objective function which contains two conflicting objectives - makespan and cost. Makespan refers to the time elapsed from the time at which the first task(T_{entry}) starts and the time at which the last task(T_{exit}) ends. The cost function varies across the applications. It can include the total cost of executing the tasks on different computational resources as well as the cost of transferring

- *Nivin Pradeep Kumar is Master's student in Computing Science at University of Groningen, E-mail: n.pradeep.kumar@student.rug.nl.*
- *Siddharth Mitra is is Master's student in Computing Science at University of Groningen, E-mail: s.mitra.2@student.rug.nl.*

data between resources. Figure 1 shows an example of a DAG which contains five tasks, topologically sorted.

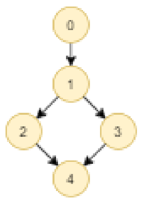


Fig. 1: An example of a workflow represented as a DAG

3 Workflow Scheduling Algorithms

3.1 Particle Swarm Optimization based Algorithm

The Particle Swarm Optimization (PSO) is a self-adaptive global search based optimization method[5]. The logic behind the algorithm is similar to Genetic Algorithms. The PSO technique does not use a crossover operation to produce offsprings but instead relies on the social behaviour of the particles. In every generation, a local best and a global best particle are selected. These particles influence the behavior of the rest of the particles in the population. As a consequence, the randomness within each particle is increased, aiding the algorithm to converge to a *pareto front*.

Besides the properties of the workflow-DAG mentioned in Section 2, the PSO based algorithm adds a few more characteristics to the DAG. These include a set of storage sites S ={1,.....,i} and a set of compute sites PC ={1,.....,j}. The aim of the algorithm is to minimize the total cost of computation of an application workflow[5]. The cost includes the cost of transferring data between computation resources. The edge weight between two tasks t_i and t_j represents the size of the data transferred between the two tasks. The algorithm makes the following assumptions:

- The average computation cost of running a task on an instance for a fixed time is known.
- The cost of transferring a unit data between two instances is known.

The paper [5] describes the problem the PSO - based heuristic as finding a task-resource mapping instance M, such that when estimating the total cost incurred using each instance I_j, the highest cost among all the compute resources is minimized. The total cost of executing all the tasks which are assigned to an instance I_j is shown by $C_{exe}(M)_j$ in equation 3. $C_{tx}(M)_j$ represents the total access cost between tasks assigned to the instance I_j and the tasks which are not. This is shown by equation 4. $d_{M(k1),M(k2)}$ represents the cost of transferring a unit data between instance M(k1) and M(k2). As shown in Equation 5, C_{total} shows the total cost for instance I_j which includes the access cost as well as the execution cost. The equation 6 is considered as the fitness function.

$$C_{exe}(M)_j = \sum_k w_{kj} \quad \forall M(k) = j \tag{3}$$

$$C_{tx}(M)_j = \sum_{k_1 \in T} \sum_{k_2 \in T} d_{M(k1),M(k2)} e_{k1,k2} \tag{4}$$

$$\forall M(k1) = j \;\; and \;\; M(k2) = j$$

$$C_{total}(M)_j = C_{exe}(M)_j + C_{tx}(M)_j \tag{5}$$

$$Cost(M) = max(C_{total}(M)_j) \quad \forall j \in P \tag{6}$$

$$Minimize(Cost(M) \quad \forall M) \tag{7}$$

$$v_i^{k+1} = \omega v_i^k + c_i rand_i \times (pbest_i - x_i^k) + c_2 rand_2 \times (gbest - x_i^k), \tag{8}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{9}$$

here:
v_i^k velocity of particle i at iteration k
v_i^{k+1} velocity of particle i at iteration k + 1
ω inertia weight
c_j acceleration coefficients; j = 1, 2
$rand_i$ random number between 0 and 1; i = 1, 2
x_i^k current position of particle i at iteration k
$pbest_i$ best position of particle i
$gbest$ position of best particle in a population
x_i^{k+1} position of the particle i at iteration k + 1

Algorithm 1 represents a scheduling heuristic for the dynamic scheduling of the tasks belonging to the workflow. This algorithm works on the solution provided by the Particle Swarm Optimization Technique by trying to optimize the cost of the task to resource mapping.

Algorithm 1 Scheduling algorithm

```
 1: Calculate average computation cost of all tasks in all compute resources.
 2: Calculate average cost of (communication/size of data) between resources.
 3: Set task node weight $w_{kj}$ as average computation cost.
 4: Set edge weight $e_{k1,k2}$ as size of file transferred between tasks.
 5: Compute PSO($t_i$) /* a set of all tasks i  k*/
 6: repeat
 7: for all \ready" tasks $t_i \in T$ do
 8:     Assign tasks $t_i$ to resources $p_j$ according to the solution provided by PSO
 9: end for
10: all “ready” tasks
11: Wait for polling_time
12: Update the ready task list
13: Update the average cost of communication between resources according to the current network load
14: Compute PSO($t_i$)
15: until there are unscheduled tasks
```

The Algorithm 2 presents the steps for PSO to produce an optimized task to resource mapping. In PSO, every solution which is part of the population is considered as a particle. The initial population is generated randomly. A fitness function is used to evaluate each particle of the current population. The function is optimized with each new generation. Each particle in the population is aware of its own best position *pbest* and the best position among the particles in the current population, *gbest*. *pbest* is the best fitness value of the particle, achieved so far. *gbest* is the particle with the best fitness value in the population. The values for the velocity and the position of each particle is updated in each generation.

Algorithm 2 PSO algorithm

```
1: Set particle dimension as equal to the size of ready tasks in $t_i \in$ T
2: Initialize particles position randomly from PC = 1, ..., j and velocity $v_i$ randomly.
3: For each particle, calculate its fitness value as in Equation 4.
4: If the fitness value is better than the previous best pbest, set the current fitness value as the new pbest.
5: After Steps 3 and 4 for all particles, select the best particle as gbest.
6: For all particles, calculate velocity using Equation 6 and update their positions using Equation 7
7: If the stopping criteria or maximum iteration is not satisfied, repeat from Step 3.
```

3.2 Evolutionary Multi-Objective Optimization Scheduling Algorithm

As we saw in the previous section, the problem of workflow scheduling has a multi-objective nature and is also known as a Multi-objective Optimization Problem(MOP). A solution to the problem can be represented as a set containing three elements - (*Ins*, *Type*, *Order*).

$$Ins : T \to I, Ins(T_i) = I_j \tag{10}$$

$$Type : I \to P, Ins(I_s) = P_t \tag{11}$$

Ins is a function which maps a task to an instance in the cloud. *Type* is a function that maps an instance in the cloud to an instance type. I is a set of all the available instances in the cloud. Theoretically, this would be an infinite set, since the cloud provides an illusion to the end-user of unlimited resources. P is a finite set in which every element represents a particular instance type the cloud offers as a service.

Order is a vector which contains the scheduling order of all the tasks belonging to the workflow. The vector should satisfy the dependecy constraints of each task. For instance, if T_i and T_j are tasks such that T_i depends on T_j with respect to data or control, the task T_j should appear before task T_i in the *Order* vector.

The logic behind evolutionary algorithms (EA) is driven by the theory behind the process of natural evolution. As discussed earlier a solution, i.e a (*Ins*, *Type*, *Order*) set is considered as a member of the population. Let us observe the various Genetic Algorithm operations provided in [7].

3.2.1 Encoding

To convert the solutions into population candidates the encoding operation is used. As discussed in section 3.2 a solution is a set containing three elements - Order, Ins, and Type. Hence, the chromosome contains three strings to represent the three elements of the solution. The encoding process starts with the topological sorting of the nodes of the graph. This depends on the data and control dependencies between the nodes. For instance, the task T_i will have an index of i. The string representing order contains a permutation of these indexes. The order of the indexes in the string determines the order in which the hosting instance of the tasks will be determined. If the index of task t_i comes before t_j in the order string, then the hosting instance of t_i is determined before task t_j. It does not imply that task t_i will be executed before task t_j. The other two strings are named *task2ins* and *ins2type*, representing the ins and type relations respectively. The *task2ins* string represents a size n vector, in which each task corresponds to a task and the value stored inside that element is the index of the instance to which the task is mapped to. Similarly, the third string *ins2type* is a m-sized vector which maps an instance to its instance type. Figure 2 displays a representation of the encoding.

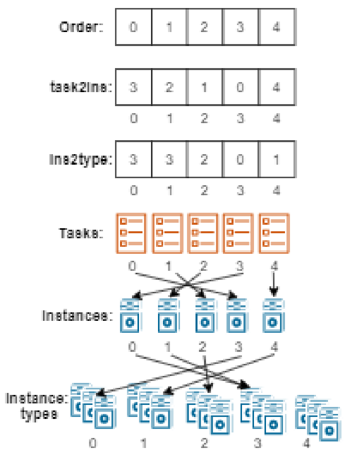


Fig. 2: Encoding

3.2.2 Initial Population

To have the genetic algorithms converge to a *Pareto front*, the initial chromosomes/solutions are generated using initialization methods. The initial population includes two solutions which are generated by known scheduling heuristics:

- HEFT: To generate the fastest workflow schedule.
- A slightly extended-HEFT algorith to generate the "cheapest" solution.

The rest of the solutions are generated randomly. Algorithm 3 shows a procedure RandTypeOrIns contains the pseudocode for the same.

Algorithm 3 The RandTypeOrIns procedure

```
1:  procedure RANDTYPEORINS
2:      $n \leftarrow$ number of tasks
3:      $m \leftarrow$ number of instance types
4:      $order \leftarrow [0, 1, ...., n-1]$
5:      $ins2type \leftarrow replicate(m, RandInt(0, m-1))$
6:      if $Rand(0,1) < 0.5$ then
7:          $task2ins \leftarrow replicate(n, 0)$
8:      else
9:          for all$i \in [0, n-1]$ do
10:             $task2ins[i] \leftarrow RandInt(0, n-1)$
11:         end for
12:     end if
13:     $sched \leftarrow order, task2ins, ins2type$
14: end procedure
```

3.2.3 Fitness Function

The fitness of a solution involves a trade off between the two objectives - makespan and cost. As discussed in section 2, calculating the makespan involves calculating the finish time of the final task $task_{exit}$.The cost function varies according to the platform-specific charging model.

3.2.4 Crossover

To produce the next generation of the population, the solutions of the current generation participate in what is known as a crossover process. In this process, two parent solutions crossover to produce offsprings. The crossover operator for the order vector is provided in Algorithm 4. A random cut-off point is selected which divides a parent string into two substrings. The first substrings of both parents are then swapped and the latter substrings discarded. (Steps 4 and 5). The parent order-strings are then traversed again to append any missed out tasks to the end of the first substrings. (Steps 6-10 and 11-15). Note that the new strings formed do not violate the dependency constraints since the order of any two tasks should have already existed in one parent.

Algorithm 4 Crossover for Order vector

```
1:  procedure CROSSOVERORDER($A, B$)
2:      $n \leftarrow$ number of tasks
3:      $p \leftarrow RandInt(0, n-1)$
4:      $order_a \leftarrow SubString(B, 0, p)$
5:      $order_b \leftarrow Substring(A, 0, p)$
6:      for all T in A.order do
7:          if T not in $order_a$ and T not in $order_b$ then
8:              append T to the end of $order_a$
9:              append T to the end of $order_b$
10:         end if
11:     end for
12: end procedure
```

The crossover of the task2ins and ins2type take place together. Similar to the crossover of the order string, a random cut-off point is chosen which divides the two task2ins strings into two parts each. The first substrings are then swapped. Since the type of instance on which the task is running is important information, the types of instances follow the task. For instance, suppose a task T, which is running on an instance I of type P_s is rescheduled to another instance I* of type P_r, then the type of instance I* is changed from P_r to P_s. Note, the change can potentially cause a break between other tasks and their corresponding instance types. Say a task T' originally scheduled to instance I* is not reassigned to any new instance, would prefer the instance I* to have its previous type P_r. In case such a situation arises, the type of instance I* is chosen randomly between P_s and P_r. For the crossover of the task2ins and ins2type strings the pseudo code for the procedure CrossoverIns is show in Algorithm 6.

The algorithm contains a procedure DecideType which decides on the instance type of the new hosting instance of a task(T). This is done before the swapping of the first substrings of the two parent task2ins strings(step 7). In the DecideType procedure, the types of the instance types of the new instance I', from both individuals(A and B) are taken.(P_a and P_b)(Steps 2-3). Step 3 is responsible for deciding on whether the type of instance I* in B should be changed to P_a or not. To do so, there should not exit a task in B with an index greater than or equal to p-1 which is scheduled to I* then the type of I* is changed to P_a with the addition of a small mutation to cause variation. Else, the type will be randomly chosen between types P_a and P_b.

Algorithm 5 Crossover procedure for Task2Ins and Ins2type

```
1:  procedure CROSSOVERINS($A, B$)
2:      $n \leftarrow$ number of tasks
3:      $p \leftarrow RandInt(0, n-1)$
4:      for $i \leftarrow 0, ..., p-1$ do
5:          $DecideType(T_i, A, B, p)$
6:          $DecideType(T_i, B, A, p)$
7:          $Swap(A.task2ins[T_i], B.task2ins[T_i])$
8:      end for
9:  end procedure
1:  procedure DECIDETYPE($T_i, A, B, p$)
2:      $I' \leftarrow A.task2ins[T_i]$
3:      $P_a, P_b \leftarrow A.ins2type[I'], B.ins2type[I']$
4:      if $\exists j : j \geq p \wedge$B.task2ins[T$_j$] $= I'$ then
5:          if $P_a \neq P_b$ then
6:              $P \leftarrow RandChoice(P_a, P_b)$
7:          end if
8:      else
9:          $B.ins2type[I'] \leftarrow P_a$
10:         Mutate $P_a$ with a small probability
11:     end if
12: end procedure
```

3.2.5 Mutation

The Mutation is performed to introduce variation in a solution. In this case, Algorithm 4 shows the MutationOrder procedure. Taking T as the starting point, the algorithm searches for a substring that contains tasks that are neither predecessors nor successors of T (Steps 4-10). The task T is then moved to a random new position inside the substring(Steps 11-12). The search proceeds in both directions starting from task T and stops when it encounters a task which is either a predecessor or a successor of the task T.

Algorithm 6 Mutation in order vector

```
1:  procedure MUTATEORDER($X, pos$)
2:      $n \leftarrow$ number of tasks
3:      $T \leftarrow X.order[pos]$
4:      $start, end \leftarrow pos$
5:      while $start \geq 0 \wedge X.order[start] \notin pred(T)$ do
6:          $start \leftarrow start + 1$
7:      end while
8:      while $end < n \wedge X.order[end] \notin succ(T)$ do
9:          $end \leftarrow end - 1$
10:     end while
11:     $pos' \leftarrow RandInt(start+1, end-1)$
12:     Move T to $pos'$ in $X.order$
13: end procedure
```

3.3 IaaS Cloud Partial Critical Paths Scheduling Algorithm (IC-PCP)

In this section, we give an overview of the IC-PCP algorithm with some basic definitions. IC-PCP is a one-phase algorithm that has a strategy similar to the deadline distribution phase of Partial Critical Path(PCP). PCP algorithm was proposed for the utility Grid model which has two main phases: Deadline distribution and planning. In the deadline distribution phase, the overall deadline of the workflow is distributed over individual tasks. The Planning algorithm schedules the workflow by assigning each task to the cheapest service which meets its subdeadline.

In IC-PCP, the change being the way subdeadlines are assigned to the task of a critical path. The algorithm tries to schedule them by finding an instance of a computational service which can execute the entire path before its latest finish time.

3.3.1 Definitions

In this scheduling algorithm, we have two variables which represent the start time of the tasks - the Earliest Start Time computed before scheduling the workflow, and the start time which is calculated after the task is scheduled. The Earliest Start Time of each scheduling task t_i is given as:

$$EST(t_i) = \max_{t_p \in t_i' sparents} \{EST(t_p) + MET(t_p) + TT(e_{p,i})\} \qquad (12)$$

where the Minimum Execution Time of a task t_i, MET(t_i), is the execution time of task t_i on a service $s_j \in$ S which has the minimum expected time ET(t_i,s_j) between all available services. The Critical Parent of a node t_i is the unassigned parent of t_i that has the latest data arrival time at t_i, that is, it is the parent t_p of t_i for which $EFT(t_p) + TT(e_{p,i})$ is maximal. The expected time (ET) is the amount of time required for a schedule to complete its task. The Earliest Finish Time, EFT of an unscheduled task t_i and Latest Finish Time of an unscheduled task t_i, LFT(t_i), can be defined as equation 13 and 14:

$$EFT(t_i) = EST(t_i) + MET(t_i) \qquad (13)$$

$$LFT(T_i) = \min_{t_p \in t_i' sparents} \{LFT(t_c) - MET(t_c) - TT(e_{i,c})\} \qquad (14)$$

The most important concept of IC-PCP is the concept of Partial Critical Path(PCP). It is important to define two important variables- Assigned Node and Critical Parent to properly define PCP. In IC-PCP an assigned node is a node that has already been assigned to a scheduled service. The Critical Parent of a node t_i is the unassigned parent of t_i that has the least data arrival time. The critical parent is computed in such a way that EFT(t_p) + TT($e_{p,i}$) is maximum.

3.3.2 IC-PCP Algorithm

The algorithm 7 depicts the pseudo-code of the overall IC-PCP algorithm for scheduling a workflow. The first four lines of the algorithm show the initialization involved. The algorithm sets the actual start time of dummy nodes t_{entry} and t_{exit} and marks them as assigned.

AST(t_{exit}) is set to the user's deadline which is the actual exit nodes of the workflow. Lastly, the AssignParents procedure is called for t_{exit} which schedules all the unassigned parents of its input node. It will schedule all workflow tasks. We have given more insight on this procedure in the next section.[1]

Algorithm 7 The IC-PCP Scheduling Algorithm

```
1: procedure SCHEDULEWORKFLOW($G(T,E),D$)
2:     determine available computational services
3:     add $t_{entry}$, $t_{exit}$ and their corresponding dependencies to G
4:     compute EST($t_i$), EFT($t_i$), and LFT($t_i$) for each task in G
5:     AST($t_{entry}$) ← 0, AST($t_{exit}$) ← $D$
6:     mark $t_{entry}$ and $t_{exit}$ as assigned
7:     call AssignParents($t_{exit}$)
8: end procedure
```

3.3.3 Parents assigning algorithm

The pseudo-code for AssignParents is given in Algorithm 8. The input of the algorithm is an assigned node which schedules all of its unassigned parents before the start time of the input node. The algorithm finds the PCP of the input node. In the first call of the algorithm, it begins with t_{exit} and follows back the critical parents until it reaches t_{entry} to find the overall critical path of the workflow algorithm.

In the next step, the algorithm calls procedure AssignPath which gets a path (an ordered list of nodes) as input and schedules the whole path on the cheapest service which can finish each task before its latest finish time. We have explained more on this in the next section. When a task is scheduled, the ESTs, EFTs of the successors and the LFTs of its predecessors may change, due to this the algorithm updates these values for all tasks of the path in the next loop. After that, the algorithm begins to schedule the parents of each node on the partial critical path, from the beginning to the end of the path, by calling the procedure recursively.

Algorithm 8 Parents Assigning Algorithm

```
1: procedure ASSIGNPARENTS($t$)
2:     while ($t$ has an unassigned parent) do
3:         PCP ← null, $t_i$ ← t
4:         while (there exists an unassigned parent of $t_i$) do
5:             add CriticalParent($t_i$) to the beginning of PCP
6:             $t_i$ ← CriticalParent($t_i$)
7:         end while
8:         call AssignPath(PCP)
9:         for all (t$_i \in PCP$) do
10:            update EST and EFT for all successors of $t_i$
11:            update LFT for all predecessors of $t_i$
12:            call AssignParents($t_i$)
13:        end for
14:    end while
15: end procedure
```

3.3.4 Path Assigning algorithm

AssignPath algorithm shown in Algorithm 9 receives a path as input and schedules all of its tasks on a single instance of a computation service. The condition of the minimum price which can finish each task before its latest finish time is followed. Since all the tasks of the path are scheduled on the same instance, the data transfer time becomes zero, but the data transfer of the outside tasks have to be considered. The algorithm tries to schedule the entire path on the cheapest existing instance for the input path. An instance is applicable if it satisfies the following conditions:

- The scheduled path on the instance should be such that each task of the path is finished before its latest finish time.
- The new schedule uses the extra time of the instance, which is the last remaining time of the last interval of that instance.

Algorithm 9 Path Assigning Algorithm

```
1: procedure ASSIGNPATH(P)
2:     $s_{ij}$ ← the cheapest applicable existing instance for P
3:     if ($s_{ij}$ is null) then
4:         launch a new instance $s_{ij}$ of the cheapest service $s_i$ which
   can finish each task of P before its LFT
5:     end if
6:     schedule P on $s_{ij}$ and set $SS(t_i)$, AST($t_i$) for each $t_i \in$ P
7:     set all task of P as assigned
8: end procedure
```

There are three main things to consider in this algorithm. Firstly, to schedule a path on an existing instance, the algorithm follows two cases - If the instance executes one of the children of the last task of the path, it tries to schedule the entire path right before that child by shifting forward the children and its successors on that instance. Otherwise, the algorithm considers to schedule the path before the start time of the first task of the instance and after the finish time of the last task of the instance. Secondly, the cost of using an existing instance for the input path will be equal to the sum of the new time intervals which are added to the instance for executing the inputting path. Thirdly, if the existing instance executes the parent of a task which belongs to the input path, the data transfer time between them becomes zero when considering that instance for execution of the input path.

3.4 IaaS Cloud Partial Critical Paths with Deadline Distribution Scheduling Algorithm (IC-PCPD2)

IC-PCPD2 algorithm has two phases: Deadline distribution and Planning. In the deadline distribution phase, the overall deadline of the workflow is distributed over individual tasks, and in the planning phase each task is scheduled on an instance of a computation service according to its assigned sub deadline. On the whole, the structure of the algorithm is similar to Algorithm 7, a new line is introduced to procedure Planning after calling AssignParents. However, in the IC-PCPD2 algorithm, a strategy similar to the IC-PCP algorithm is used, the AssignParents procedure tries to assign subdeadlines to all unassigned parents of its input node. The planning procedure carries out the actual scheduling of each task on the cheapest instance that can execute the task before its sub deadline. Furthermore, the initialization of the algorithm is little different, the sub deadlines should be initialized to the same value which means sub deadlines(t_{entry}) ← 0 and subdeadlines(t_{exit}) ← D. We have explained more on this in the upcoming sections.

3.4.1 Parents assigning algorithm

The AssignParents algorithm is similar to Algorithm 8. The main change is when the procedure AssignPath is called, it assigns subdeadlines to all unassigned parents of its input node, instead of scheduling them on a service. The next difference is the concept of the assigned node, which is a node that has already assigned a subdeadline.

3.4.2 Path assigning algorithm

A path, P = $\{t_1,...,t_k\}$, is passed as an input to the AssignPath algorithm. The algorithm assigns subdeadlines to each of its nodes. The algorithm, tries to fairly distribute the path subdeadline among all tasks of the path in proportion to their minimum execution time. The path subdeadline, PSD, is defined as the difference between the latest finish time of the last task, and the Earliest Start Time of the first task on the path. The subdeadline of each task t_i can be defined as shown in equation 15 and 16

$$PSD = LFT(t_k) - EST(t_1) \tag{15}$$

$$subdeadline(t_i) = \frac{EFT(t_i) - EST(t_1)}{EFT(t_k) - EST(t_1)} \times PSD \tag{16}$$

The AssignPath algorithm first computes the subdeadline of each task of the input path using the first equation, and then marks them as assigned nodes.

3.4.3 Planning algorithm

The planning algorithm is depicted in Algorithm 10, which selects the cheapest available instance for each task that can finish the task before its subdeadline. In each stage, the algorithm selects a ready task, a task whose parents have already been scheduled, and schedules it on the cheapest applicable existing instance. The definition of an applicable instance for a task is similar to its definition described in Section 3.3.4. The cost of using the extra time of an existing instance is considered to be zero. If the algorithm cannot find an applicable existing instance, it produces a new instance of the cheapest service which can finish the task before its subdeadline, and schedules the task on the new instance. This procedure repeats until all the tasks are scheduled.

Algorithm 10 Planning Algorithm

```
1: procedure PLANNING(G(T,E))
2:     Queue ← $t_{entry}$
3:     while (Queue is not empty) do
4:         t ← delete first task from Queue
5:         $S_{ij}$ ← the cheapest applicable existing instance for P
6:         if ($S_{ij}$ is null) then
7:             launch a new instance $S_{ij}$ of the cheapest service $S_i$ which can finish t before its subdeadline
8:         end if
9:         schedule t on $S_{ij}$ and set SS(t) and AST(t)
10:        for all ($t_c$ ∈ children of t) do
11:            if (all parents of $t_c$ are scheduled) then
12:                add $t_c$ to Queue
13:            end if
14:        end for
15:    end while
16: end procedure
```

4 Comparison

The table 1 shows the comparison of workflow scheduling algorithm discussed in the paper.

Workflow Algorithms	*Optimization strategy*	*Scheduling Objectives*	*Time complexity*
Particle Swarm Optimization	Meta-Heuristic method	Minimizes both computational cost & data transmission cost	$O(n^2)$
Evolutionary Multi-objective Optimization based Algorithm	Meta-Heuristic method	Optimizes both makespan & cost	$O(kgn^2)$ [7]
IaaS Cloud Partial Critical Paths (IC-PCP)	Heuristic method	Minimizes the total execution time by satisfying a user defined deadline	$O(n^2)$ [1]
IaaS Cloud Partial Critical Paths with Deadline Distribution	Heuristic method	Minimizes the total execution time by satisfying a user defined deadline	$O(n^2)$ [1]

Table 1: Table to test captions and labels

5 Conclusion and Future Work

In this paper, we have given an overview of different heuristic and meta-heuristic workflow scheduling algorithms. We have also summarized the similarities and differences of each algorithm. The following is a brief summary on each algorithm.

PSO: PSO takes into account both computation cost and data transmission cost. The PSO algorithm is similar to other evolutionary algorithms. It takes into account communication costs of all the tasks, including dependencies between them. PSO distributes tasks to resources according to the size of data. It balances the load on compute resources by distributing tasks to available resources. PSO minimizes the maximum total cost of assigning all tasks to resources. The PSO algorithm has a time complexity of $O(n^2)$.

Evolutionary Multi-objective Optimization based Algorithm: The algorithm optimizes both makespan and cost as a Multi-objective Optimization Problem (MOP). It is a suitable scheduling algorithm for most of the workflows which takes into account the instance-based IaaS computing and cloud - specific pricing models. The algorithm provides different trade-offs between cost and time, so that users can choose acceptable schedules according to their preferences. This algorithm is highly promising and has wide range of application.

IC-PCP: IaaS Cloud Partial Critical Paths is a one-phase algorithm. It has a polynomial time complexity. IC-PCP performs better than IC-PCPD2 in many cases. The algorithm aims to create a schedule that minimizes the total execution cost of a workflow, while satisfying a user-defined deadline. The main difference between IC-PCP and IC-PCPD2 is that IC-PCP schedules the workflow in one phase by scheduling each partial critical path on a single instance of a computation service.

IC-PCPD2: IaaS Cloud Partial Critical Paths with Deadline Distribution is a two-phase algorithm. Just like IC-PCP, this algorithm also has a polynomial time complexity. IC-PCPD2 first distributes the overall deadline on the workflow tasks and then schedules each task based on its subdeadlines. Both algorithm IC-PCP and IC-PCPD2 has $O(n^2)$ time complexity.

As part of our future work, we would like to combine these heuristic and meta-heuristic algorithms to produce a hybrid approach. A heuristic algorithm can be used for the initialization of the meta-heuristic method and hence creating a better starting point for the algorithm.

Acknowledgements

The authors wish to thank the University of Groningen for providing the material, guidance and support for writing this paper. We'd also like to thank Nafiseh Soveizi for her support.

References

[1] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[2] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[3] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. Askalon: A grid application development and computing environment. In *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.*, pages 10–pp. IEEE, 2005.

[4] G. Juve, M. Rynge, E. Deelman, J.-S. Vöckler, and G. B. Berriman. Comparing futuregrid, amazon ec2, and open science grid for scientific workflows. *Computing in Science & Engineering*, 15(4):20–29, 2013.

[5] e. a. Pandey, Suraj. *A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments.* 24th IEEE international conference on advanced information networking and applications, 2010.

[6] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. E. Dobson, and K. Chiu. A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency and Computation: Practice and Experience*, 21(16):2118–2139, 2009.

[7] Z. Zhu, G. Zhang, M. Li, and X. Liu. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Transactions on parallel and distributed Systems*, 27(5):1344–1357, 2015.

Consensus mechanisms to manage faults in Distributed Ledger Technologies
Group - 25

Sina Rouzbahani (S4121589)
Shivam Mutreja (S3926575)

Abstract—A distributed ledger is a decentralized database that is distributed across several nodes. In this technology, every node in the network operates autonomously and will maintain the ledger, and if any data changes, the ledger will get updated.
Among various core components, the consensus protocol is the defining technology behind the security and performance of any distributed ledger technology. An important subset of distributed ledger technology is the blockchain network which is created to record transactions or digital interactions and deliver transparency, efficiency, and also provide security for businesses. Many consensus algorithms have been proposed to improve the performance of the blockchain network. Various design choices in the consensus protocol can greatly impact a blockchain system's performance, including its transaction capacity, finality, and fault tolerance.
With more and more blockchain consensus mechanisms being proposed, there is a need to analyze and formally compare them. In the paper, our focus is on private permissioned networks and we will discuss a few famous consensus algorithms such as Algorand, Ouroboros, RAFT, and BFT and the best way that the algorithms perform. Also, we will investigate the criteria such as, robustness, finality, throughput and fault tolerance, under which the algorithms perform well and make comparisons between them.

Index Terms— Consensus, Distributed Ledger Technologies (DLT) , Algorand, Ouroboros, RAFT, BFT.

1 Introduction

Distributed Ledger Technologies (DLT) have revolutionized the world by transforming the existing systems to become more secure, reliable and scalable. Generally, DLTs are designed to deal with database in the form of data shared in a distributed manner, and blockchain represents one possible DLT to do it. It forms a system that provides a trustworthy ledger among a group of nodes across a network that does not fully trust each other.

Now, if the nodes cannot be trusted, it is important to think about adversaries and fault tolerance. Also, even if the nodes can be trusted, how do the nodes reach an agreement about which transaction to accept. Here comes the role of consensus mechanism. In simpler terms, the consensus is a dynamic way of reaching an agreement in a group. While voting just settles for a majority rule without any thought for the feelings and well-being of the minority, a consensus, on the other hand, makes sure that an agreement is reached which could benefit the entire group as a whole. A method by which consensus decision-making is achieved is called consensus mechanism [10].

Consensus algorithms play an essential role to achieve reliability in large-scale systems because they can provide a coherent and consistent group work among a set of machines [17]. In this paper, we will discuss various consensus algorithms, such as RAFT, BFT, Algorand and Ouroboros. We will focus on understanding how each algorithm works and in which situation or network is it suitable to use. We will then compare these algorithms based on a few criteria, namely, robustness, finality and throughput and provide the advantages and disadvantages of each algorithm.

The rest of the paper is organized as follows: Section 2 provides a background of some explanation that we will use in the other sections and the Byzantine generals' problem, Section 3 describes some famous consensus algorithms and divides each of them into three parts, one part talks about the working which also covers how the algorithm deals with Fault Tolerance, the other part of the section mentions the transactions per second the algorithm is able to perform. Section 4 dissects a comparison between algorithms and in Section 5 we conclude what we explained in this paper.

- *E-mail: s.mutreja@student.rug.nl*
- *E-mail: s.rouzbahani@student.rug.nl*

2 Background

Although the terms blockchain and DLT have been used interchangeably between people these days, there is a detailed difference between them. The main difference being that the blockchain is a subset of DLTs and there are other kinds of the ledger. The other difference is that the blockchain is a sequence of blocks while DLTs do not necessarily need such a chain. To drive such a technology where the nodes are distributed, one needs a mechanism to reach agreement among the nodes. Thus, the consensus protocol is the defining technology in any DLTs.[3].

Selecting the wrong consensus algorithm may condition the efficiency, performance, transaction rate, potential vulnerabilities, costs of the product. We investigated an effective algorithm that can guarantee transparency and reliability between nodes in the network.

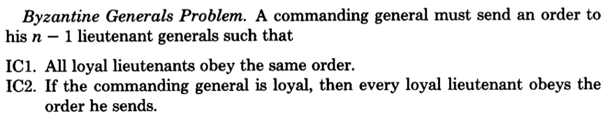
To single out one such consensus algorithm that should be used while developing one's own distributed ledger solution is practically not feasible. It depends on various factors such as, solution availability, which tells the directed audience for the solution that us being developed. It can be different for a private solution (e.g. organization) and different for a public solution (e.g. Bitcoin). It also depends on the nature of the solution being developed, if the solution requires the users with high stakes in the organization to be more powerful or users that have been loyal (w.r.t time) to the organization to hold more decision making rights. Thus, we will explain the aforementioned consensus algorithms in the next section and discuss possible flaws for the readers to make a decision on which algorithm to use.

But before delving into the consensus algorithms we shall understand a fundamental problem in distributed computing i.e. to achieve overall system reliability in the presence of s faulty processes. This problem is very well known as Byzantine Generals' Problem. [14]

2.1 Byzantine Generals' Problem

First, the section discusses about the unsolvable Two Generals Problem. Then, we will extend our learning to the Byzantine Generals' Problem.

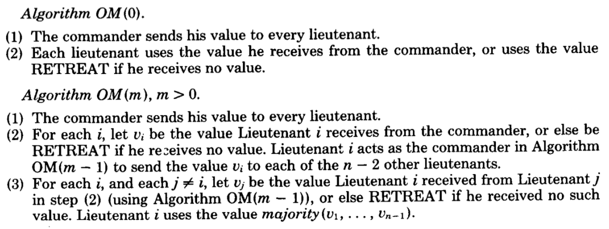
Let us consider a scenario in which there are two generals who aim to destroy the enemy laying siege on different fronts. We will assume General 1 as the leader and General 2 as the follower. If they want to beat the enemy they should attack at the same time. For this to happen they must communicate the time of the attack, this brings us



Byzantine Generals Problem. A commanding general must send an order to his $n - 1$ lieutenant generals such that

IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Fig. 1. Page 3, The Byzantine Generals Problem [14].



Algorithm OM(0).

(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm OM(m), $m > 0$.

(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM($m - 1$) to send the value v_i to each of the $n - 2$ other lieutenants.
(3) For each i, and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM($m - 1$)), or else RETREAT if he received no such value. Lieutenant i uses the value *majority*$(v_1, \ldots, v_{n-1})$.

Fig. 2. Page 7, The Byzantine Generals Problem [14].

to the most important aspect, communication. For them to attack the enemy at the same time they devise a plan to send a messenger across the enemy's camp who will deliver the time of the attack to the follower. Now, the follower has to send an acknowledgement that he has received the message and is ready to attack.

However, in this simple yet strange scenario, there is a possibility that the enemy captures the messenger and the message is never delivered to the follower. If this happens, they would never attack. In the latter scenario, if the enemy gets hold of the messenger and sends a different message, they won't be able to attack together, thus lose the battle. The Two Generals Problem has been proven unsolvable.

Byzantine Generals' Problem is a generalized version of the Two Generals' Problem, but, with a catch. It addresses the same scenario but with more than just two generals, i.e. more than two generals need to agree on the same time to attack the enemy. The added complication is easy to understand as in a real-world scenario, more than one general can be a traitor. As Leslie Lamport stated in [14]: "The leader-follower setup described in the Two Generals Problem is transformed to a commander-lieutenant setup. In order to achieve consensus here, the commander and every lieutenant must agree on the same decision."

Referring to figure 1 which is used by Leslie Lamport in [14], let us add an extra condition to IC2, what if the commander is a traitor, consensus must still be achieved. In this case, for the algorithm to reach consensus the value of the majority of votes/decisions a general observes is important. Thus, all generals take the majority vote.

Oral Message OM(m)

As stated by Leslie Lamport in [14]: "For any m, Algorithm OM(m) reaches consensus if there are more than 3m generals and at most m traitors."

Referring to figure 2, the only way to reach consensus is by having at least $\frac{2}{3}$ or more honest network nodes. And because of this condition, the system is susceptible to attacks in case the majority of the nodes decide to attack the network. (such as the 51 % attack)

If we apply our learning to the context of distributed ledger technologies, each general represents a node in the network, and the nodes need to reach consensus on the current state of the system. If we generalize the idea, the majority of participants in a distributed network must agree on the same action in order to achieve consensus and avoid network failure.

2.2 Paxos

Paxos is one of the many protocols responsible for achieving consensus in a distributed network. Paxos is actually the oldest consensus protocol and was presented by Leslie Lamport in 1989 through her paper [13]. The abstract of the paper published by Leslie Lamport, just contains a single line - "The Paxos algorithm, when presented in plain English, is very simple" [13].

The nodes in a network running the Paxos protocol has three major roles, namely, Proposers, Acceptors and Learners. The point of the aforementioned roles is to agree on a single value. The protocol works in two phases, namely, Promise and Commit.

To understand how this works, let us assume that we have 5 nodes in a network and a client wants to get hold of a lock (think of this as a key to your car). If one person (can be any device) has the lock that means nobody can access the device (in this case the car). The client would ask one of the 5 nodes to provide it with the lock. Now, one node, the proposer, would propose that the lock is provided to the client to the other nodes, namely, Acceptors. This is known as a prepare request. Consider a network partition and there are two Proposers sending a prepare request, how would one differentiate? That is why each prepare request contains a version number attached to it. Now, when the acceptor receives a prepare request, it will respond to the proposer with a promise to decline all the other requests with a lower version number. This is known as the promise phase. Once, the proposer receives the responses from the majority of the acceptors it will send an accept request to each of the acceptors with the same version number. This is known as the commit phase. Before the commit, though the acceptors will check whether they have responded to another prepare request with a higher version number or not, if they have they won't accept the accept request. On the other hand, if they respond positively to the accept request the value is committed and the value is broadcasted to all the other nodes, the learners. This process can go on for a while before a value is accepted. The proposer might not get a majority on the proposed value. But eventually, if the majority of the nodes are alive Paxos will help the network reach consensus.

2.3 Proof of Work

A Proof of Work (PoW) system (or protocol, or function) is a consensus mechanism which is developed to prevents denial service attacks and other service misuses such as spam on a network by requiring some work from service user i.e computational time. The concept was invented by Cynthia Dwork and Moni Naoras presented in a 1993 journal article[6].

PoW is used in a lot of cryptocurrencies that Bitcoin is the most famous one. And also, PoW is used to verify transactions and broadcast new blocks in the blockchain. There is a competition between *minors* on the network to complete transactions or solving computational (Cryptographic) puzzles in the network with PoW. Once a minor can succeed, he can broadcast the block to the network and all the other miners verify the solution is correct. PoW also helps the network to defence against various external attack. One of the disadvantages of PoW is that *mining* claim high computational power and it is expensive. Also, it consumes a large amount of electricity. Due to that, PoW is not the most sufficient consensus mechanism. [19].

2.4 Proof of Stake (PoS)

Some problems in Proof of Work led to another consensus mechanism which is known as Proof of Stake (PoS) which has no miners, instead, it has *Validators* or *Stakeholders* which have a certain amount of money as a deposit i.e. coin ownership on the network known as "Stake" and they are not chosen randomly and also participants are not allowed to mine a new block, instead they *forge* new blocks. The more money deposit, The more possibility to be chosen to forge new block and each chosen node checks whether all the transactions are valid. In comparison to PoW, PoS is more fair, decentralized and secure [1].

3 ALGORITHMS

A consensus algorithm is a way to reach an agreement about the state of the ledger among various nodes in a distributed network. In this way, consensus algorithms achieve reliability in the distributed network and establish trust between unknown nodes in a distributed computing environment.

In this section, we will investigate the four different consensus algorithms namely, RAFT, BFT, Algorand and Ouroboros. Also, study about how they work and consider their performance as well.

3.1 RAFT

Raft is a distributed consensus algorithm that was designed to be easily understood. The goal of Raft consensus algorithm is different from the typical consensus algorithms which agree on a single value. Raft focuses on agreeing on the order in which the operations are committed to a replicated log. A replicated log basically stores the status of various nodes in the network. It solves the problem of getting multiple nodes to agree on a shared state even in the case of failures.

3.1.1 Terminology

- Replicated State Machine - A set of nodes keep identical copies of the same state and can continue operating even if some of the nodes are down. Replicated state machines are used to solve a variety of fault tolerance problems in distributed systems. [18]

- Leader - A leader is elected by all the other nodes. At any point in time, there can be at most one leader. [7]

- Follower - Follower nodes maintain sync with the leader at regular time intervals. If and when the leader node goes down, one of the followers can contest an election to become the leader.

- Candidate - At the time of contesting an election, the nodes can ask other nodes for votes. The nodes asking for votes to be a leader are candidates. Initially, all the nodes are followers. [7]

It is equivalent to the Paxos algorithm proposed for building crash tolerant systems in terms of its fault tolerance properties and performance but is simpler to understand and implement. As Titus von Köller stated in [21]: ”Additionally, the implementation of Paxos in real-world systems has brought on many challenges, problems that are not taken into account by the underlying theoretical model of Paxos.” A fundamental difference between Raft and Paxos is that Raft implements strong leadership. Raft integrates leader election as an essential part of the consensus protocol. All the decisions within the protocol are made by the leader once the leader is elected.

3.1.2 Working

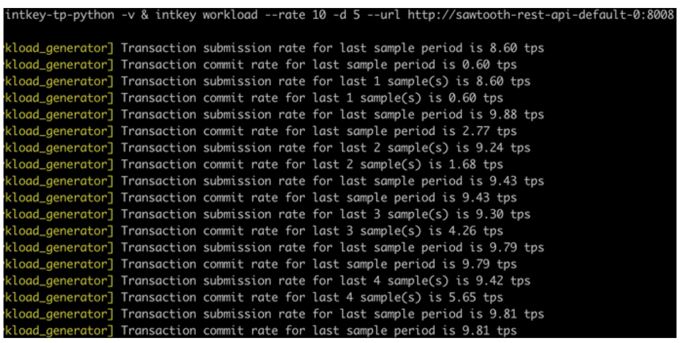
A node can exist in three possible states at one time, namely, leader, follower, or candidate. To tackle problems with clock synchronization in asynchronous systems, where the messages can have arbitrary delays, Raft uses a logical clock in the form of terms. Logical time in Raft divides time into terms of arbitrary length, each of them beginning with an election. If a candidate node wins the election, it remains the leader for the remainder of the term. In case of a vote split, the term would end without a leader.

Each term is identified by a monotonically increasing number, called term number [21]. Each server stores the current term number which is also exchanged in every communication.

Leader election - There are two types of timeouts in Raft, one is known as election timeout, it is the amount of time the follower waits until becoming a candidate. The election timeout is randomized to be between 150ms and 300ms for different nodes. After the election timeout the follower becomes a candidate and starts a new election term. The candidate votes for itself and sends out a vote request to the other nodes. If the receiving node hasn’t voted yet in this term then it votes for the candidate and the node resets its election timeout [21]. Once a candidate has a majority of votes it becomes the leader.

Now, the leader begins to send out Append Entries messages from the client to its followers, these messages are sent in the intervals specified by the heartbeat timeout. The followers then respond to the Append Entries messages and so on, until a follower stops receiving heartbeats and becomes a candidate. This is when the term ends. In case the leader dies due to network partition or some other reason, the other nodes wait for the election timeout and one of them becomes the candidate. If both the remaining nodes become the candidate at the same time, i.e. have the same election timeout, the election timeout starts again as they cannot reach the majority.

Log Replication - Once the leader is elected, all the changes need to be replicated to all the nodes in the system. The same Append



```
intkey-tp-python -v & intkey workload --rate 10 -d 5 --url http://sawtooth-rest-api-default-0:8008

kload_generator] Transaction submission rate for last sample period is 8.60 tps
kload_generator] Transaction commit rate for last sample period is 0.60 tps
kload_generator] Transaction submission rate for last 1 sample(s) is 8.60 tps
kload_generator] Transaction commit rate for last 1 sample(s) is 0.60 tps
kload_generator] Transaction submission rate for last sample period is 9.88 tps
kload_generator] Transaction commit rate for last sample period is 2.77 tps
kload_generator] Transaction submission rate for last 2 sample(s) is 9.24 tps
kload_generator] Transaction commit rate for last 2 sample(s) is 1.68 tps
kload_generator] Transaction submission rate for last sample period is 9.43 tps
kload_generator] Transaction commit rate for last sample period is 9.43 tps
kload_generator] Transaction submission rate for last 3 sample(s) is 9.30 tps
kload_generator] Transaction commit rate for last 3 sample(s) is 4.26 tps
kload_generator] Transaction submission rate for last sample period is 9.79 tps
kload_generator] Transaction commit rate for last sample period is 9.79 tps
kload_generator] Transaction submission rate for last 4 sample(s) is 9.42 tps
kload_generator] Transaction commit rate for last 4 sample(s) is 5.65 tps
kload_generator] Transaction submission rate for last sample period is 9.81 tps
kload_generator] Transaction commit rate for last sample period is 9.81 tps
```

Fig. 3. transaction submit rate of 9.61 tps and transaction commit rate of 8.07 tps

Entries message that was used for heartbeats is used to replicate the changes. First, a client sends a change entry to the leader, the change is appended to the leader’s log. Then, the change is sent to the followers on the next heartbeat. [18] An entry is committed once a majority of followers acknowledge it and finally, a response is sent to the client.

Raft can stay consistent even in the face of network partitions. If there’s a network partition, the node with lower election term will see the higher election term and step down and will roll back their uncommitted entries and match the new leader’s log. This is how the log stays consistent throughout the cluster.

3.1.3 Analysis

The following analysis is done after configuring RAFT mechanism in the Hyperledger Sawtooth blockchain. The experiment was conducted on a MacBook Pro, 2.3 GHz Intel Core i5 processor and 8 GB 2133 MHz LPDDR3 RAM. Docker containers were used to run the experiment and the configuration was made possible by following the official Sawtooth setup guide [4]. We create and submit 10 batches of a transaction and the batch submission rate we get is - 252.66 batch/sec. Referring to 3, if we continuously send in batches with a workload of 10 batches per second, we get a transaction submit rate of 9.61 tps and transaction commit rate of 8.07 tps.

- Raft is easier to implement than other alternatives, primarily the Paxos, because of a more targeted use case segment, assumptions about the distributed system.

- The leader election mechanism employed in the Raft is so designed that one node will always gain the majority of votes within a maximum of 2 terms.

- Raft is strictly single Leader protocol. Too much traffic can choke the system.

- Raft is Crash Fault Tolerant meaning the system can still correctly reach consensus if components fail but not Byzantine Fault Tolerant, which means that the network can do its job even in the presence of malicious actors.

The important features of Raft are election safety and leader append only. Raft is intended for use in a relatively small cluster of nodes of around five or less. Also, the transaction finality, ensures we don’t get forks that would slow down.

3.2 BFT

Byzantine Fault Tolerance is the characteristic which defines a system that tolerates the category of failures that belong to the Byzantine Generals’ Problem [12]. Generally speaking, Byzantine Fault Tolerance(BFT) is the feature of a distributed network to reach consensus even when a number of the nodes within the network fail to respond

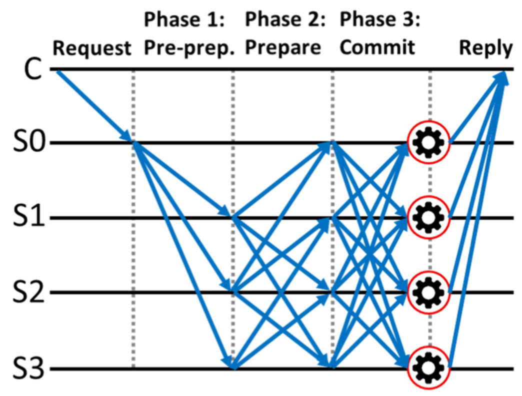


Fig. 4. PBFT consensus in which C is the client. S0 is primary server leader. S1,S2,S3 are replica servers and also it shows PBFT can tolerate one Byzantine failure when $N = 4$ [22].

or respond with incorrect information. The objective of a BFT mechanism is to safeguard against the system failures by employing collective decision making which aims to reduce to influence of the faulty nodes.

Practical Byzantine Fault Tolerant (PBFT) extends and solves the classical problem of Byzantine Generals in Distributed Computing. [9] We will consider the ”generals” in the Byzantine General’s problem section as parties participating in the distributed network. The messengers responsible for carrying messages to either attack or hold position can be closely related to sending transactions back and forth across the network.

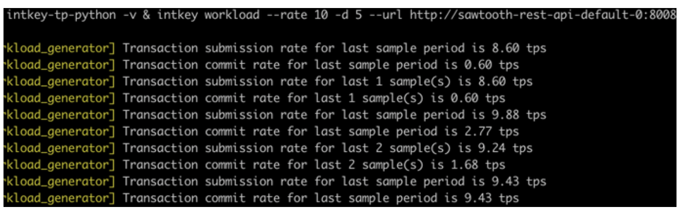
As stated by Chris Hammerschmidt in [9] : ”The collective goal of the ”loyal generals” is to decide whether or not to accept a piece of information submitted to the network as valid or not.” A valid piece of information would be, in our analogy, a correct opportunity to decide in favor of attack.

A PBFT system can function on the condition that the maximum number of adversary nodes must not be greater than or equal to one third of all the nodes in the system. As the number of nodes increase, the system becomes more secure[7]. Communication among nodes has two functions: nodes must prove that message came from a specific node, and that they must verify that the message wasn’t modified during transmission.

3.2.1 Working

The PBFT consensus algorithm is resilient for f Byzantine faults when there are $3f+1$ total nodes [2]. The PBFT algorithm is a three-phase protocol namely, - pre-prepare, prepare and commit. PBFT begins when the client submits a request to the primary node. The primary node starts the protocol immediately and is responsible for advocating for the client request. Suppose, we have a total of 4 nodes, meaning that we should be able to withstand 1 fault since $\frac{1}{4}$ is less than $\frac{1}{3}$. So let’s say one of our 4 nodes drops out due to a poor internet connection. One node might have dropped out, but the other 3 nodes might not know that yet, so they’ll keep sending messages to that node.

The next step is pre-prepare, in which the primary node sends out ”pre-prepare” messages to everyone in the network. A node accepts the ”pre-prepare” message so long as it is valid. The message contains signatures, and other useful metadata that lets nodes determine message validity. If a node accepts a ”pre-prepare” message, it follows up by sending out a prepare message to every other node in the network. And prepare messages are accepted by receiving nodes as long as they’re valid. A node is called prepared if it has seen the original request from the primary node, has pre-prepared and has seen 2f prepare messages that match its pre-prepare - making for $2f+1$ prepares. After nodes are ”prepared”, they send out a commit message. If a node receives $f+1$ valid commit messages, they carry out the client request and finally, send a reply to the client. The client then waits for $f+1$



```
intkey-tp-python -v & intkey workload --rate 10 -d 5 --url http://sawtooth-rest-api-default-0:8008

kload_generator] Transaction submission rate for last sample period is 8.60 tps
kload_generator] Transaction commit rate for last sample period is 0.60 tps
kload_generator] Transaction submission rate for last 1 sample(s) is 8.60 tps
kload_generator] Transaction commit rate for last 1 sample(s) is 0.60 tps
kload_generator] Transaction submission rate for last sample period is 9.88 tps
kload_generator] Transaction commit rate for last sample period is 2.77 tps
kload_generator] Transaction submission rate for last 2 sample(s) is 9.24 tps
kload_generator] Transaction commit rate for last 2 sample(s) is 1.68 tps
kload_generator] Transaction submission rate for last sample period is 9.43 tps
kload_generator] Transaction commit rate for last sample period is 9.43 tps
```

Fig. 5. transaction submit rate of 9.24 tps and transaction commit rate of 6.07 tps

of the same reply, this is because we allow for f faults, thus, we have to wait for $f+1$ responses, as this ensures validity.

3.2.2 Analysis

The following analysis is done after configuring PBFT mechanism in the Hyperledger Sawtooth blockchain. The experimental setup remains the same as for RAFT and the configuration was made possible by following the official Sawtooth setup guide [4]. We create and submit 10 batches of a transaction and the batch submission rate we get is - 152.66 batch/sec. Referring to figure 5 if we continuously send in batches with a workload of 10 batches per second, we get a transaction submit rate of 9.24 tps and for the transaction commit rate, we can see it’s increasing but doesn’t increase steadily. So, the average transaction commit rate of around 6.07 tps.

- The PBFT protocol can process high transaction throughput and can scale incredibly across the network.
- The nature of PBFT means that transactions can be agreed upon and finalized without needing multiple confirmations. There is no waiting period to make sure a transaction is secure after including it in a block. [23]
- The PBFT protocol can process high transaction throughput and can scale incredibly across the network.

3.3 Algorand

Algorand is Byzantine agreement protocol which brings to the table several technical advancements that are essential for Proof-of-Stake blockchains. Before delving into the working of Algorand consensus protocol, let us understand a few terminologies related to the protocol.

3.3.1 Terminology

- Verifiable Random Function (VRF) - The VRF is used to map inputs to a pseudo-random output, with a proof that anyone can use to verify the result. The VRF function is used to choose leaders to propose a block and committee members to vote on a block.
- Participation Key - A user node must be online to participate in the consensus protocol. However, to reduce exposure, online users do not use their spending keys (i.e., the key used to sign transactions) for consensus. Instead, a user must generate and register a specialized participation key before going online. With this key, an online node can participate in proposing and confirming blocks.[16]

Algorand is said to be a fast consensus protocol due to the use of VRF which are used to perform secret cryptographic sortition to select committees to run the consensus protocol [16] which means it could really increase the number of transactions per second. One other improvement that Algorand has made as compared to its counterparts is that it can withstand elongated network partitions and recover very quickly from them, addressing a lot of real-world attacks.

As stated by Silvio Micali in [15]: ”Algorand is based on a new Proof-of-Stake: Pure PoS. Essentially, a Pure PoS does not try to keep users honest by the fear of imposing fines.” Rather, being dishonest by a minority of the money is impossible, and by a majority of the money, stupid.

3.3.2 Working

Algorand uses Byzantine Fault Tolerance protocol called Byzantine Agreement Protocol (BA) in [8] and Verifiable Random Functions (VRF) for the committee election. When a block is proposed to the blockchain, there is a committee of voters that is selected to vote for the block proposal. If a super majority of the votes are from honest participants, the block can be certified [8]. Committees are comprised of pseudo-randomly selected nodes with voting power dependent on their online stake. There are three steps to reach consensus, namely, propose, soft vote and certify vote.

In the proposal phase, each node in the network loops through the nodes it manages and for each node that is online, it runs the Algorand Verifiable Random Function to determine the node to be selected to propose the block. The VRF acts similar to a weighted lottery where the stake of a node participating online affects the node's chance of being selected.

Next step is the Soft Vote phase, the purpose of this phase is to filter down the number of proposals to one, which guarantees that only one block is certified. Each node in the network will get proposal messages from other nodes. Each node is responsible to validate the VRF proof of these messages. In the next step the node will compare the hash from each validated winner's VRF proof to determine which is the lowest and will only propagate the block proposal with the lowest VRF hash. This process continues until the timeout is reached for this step. The next step for all the nodes is to check whether the participating nodes it manages is chosen to participate in the soft vote committee, by running the VRF for all the participating nodes.

If a node is chosen, it will have a weighted vote based on the stake the node has in the network, and these votes will be propagated to the network. These votes will be for the lowest VRF block proposal calculated at the timeout, thus, will be sent out to the other nodes along with the VRF Proof [16]. A new committee is selected at each iteration in the process and each step has a different committee size. A quorum of votes is needed to move to the next step and must be a certain percentage of the expected committee size [15].

The new committee checks the block proposal that won in the Soft Vote stage, for overspending, double spending or any other problems. If valid, the new committee votes again to certify the block [8]. This is done in a similar manner as the Soft Vote. These votes are then collected and validated by each node until a quorum is reached. This is followed by the node creating a certificate for the block and write it to the ledger. At this point a new round is initiated and the process starts over. If a quorum is not reached in a committee vote until a certain timeout, the network will enter recovery mode.

The Algorand protocol is robust against protocol level attacks such as Denial of Service. As Georgios Vlachos stated in [20]: "The Algorand protocol is secure against an adversary who not only deviates from the prescribed protocol rules, but can also launch arbitrary network level attacks." If the network stalls, the nodes move into recovery mode. The nodes then keep sending recover messages to the network, which during network partition are not propagated. These messages are propagated once the network is up and running, and when a required threshold of messages is accumulated, the nodes are synchronized and the system continues to move forward.

3.3.3 Analysis

- Algorand achieves probabilistic finality. As long as the adversary owns less than $\frac{1}{3}$rd of the stake, Algorand can guarantee that the probability for forks is negligible.
- Experimental results show that Algorand confirms transactions in under a minute and incurs almost no penalty for scaling to more users [8].

3.4 Ouroboros

The Ouroboros was developed by Kiayias et al.[11] in 2017 and it has been used for a new cryptocurrency platform known as Cardano. There are 2 famous generations of Ouroboros family: 1) Ouroboros 2) Ouroboros Praos. As we mentioned in the section 2.4, The Ouroboros algorithm belongs to the committee-based PoS which operates a multi-party computation (MPC) scheme to manage a committee to arranged generate blocks.

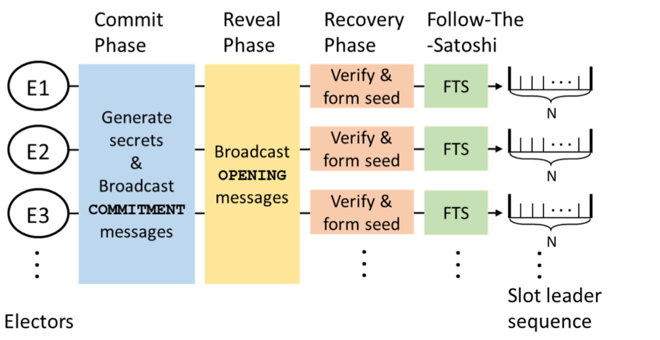


Fig. 6. PVSS-based slot leader sequence generation in Ouroboros [22].

3.4.1 Terminology

- Committee-based Proof of Stake: It defined as a chosen set of participants a *Committee* consists of stakeholders with their deposits or stakes. The committee is safe for generating new blocks in the blockchain. A reliable multiparty computation (MPC) scheme is regularly applied to obtain such a committee in the distributed network. The goal of MPC is creating methods for parties to simultaneously compute a function over individual inputs with the same result as output [22].
- Publicly Verifiable Secret Sharing: In Ouroboros, The Publicly Verifiable Secret Sharing (PVSS) used as the main cryptographic scheme in randomly generation process. There are two schemes: Secret and PVSS scheme. In secret scheme, a "sharer" sends a secret to many "shares". If a single party had access to more than 50% of shares, the original secret can be built. However, in the PVSS scheme, the sharer publicly posts proof that the shares are valid [22].

3.4.2 Working

Ouroboros distributes the actual time into fixed-time epochs and every epoch is again divided into N slots. Only one slot leader can use each of N slots to generate a new block for the network. In each epoch, Electors can elect slot leaders for next epoch through PVSS process. In the Figure 4, there are electors E_i which broadcast a commitment message in the "Commit Phase" involved a random secret. Then, there is a "Reveal Phase" in which E_i broadcasts an opening message that illustrates the earlier sent secret. Next, there is a "Recovery Phase" in which all the electors verify that the opening match and the commitments are match. In the end, all the electors can pas throughout FTS and achieve the same slot leader sequence. As the figure shows, Ouroboros' one-slot-one-leader involves powerful network synchrony and the PVSS-based leader selection may display the elected leaders to targeted attacks[22].

Ouroboros Praos was introduced by David et al. in 2017 to address two security concerns of Ouroboros[5]. It is a provably secure proof-of-stake protocol that is the first to be secure against adaptive attackers and scalable in a really practical sense. Also, it requires stringent network synchrony for slot leaders to use their designated slots correctly, which is vulnerable to desynchronization attacks[22].

In Ouroboros Praos, a Verifiable random function (VRF) used as a core for randomly generation process. There are a private key and an input and there are a pseudo-random number and proof as an output. So with the public key and the proof and the given input, everyone can verify the number was produced but cannot produce the number before that time.

Table 1. Comparing various consensus algorithms for various parameters [22].

Consensus	Guarantees Finality	Throughput(txns/sec)	Robustness (BFT)	Fault Tolerance
PBFT	Yes	Thousands	Yes	33% nodes
Algorand	Yes	Hundreds	Yes	33% token wealth
Raft	Yes	Hundreds	No	50% nodes for Crash FT
Ouroboros and Praos	Yes	Hundreds	Yes	50% token wealth

3.4.3 Analysis

In comparison, Ouroboros Praos manipulates a locally executed verifiable random function (VRF) that makes the elector itself only knows its block proposing for next epoch including VRF proofs verification. The VRF scheme can save much of the communication cost at the local cryptographic computation in comparison to PVSS-based leader election. Some similar schemes are using contemporary protocols such as Algorand. Also, Ouroboros Praos does not restrict the magnitude of consensus participants and provides for a flexible committee.

If we want to mention the differences of Ouroboros and Ouroboros Praos; In Ouroboros slot leaders are recognised publicly in advance and there is always one slot leader per slot, while in Ouroboros Praos, every stakeholder understands which slots they lead in advance, although the others can know when a block is published. Either can be various slot leaders for a slot or no slot leaders at all. Also, Praos has an improved security guarantee.

4 Comparison of Algorithms

In this section we make a performance comparison between the consensus algorithms mentioned throughout the paper. The evaluation metrics that we'll be using are the following :

- Throughput: measured as the number of transactions successfully committed per second.
- Fault Tolerance : Ability of a network to function even after a few nodes of the network fail.
- Finality : It is the affirmation that all well-formed blocks will not be revoked once committed to the distributed ledger.

The Table 1 provides a summary of comparison between consensus algorithms.

5 Conclusion

Every distributed system has a few specific characteristics, such as, concurrency, lack of global lock, fault tolerance and communication between nodes. An algorithm achieves consensus if it satisfies agreement. As we saw during this paper, a thorough review of distributed ledger consensus protocols was provided. We analyzed these protocols with respect to fault tolerance, performance and vulnerabilities. We saw different algorithms have different advantages and varying performance. As a user or developer of a distributed ledger system one should look at one's own requirement and compare different algorithms before starting development. Regardless of how a distributed ledger system is designed, if an adversary is powerful enough to maintain a network partition forever, no blocks will ever be produced. One can just try to maximize the cost for the adversary to do such an act.

Acknowledgements

The authors wish to thank Vasileios Theodosiadis and Vasilios Andrikopoulos for reviewing this paper.

References

[1] M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. McCallum, and A. Peacock. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews*, 100:143–174, 2019.

[2] M. Castro and B. Liskov. Practical byzantine fault tolerance. Massachusetts Institute of Technology, February 1999.

[3] M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han, and P. Sarda. Blockchain versus database: a critical analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1348–1353. IEEE, 2018.

[4] I. Corporation. Configuring and deploying sawtooth raft. https://sawtooth.hyperledger.org/docs/raft/nightly/master/, May 2015.

[5] B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[6] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[7] GeeksforGeeks. *practical Byzantine Fault Tolerance(pBFT)*, July 2017.

[8] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[9] C. Hammerschmidt. Consensus in blockchain systems. in short. https://medium.com/@chrshmmmr/consensus-in-blockchain-systems-in-short-691fc7d1fefe, January 2017.

[10] S. Kadam. Review of distributed ledgers: The technological advances behind cryptocurrency. In *International Conference Advances in Computer Technology and Management (ICACTM)*, 2018.

[11] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[12] G. Konstantopoulos. Understanding blockchain fundamentals, part 1: Byzantine fault tolerance. 2017.

[13] L. Lamport. Paxos made simple, November 2001.

[14] R. S. Leslie Lamport and M. Pease. The byzantine generals problem. SRI International, July 1982.

[15] S. Micali. Algorand's core technology. https://medium.com/algorand/algorands-core-technology-in-a-nutshell-e2b824e03c77, April 2019.

[16] S. Micali. Protocol overview. Algorand Inc, December 2019.

[17] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[18] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. Stanford University, May 2014.

[19] M. Salimitari and M. Chatterjee. A survey on consensus protocols in blockchain for iot networks. *arXiv preprint arXiv:1809.05613*, 2018.

[20] G. Vlachos. Algorand's instant consensus protocol. https://www.algorand.com/resources/blog/algorands-instant-consensus-protocol, May 2018.

[21] T. von Köller. Raft explained - overview of the core protocol. 2017.

[22] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 2020.

[23] W. L. Y. T. H. Yang Xiao, Ning Zhang. A survey of distributed consensus protocols for blockchain networks. Washington University, January 2020.

An updated literature review of service choreography adaptation

Wouter Hertsenberg, Jurgen Nijland

Abstract—Due to increasing demands in service oriented architectures in both classical software systems as well as with the Internet of Things, there is an ever increasing demand for service orchestration and choreography. Especially choreography sees an increase in demand since this removes the need for a central authority which can be regarded as a single point of failure. In this paper, we aim to extend a previous literature review about service choreography from Leite et al. [5]. This paper was published in 2013, which leaves us with another seven years of published papers that we want to analyse to see how the field of service choreography is progressing. We categorise these papers according to which strategy they propose for utilising service orchestration. We have found 21 relevant papers, which we divided into six categories. These categories include the following: model-based, measurement-based, multi-agent-based, formal method-based, semantic reasoning-based and proxy layer-based. On top of that we also look at the research effort of the 21 papers we found with respect to scalability and automaticity.

Index Terms—Service choreography, Choreography adaptation, Choreography customisation, Service composition, Systematic literature review, Extended literature review

1 Introduction

Due to increasing demands in interoperability between information systems, modularity, scalability and flexibility have become important requirements. To address these requirements the Service oriented architectures (SOA) have become available. SOA is a programming paradigm where business logic is divided into services which have their own state. These services are aimed to be loosely coupled, meaning they can be easily replaced or re-used by other services. SOA requires a method to manage and coordinate the operation of all these services.

Service orchestration refers to managing services from a single entity. A depiction of service orchestration can be seen in Figure 1. The central control agent within the system is called the orchestrator. The main language used for describing orchestrations in XML is the Web Services Business Process Execution Language (WS-BPEL) [20]. Within this language Business Processes are described as actions between Web Services.

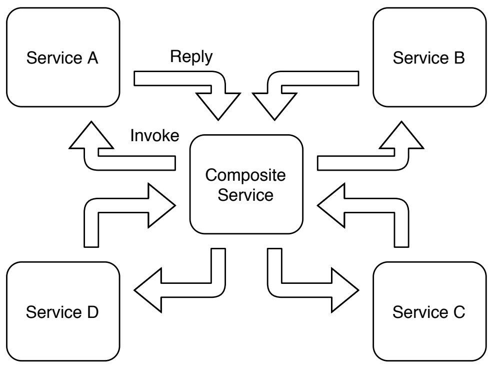


Fig. 1. Service orchestration using a central control agent

On the other hand, Service Choreography is the collaboration between services in a decentralised matter. A depiction of this can be seen in Figure 2. It refers to choreography in the sense of dancing, where dancers move according to a global scenario without a central point of control. Service Choreography is based on ordered message exchanges between services to accomplish a common business goal. There exist multiple languages to express choreography models, however there exists no status quo language [5]. Choreography techniques with distributed data exchange do under certain requirements have better aggregated cost and response time compared to orchestration techniques [10].

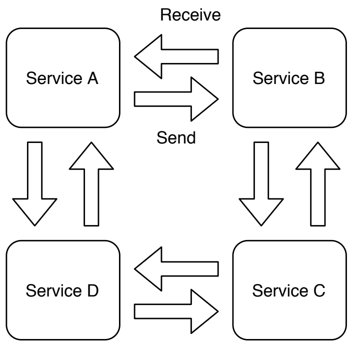


Fig. 2. Service choreography using decentralised data communication

Leite et al. published a systematic literature review of service choreography adaptation in the period between 2005 and 2013 [5]. Due to the rise of the Internet of Things and the increasing adoption of service oriented architectures the demand for orchestration and choreography strategies are increasing in demand. Next to this, service choreography is still an active field of research, so we aim to provide an update of this literature review using the same methods.

Our paper is structured as follows. In section 2 we discuss our methods of finding and selecting the papers. In section 3 we perform an investigatory analysis of the papers that we found and divide them into choreography method categories. In section 4 we further analyse the selected papers and give some background and insight in our decisions. To summarise, in section 5 we conclude our findings and in section 6 we discuss future work on this topic.

• *Wouter Hertsenberg*
E-mail: w.hertsenberg@student.rug.nl.
• *Jurgen Nijland*
E-mail: j.nijland.2@student.rug.nl.

2 Research method

In this section we describe the steps taken in our systematic literature review. First, the research question is presented and explained, followed by the method we used to collect papers we could possibly deem relevant and as last the steps we took to determine whether to include a paper in our systematic review or to exclude the paper.

2.1 Research question

We want to see how the field of research has evolved in the last seven years of publishing this paper. Therefore we formulate the following research question: "how has the research effort in the field of service choreography evolved compared to the years 2005-2012 as published in the paper written by Leite et al. [5]?" To see how the field has evolved we compare the focus categories of each paper we deemed relevant to the focus categories published by Leite et al.. These focus categories can be found in Section 3.2. On top of that we also look at the publication dates of the papers we deemed relevant and how these papers publication dates are distributed over the years. Again the distribution of the papers over the years will be compared to the findings found by Leite et al.. We also look at the research focus areas of the papers we found with respect to scalability and automaticity, but these are less of a focus point for our systemic literature review.

2.2 Collecting possible relevant papers

In order to find relevant papers that are not biased, we need to make sure to use the same keywords for every database or library we search. The keywords we used were taken of the search query published by Leite et al. [5] as can be seen in subsection 2.2. We took their search query since it allowed us to stay close to their results and thus allowed for a better comparison. In this query the papers' abstracts and titles

```
("choreography" OR
 "decentralized composition" OR
 "decentralized service composition" OR
 "distributed composition" OR
 "distributed service composition" OR
 "decentralized interacting services"
)
AND
(custom* OR adapt* OR reconfig* OR
 self-config* OR auto-config* OR
 "self-healing")
```

Fig. 3. Paper search query

are considered. The query consists of two query groups, one for choreography and the other one is for self adaptation. We only want to consider papers that contain both of these concepts, we join the two query groups with an `AND` operator. The '*' in the query indicates a wildcard, meaning that every word that contains the prefix before the wildcard is included in our search result.

Libraries/databases used: We searched for relevant papers in IEEE Xplore [1]. We chose this library since IEEE Xplore is commonly used by academics. This paper could be extended by including additional sources such as ACM digital library, CiteSeerX, Google scholar etc. To limit the scope of this project we only include results from IEEE Xplore.

2.3 Determining relevance of a paper

We used the following approach to determine whether a paper was relevant:

Step 1 Both authors read the paper.

Step 2 Both authors decided if the paper was relevant.

Step 3 If there was a disagreement over the relevance of a paper, said paper was investigated further until an agreement was reached.

[1]`https://ieeexplore.ieee.org/Xplore/home.jsp`

For example the query used as can be seen in subsection 2.2 returned papers in Turkish or about tango dances. Therefore we decided to only review the papers if they were written English. And the papers about tango dances simply did not have anything to do with our topic: service choreography.

2.4 Categorising a paper

The relevant papers also had to be categorised in the categories as described in Section 3.2. This steps taken to categorise the papers until both authors were in agreement are described below.

Step 1 The categories were defined.

Step 2 Both authors read the paper and decided on which categories the paper belonged to according to the category definitions that were determined in Step 1.

Step 3 If there was a disagreement over the categories a paper belonged to, said paper was researched further until both authors reached an agreement.

3 Characterisation of the selected studies

In this section the characteristics of the papers will be presented. In total 37 studies were found of which 21 were deemed relevant.

3.1 Distribution over time

In Table 1 one can see the results found with respect to the year. In Table 1 one can also see the number of papers that were deemed relevant for each year.

Table 1. Number of retrieved papers per year

Year	Query results	Accepted	Accepted (%)
2013	9	7	78
2014	7	4	57
2015	2	1	50
2016	7	5	71
2017	5	2	40
2018	4	3	75
2019	3	0	0
2020	0	0	0

3.2 Paper categories

First, the different categories and their meanings will be explained. These categories are the same as in the paper of Leite et al [5]. Next to the explanation of each category an example paper belonging to each category will be presented. The example papers presented to each category have been chosen based on the ease of explaining the contents of the paper. We have chosen to stick with the same categories to keep consistency between our review and theirs.

Model-based: A category where a paper falls in if it uses a model to support development. On top of that, the paper should talk about high-level abstractions instead of implementation details, like looking at the source code.
An example of a model-based paper is written by Wang et al. [19]. In this paper the authors present a middleware that is chemistry-inspired. This chemistry-inspired middleware models the decentralized service coordination and adaptation as a series of pervasive chemical reactions. This middleware can be executed on three different models: orchestration model (model with a central orchestrator), semi-choreography model (central orchestrator configures the services, so they are "virtually" connected), and the auto-choreography model (a self-managed and self-adaptive cell composition cell is passed among the services in the choreography).
Measurement-based: A category where the paper falls in if a major

focus is put on putting thresholds on agents in service choreography systems. These thresholds are used as indications and the agents try to avoid crossing these thresholds. These thresholds often have to do with QoS (quality of service) parameters.
An example of a measurement-based paper is written by Kothmayr et al.[9]. This paper shows a model that can be reconfigured nearly instantly with a user-interface. The authors first present an abstract description of this approach after which they demonstrate a real-world example. The resulting choreographies that this method offer are both deterministic and have verifiable real-time properties.
Multi-agent-based: If the focus of the paper is on separate agents and how they interact with each other in a service choreography. These agents are mostly autonomous and only communicate with other agents if necessary. On top of that these autonomous agents can learn and analyze to change their actions accordingly to the environment.
The paper written by Herry et al. [7] is an example of a multi-agent-based approach. The authors present a technique that allows users to autonomously reconfigure a computing infrastructure by automatically generating a set of reactive agents.
Formal method-based: These papers talk about applying process calculus or finite state automata to model current systems. These techniques, i.e. process calculus and finite state automata, are used to check the correctness of a system and also to see if the modelled system is actually realizable.
The paper written by Moschoyiannis et al. [11] present a trace-based model to choreography specification. By using this approach one can verify whether the local behaviour of the particpants adheres to the global protocol prescribed by the choreography.
Semantic reasoning-based: Papers in this category reason about the communication between services by using ontologies. On top of that they also reason about the replaceability of a service in a service choreography. In other words, the service tries to understand the data sent by other services by looking at the semantics. The semantics are better understood by using the relevant topology of the service's domain.
The paper written by Weiß [23] is an example of a semantic reasoning-based approach. Weiß et al. present a life cycle that enables the trial-and-error modeling and execution of multi-scale and/or multi-field simulations. This life cycle is also geared towards extending standard-based technologies from business applications in a generic and domain-independent way.
Proxy layer-based: The papers in this category are related to services that focus on sending and intercepting messages in a service choreography. The services in this category use a proxy to make the necessary adaptions to intercepted messages.
The paper written by Salle et al. [15] is an example of a proxy layer-based research. The paper talks about how challenging it is to achieve full automation of protocol coordination and adaptation when composing services that are heterogeneous. The authors conclude that they plan to keep on studying on how to achieve choreography adaptation and evolution through complex data mappings.

Table 2. Categories of the selected papers

Category	Papers	Total
Model-based	[8] [2] [23] [1] [18] [21] [19] [22] [14]	9
Measurement-based	[9] [18] [6] [24] [12]	5
Multi-agent-based	[8] [13] [17] [7] [12]	5
Formal method-based	[2] [11] [3]	3
Semantic reasoning-based	[23]	1
Proxy layer-based	[15]	1

3.3 Scalability & Automaticity

We wanted to further analyse what contributions the selected papers have made to especially automaticity and scalability. The results of our research can be seen in Table 3.
The automaticity meaning whether the paper has a major focus related to the research of automating the system. And with scalability meaning whether the paper focuses a lot on the research related to the scalability of the system. We will briefly discuss what the papers that deal with the automaticity and scalability have found about this requirements with respect to service choreography.

Table 3. Automaticity and scalability distribution of the papers

Category	Papers	Total
Automaticity	[17] [18] [15] [6] [3] [1] [12] [2] [19] [24]	10
Scalability	[6]	1

3.3.1 Scalability

There is only one paper that mentions scalability and means to increase it in their system. The relevant paper concerning scalability is written by Furtado et al. [6]. They state that manual deployment of new instances is not efficient and does not scale. They propose a choreography enactment engine, which can deploy and execute a given composition. It also dynamically reconfigures and provides automatic resource provisioning based on Service Level Agreement constraints. This largely removes the need for human intervention.

Especially in their section about the monitoring for system load, they state the following: "The monitoring probe can also apply predefined filters to the generated event feed. Not only can a filter reduce the number of events reported to the RMA but it can also aggregate events. The frequency at which events are created can be significantly high and, for this reason, filters are essential to achieve good performance and scalability in our architecture". They are taking measures not only to add the scalability by allowing new resources to be allocated to the system, but they are also making sure their automatic monitoring system can scale accordingly as well. If the monitoring service is unable to handle the load, there is no way to automatically know when new resources need to be allocated or new instances need to be launched.

Other papers also mention scalability, however they do not go into detail what part of the scalability issue they actually try to solve. That service choreography methods mention scalability is trivial since the point of choreography is to manage a service composition, which main advantage is scalability. To name a few, one of these papers is Moustafa et al. [12]. Tbis paper mentions a new theoretical model to handle decentralised service interactions even in highly dynamic environments. They test their model in a large-scale environment and achieve polynomial results where previous methods would expect an exponential trend. In more papers we see scalability mentioned as an important attribute [7], [24], [11], however, in no paper we analysed it is the main criterion of the model.

3.3.2 Automaticity

A lot of these papers have at least some mention or influence on the automaticity of web compositions. There are however a few papers that directly work or extend on methods to reduce human intervention. One of the papers that do talk about automaticity is from Vargas-Santiago et al. [18]. The paper they wrote extends IBM's Autonomic Computing initiative [16] with fuzzy logic and ranking to certain attributes of web services to increase its self-monitoring capabilities. IBM's autonomic computing program aims to implement self-configuring, self-healing, self-monitoring and self-optimizing for service compositions.

Another example is also in the subfield of monitoring by introducing self-adaptive monitors [3]. Instead of applying fuzzy logic, this paper tends more towards correctness of configuration changes. This paper is more on the theoretical side and does not go into the implementation. This is also directly visible from Table 2 where it is categorized under formal methods.

And another example is the paper we already mentioned before by Moustafa et al. [12]. Even though they only tested the scalability capabilities of their model, it is built around switching service compositions by detecting Quality of Service degradations of certain services and reconfiguring the system.

4 DISCUSSION

In this section, we will attempt to answer the research question formulated in Section 2.1: "how has the research effort in the field of service choreography evolved compared to the years 2005-2012 as published in the paper written by Leite et al. [5]?"
To give an answer we will first look at the distribution of our selected papers and we will attempt to explain the possible differences we found when comparing to the paper written by Leite et al. After which we will look at the differences in categories compared to the findings published by Leite et al.. We will again to attempt to give an answer to the differences found.
The distribution of the selected papers over the years and the distribution of the papers over the different categories can be found in Table 4 and Table 5 respectively.

Table 4. Distribution of selected papers compared to the year

Year published (Leite et al.)	Quantity	(%)	Year published (our findings)	Quantity	(%)
2005	5	21	2013	7	33
2006	2	8	2014	3	14
2007	4	17	2015	1	5
2008	3	12	2016	5	24
2009	3	12	2017	2	10
2010	4	17	2018	3	14
2011	2	8	2019	0	0
2012	1	4	2020	0	0
Total	24	100	Total	21	100

Table 5. Difference of selected papers with respect to category

Category	2005-2012 (Leite et al.)	(%)	2013-2020	(%)
Model-based	8	33	8	35
Measurement-based	4	17	5	22
Multi-agent-based	4	17	5	22
Formal method-based	4	17	3	13
Semantic reasoning-based	2	8	1	4
Proxy layer-based	2	8	1	4
Total	24	100	23	100

Our papers are less evenly spread out over the years compared to the papers found by Leite et al. as can be seen in Table 4. On top of that, we also have 2 years without any relevant papers, namely the years 2019 and 2020. We are however only 2 months in 2020 which probably explains the lack of relevant papers in 2020. The fact that there are more papers published with respect to service choreography in the years 2013 to 2016 compared to 2017 to 2020 might indicate that the research in service choreography is slowing down. However, the sample size, which is only 21 publications big, is probably not big enough to say with certainty.
As one can see in Table 5 our findings do not seem to differ too much from the findings published by Leite et al.. There are some minor differences like, e.g. a slightly higher percentage of findings in the model-based category or a lower amount of findings in the proxy layer-based category, but in general they appear to have the same distribution over the categories. These differences in findings can probably be attributed to simply not having a big enough sample size. Namely only 24 findings in Leite et al. compared to our 21 findings. Since the findings in Table 5 are looking similar to the findings found by Leite et al. one might conclude that the research focus with respect to the categories in service choreography has not changed much in the period 2013 to 2020 compared to the research done on service choreography in the period 2005-2012.
If one were to look at the scalability differences and compare those to Leite et al. one would find that Leite et al. found 2 papers with respect to scalability and we only found 1 paper. So again similar looking results.
The way we categorized automaticity cannot be compared to Leite et al. since we used a different meaning for automaticity.

5 CONCLUSION

In this paper, we have presented our paper selection strategy. This paper selection strategy was heavily influenced by the paper written by Leite et al. [5]. We have also presented the results we found. In the discussion, we compared these results with the comparison paper written by Leite et al.. So it was only natural for us to take the categories as defined by Leite et al. as our reference categories. We found that, as it currently appears, the service choreography field is still being researched actively and the research focus did not seem to have changed significantly compared to the reference paper.
But it is probably too early to draw a definitive conclusion. This is the case since our paper was written by only 2 people, which meant we only had limited time available to perform research. This caused us to limit our search libraries/database to only one library, IEEE Xplore, in our research. Which in turn caused us not to be able to evaluate only 37 papers. And in order to get a better insight into the service choreography field, more research needs to be done, thus more papers need to be analyzed and more in-depth research of every paper needs to be done.

6 FUTURE WORK

Our literature review is a simple extension on the paper of Leite et al [5]. We have omitted several sources of papers such as CiteSeerX, ACM digital library and more which were included in Leite et al. This paper could be extended by analyzing these paper sources as well.

Next to this, we could also have more thoroughly analysed the new papers and even derive new categories if we deemed them to be necessary. For example, we encountered a few block-chain based technologies proposed for service orchestration. Even though bitcoin, which was one of the main implementations that sparked the public interest in block-chain, was developed in 2008, blockchain only gained popularity a few years after the release [4]. This means that this technology was certainly not a category yet in the previous literature review.
Another point of improvement would be to use more search libraries/databases as Leit et al. did. However doing such a thorough review, as Leite et al. did for more than 400 papers, is not feasible in the time span of writing this paper, especially since we are only with two authors and Leite et al. had a whole team. Having either a larger amount of time, more authors or both could increase the number of paper sources to include in this literature review.
It would also be interesting to see how this field will develop in the next decade. In the future this literature review can be done once again for papers ranging from 2020 to some year in the future. As we have shown the area is still in active research and there is no indication of a decrease in the amount of paper publications in this subject area.

ACKNOWLEDGEMENTS

This paper was written for the Student Colloquium 2020 organised by the Department of Computing Science of the University of Groningen.

References

[1] V. Andrikopoulos, S. G. Sáez, D. Karastoyanova, and A. Weiß. Towards collaborative, dynamic and complex systems (short paper). In *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, pages 241–245, Dec 2013.

[2] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel. Salt: A simple application logic description using transducers for internet of things. In *2013 IEEE International Conference on Communications (ICC)*, pages 3006–3011, June 2013.

[3] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Self-adaptive monitors for multiparty sessions. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 688–696, Feb 2014.

[4] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sep. 2013.

[5] L. Ferreira Leite, G. Oliva, G. Nogueira, M. A. Gerosa, F. Kon, and D. Milojicic. A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, pages 1–18, September 2012.

[6] T. Furtado, E. Francesquini, N. Lago, and F. Kon. Towards an enactment engine for dynamically reconfigurable and scalable choreographies. In *2014 IEEE World Congress on Services*, pages 325–332, June 2014.

[7] H. Herry, P. Anderson, and M. Rovatsos. Choreographing configuration changes. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 156–160, Oct 2013.

[8] A. Kattepur, N. Georgantas, and V. Issarny. Qos composition and analysis in reconfigurable web services choreographies. In *2013 IEEE 20th International Conference on Web Services*, pages 235–242, June 2013.

[9] T. Kothmayr, A. Kemper, A. Scholz, and J. Heuer. Instant service choreographies for reconfigurable manufacturing systems - a demonstrator. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sep. 2016.

[10] D. Liu, K. H. Law, and G. Wiederhold. Analysis of integration models for service composition. In *Proceedings of the 3rd International Workshop on Software and Performance*, WOSP '02, page 158–165, New York, NY, USA, 2002. Association for Computing Machinery.

[11] S. Moschoyiannis, L. Maglaras, and N. A. Manaf. Trace-based verification of rule-based service choreographies. In *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 185–193, Nov 2018.

[12] A. Moustafa, M. Zhang, and Q. Bai. Trustworthy stigmergic service compositionand adaptation in decentralized environments. *IEEE Transactions on Services Computing*, 9(2):317–329, March 2016.

[13] H. N. Nguyen, P. Poizat, and F. Zaïdi. Automatic skeleton generation for data-aware service choreographies. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 320–329, Nov 2013.

[14] S. Parsa, A. Ebrahimifard, M. J. Amiri, and M. K. Arani. Towards a goal-driven method for web service choreography validation. In *2016 Second International Conference on Web Research (ICWR)*, pages 66–71, April 2016.

[15] A. D. Salle, P. Inverardi, and A. Perucci. Towards adaptable and evolving service choreography in the future internet. In *2014 IEEE World Congress on Services*, pages 333–337, June 2014.

[16] D. Sinreich. An architectural blueprint for autonomic computing (ibm white paper). 2006.

[17] C. Stary, A. Fleischmann, and W. Schmidt. Subject-oriented fog computing: Enabling stakeholder participation in development. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 7–12, Feb 2018.

[18] M. Vargas-Santiago, L. Morales-Rosales, S. E. Pomares-Hernandez, H. Khlif, and H. Hadj-Kacem. Towards dependable web services in collaborative environments based on fuzzy non-functional dependencies. In *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 121–129, Oct 2017.

[19] C. Wang and J. Pazat. A chemistry-inspired middleware for self-adaptive service orchestration and choreography. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 426–433, May 2013.

[20] Wei Tan, Yushun Fan, and MengChu Zhou. A petri net-based method for compatibility analysis and composition of web services in business process execution language. In *IEEE Transactions on Automation Science and Engineering*, volume 6, pages 94–106, Jan 2009.

[21] A. Weiss, V. Andrikopoulos, M. Hahn, and D. Karastoyanova. Model-as-you-go for choreographies: Rewinding and repeating scientific choreographies. *IEEE Transactions on Services Computing*, pages 1–1, 2017.

[22] A. Weiß, V. Andrikopoulos, M. Hahn, and D. Karastoyanova. Enabling the extraction and insertion of reusable choreography fragments. In *2015 IEEE International Conference on Web Services*, pages 686–694, June 2015.

[23] A. Weiß and D. Karastoyanova. A life cycle for coupled multi-scale, multi-field experiments realized through choreographies. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference*, pages 234–241, Sep. 2014.

[24] L. Zhang, M. Ma, G. Zhang, and A. S. Lim. Distributed composition services for self-adaptation wireless sensor networks. In *2013 Computing, Communications and IT Applications Conference (ComComAp)*, pages 47–52, April 2013.

Parallel Computation of Connected Component Trees in Giga and Tera-Scale Images

Kevin Gevers and Pieter Jan Eilers

Abstract—Component trees are region-based, hierarchical representations of connected components of an image. A max-tree is a type of component tree that can be used for efficient attribute filtering and labeling. For tera-scale images, the sequential computation is extremely time consuming. To compute these image representations in reasonable time, there is a need to parallelize the construction of a max-tree. Recently, different algorithms have been proposed to construct and filter component trees using parallel methods on distributed and shared-memory machines. In this paper, we analyze several concurrent implementations and compare their advantages, disadvantages, and performance. We discuss three state-of-the-art algorithms, a shared-memory algorithm using a *pilot-tree*, a distributed algorithm using a distributed component forest and boundary trees, and a hybrid-memory algorithm using tuples. It becomes clear that the hybrid-memory algorithm is the fastest for tera-scale images, but is not able to handle images with a high bit depth. The pilot-tree algorithm is not as fast on tera-scale images with a low bit depth but can handle very high bit depth much faster than the other algorithms.

Index Terms—Connected filters, component-trees, max-tree, parallel computing, Mathematical morphology.

1 Introduction

Connected components are typically sets of 4 or 8-connected pixels in 2-D images of maximal extent with the same intensity value. Connected component labeling is used to label each of these sets of an image. This can, for instance, be used for computer vision applications to help detect certain objects. When filtering an image, components with a certain attribute value lower than a given threshold can be removed or preserved. Area filtering, for example, can be used to remove regions of an image with an area smaller than a certain value. In order to label each connected component, we can use a structure called a component tree of max-tree, representing the connected components at every threshold or gray-level of an image. The leaf nodes of a max-tree represent the local image maxima and the root node of the max-tree represents the image minimum or background. Traversing the max-tree gives the inclusion relation of the connected components, and thus can be used for labeling. While this works in reasonable time with a sequential algorithm for smaller images, it becomes too slow for larger images. In astronomy, for example, there is a need to detect structures, like galaxies, on very large surveys. In order to parallelize the algorithm, the image needs to be split up into tiles, so a max-tree can be computed for each tile separately. The difficult and costly part is merging the tiles or local max-trees back together, especially for images of high or extreme dynamic range (XDR). Many current applications in the field of imagery rely on high-dynamic range or floating point imagery, due to increasing sensitivity of equipment and other technological improvements. In 2008, Wilkinson et al. [15] first introduced an efficient parallel computation of components trees which paved the way for the current state-of-the-art in this sub-domain of image processing. Up until recently, there were no parallel algorithms suitable for building max-trees of an image with a bit depth of more than 8 bits per pixel (bits per pixel). New algorithms have recently been developed to solve this problem.

A distributed component forest or DCF is a memory-efficient and fast way of building a max-tree where the entire max-tree does not have to be computed, but it can be represented by a forest of smaller trees. This is especially useful in the case of distributed memory, where every node has to store only the local max-trees of its threads. The *boundary*-tree approach uses such a DCF on distributed memory. The *halo*-approach also computes local max-trees on separate nodes but uses tuples to communicate and correct for adjoining tiles. The local max-trees are computed using shared memory, communicating and resolving the tuples between threads is done using distributed memory, making it a hybrid-memory algorithm. We also look into a shared-memory solution using a *pilot* max-tree, optimized for XDR images. The *pilot* max-tree is, in this case, a max-tree of the quantized image where the intensities are mapped to a reduced range. This tree is constructed first and later refined to the final max-tree in a refinement stage.

In Section 2, we provide a brief introduction to two sequential max-tree algorithms and their implementations. The subsequent Section 3 presents an overview of the proposed state-of-the-art parallel max-tree algorithms. In Section 4, we will compare the different techniques that have been proposed and outline their advantages and disadvantages in terms of memory efficiency, speed-up and ability to deal with XDR images. Section 5 concludes the paper, discussing the best methods to use in different situations and describing possible future work.

2 Related Work

We will first discuss the sequential algorithms for generating max-trees. In the sequential case, the algorithms for building the hierarchical representations of the connected components are well established. The most important ones are leaf-to-root and root-to-leaf algorithms. For root-to-leaf, *flooding* algorithms are most common, the one introduced by Salembier et al. in 1998 [11] is the most commonly used. For leaf-to-root, the *Union-Find* algorithm, introduced by Tarjan [14] in 1975 is most common. Both algorithms can be used to generate a max-tree in order to create the hierarchical representation of the connected components in the image.

2.1 Two Sequential Max-Tree Algorithms

Given a greyscale image f, a *peak component* P_h^i at level h is the set of all pixels of a connected component of the thresholded image $T_h(f)$, where $T_h(f) = \{p \mid f(p) \geq h\}$. Because there can be many connected components at level h, each component

- *Kevin Gevers is with the University of Groningen, E-mail: k.v.f.gevers@student.rug.nl.*
- *Pieter Jan Eilers is with the University of Groningen, E-mail: p.j.eilers@student.rug.nl.*

is indexed by i. Connected components can be organized in a max-tree structure where each peak component P_h^i corresponds to a node C_h^i[9]. Max-tree building algorithms can be divided into two categories. The first one being root-to-leaf *flooding* [11], which starts from the root node, i.e. the pixel with the lowest intensity, and traverses the connected components in a depth-first approach. The leaf-to-root *merging* category starts from the pixels with the highest intensity, the image maxima, and they work on the sorted pixels that are merged into nodes of a tree, based on Tarjan's union-find algorithm[14] proposed in 1975. Every node of the tree can be seen as a peak component. In the following sections, we describe these two main sequential max-tree algorithms which form a basis for the parallel methods described in Section 3.

2.1.1 Flooding

One way to efficiently label all the connected components is by using the tree-based flooding algorithm, originally created by Salambier et al. [11]. This algorithm creates a max-tree, in a root-to-leaf fashion. This means it starts at the pixel with the lowest intensity and performs a depth-first traversal of the connected components at higher intensities. This is done recursively using a hierarchical *"first-in, first-out"* (FIFO) queue. For each peak component, a node in the max-tree is created. A node contains four fields: *Level* the original level of intensity, *NewLevel* the new value after filtering, *Attribute* the value used for filtering, and *Parent* which points to the parent node. Important to note is that a connected component can be connected at the horizontal and vertical edges (4-connected components) or connected at the horizontal, vertical and diagonal edges (8-connected components) for 2-D images. 3-D images can be either 6-connected or 26-connected [13]. Salembier's algorithm was later rewritten in a non-recursive implementation by Carlinet et al. [2]. The pseudocode of which is shown in Algorithm 1. A detailed explanation of this algorithm is shown in the paper by Götz et al. [7].

2.1.2 Union-Find

The union-find algorithm, first introduced by Tarjan [14], was designed to keep track of disjoint sets. Connected components of an image do not overlap, meaning that this algorithm can be adapted to keep track of these components in the image domain. Each set can be represented by a tree, the root of the tree is an arbitrary element of that set. Each root points to itself and all other elements point to the root. To determine if two elements belong to the same set of components, it is logical to check if they have the same root. The quasi-linear approach introduced in [14] uses three important operations. MakeSet(a) which makes a into a singleton, meaning the set containing only a. FindRoot(a), returning the root of the tree containing a and Union(a, b), merging the two sets containing a and b. The union-find approach was adapted for max-trees in [10]. With this adaptation, the root of a set of points now points to a connected component of lower intensity. Only these nodes which have a parent node at lower intensity are necessary to represent the entire tree and hold the correct attribute values. These nodes are known as canonical elements or *level roots*. The sequential Berger algorithm [1], shown in Algorithm 2, uses Tarjan's method to build the max-tree.

3 Concurrent Methods

Parallel computation can use either a shared memory, distributed memory, or a combination of both. In shared-memory algorithms, each process has access to the same memory concurrently, while for distributed memory algorithms, each process has its own memory and has to communicate with the other processes using a messaging system like the Message Passing Interface [6]. In order to parallelize the connected component labeling, the images are split up and divided over a certain amount of processors. Each processor will execute one of the algorithms, resulting in a max-tree for each tile of the image. To get the connected components correctly labeled in the full image, the max-trees have to be joined together. We will discuss three different techniques to do join these max-trees back together. Firstly, we will discuss a technique that first creates a *pilot* max-tree of a quantized image, following is the refinement stage, where the *pilot* max-tree is used to build the final "refined" max-tree. The second technique uses the edges of each tile to create boundary-trees which can be communicated to compute the correct local max-tree of each tile. The last approach uses so-called "Halos", where each tile of the image has a 1-pixel wide strip at each edge, which overlaps with the tile bordering it. This redundant

Algorithm 1 The non-recursive version of Salembier's flooding algorithm as presented by Carlinet et al. [3]

```
1:  procedure PROCESS-STACK(r,q)
2:    $\lambda \leftarrow f(q)$
3:    POP(levroot)
4:    while levroot not empty and $\lambda < f(\text{TOP}(levroot))$ do
5:      INSERT_FRONT(S, r)
6:      $r \leftarrow parent(r) \leftarrow$ POP(levroot)
7:    if levroot empty or $f(\text{TOP}(levroot)) \neq \lambda$ then
8:      PUSH(levroot, q)
9:    ▷ Particular case for the last element:
10:   $parent(r) \leftarrow$ TOP(levroot)
11:   INSERT_FRONT(S,r)
12: function MAX-TREE(f)
13:   ▷ INITIALIZATION:
14:   for all p do $parent(p) \leftarrow -1$
15:   $start\_pixel \leftarrow$ ANY POINT IN $\Omega$
16:   PUSH(pqueue, start_pixel)
17:   PUSH(levroot, start_pixel)
18:   $parent(start\_pixel) \leftarrow$ INQUEUE
19:   ▷ FLOODING:
20:   LOOP
21:     FLOOD
22:     $p \leftarrow$ TOP(pqueue); $r \leftarrow$ TOP(levroot)
23:     for all $n \in \mathcal{N}$ SUCH THAT $parent(p) = -1$ do
24:       PUSH(pqueue,n)
25:       $parent(n) \leftarrow$ INQUEUE
26:       if $f(p) < f(n)$ then
27:         PUSH(levroot,n)
28:         GOTO FLOOD
29:     POP(pqueue)
30:     $parent(p) \leftarrow r$
31:     if $p \neq r$ then INSERT_FRONTS(S,p)
32:   ▷ ROOT FIXING:
33:   while pqueue not empty do
34:     ▷ ALL POINTS AT CURRENT LEVEL DONE?
35:     $q \leftarrow$ TOP(pqueue)
36:     ▷ ATTACH R TO ITS PARENT
37:     if $f(q) \neq f(r)$ then PROCESS-STACK(()r,q)
38:   ▷ PARTICULAR CASE FOR THE LAST ELEMENT, TREE ROOT:
39:   root ← POP(levroot)
40:   INSERT_FRONTS(S,root)
```

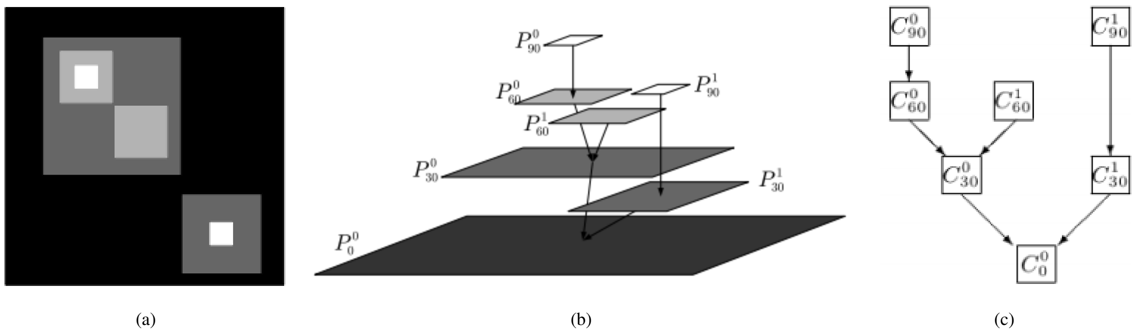


Fig. 1: (a) Original gray-scale image. (b) The peak components of each threshold set. (c) The resulting max-tree. Originals from [5]

Algorithm 2 Pseudocode of the sequential Berger algorithm as presented by [9] which uses Tarjan's Union Find.

```
procedure RUN BERGER(Image f)
  $S \leftarrow$ SORTPIXELSDECREASING(f);
  for all $pixel\, p \in S$ do
    $node[p].parent \leftarrow p$;
    $zpar[p] \leftarrow p$;
    for all pixel $q$ neighbours of $p$ with $zpar[q] \neq -1$ do
      $r \leftarrow$ FINDROOT(q);
      if $r \neq p$ then
        $zpar[r] \leftarrow p$;
        $node[r].parent \leftarrow p$;
        $node[p].Area \leftarrow node[p].Area + node[r].Area$;
procedure FINDROOT(Pixel p)
  if $zpar[p] \neq$ then
    $zpar[p] \leftarrow$ FINDROOT($zpar[p]$)
  return zpar[p]
procedure INIT(Image f)
  for all pixel $p \in f$ do
    $zpar[p] \leftarrow -1$
    $node[p].parent \leftarrow -1$
    $node[p].Area \leftarrow -1$
  RUNBERGER($f$)
```

halo is helpful to merge the information in the different individual trees, in the specific case where we use distributed memory machines.

3.1 Shared-Memory Hybrid Algorithm

Parallelization of state-of-the-art sequential algorithms used for max-tree construction, like the one described in Salembier et al.[11], becomes problematic when dealing with images with extreme dynamic range. Because the complexity of merging sub-trees is proportional to the number of bits per pixel, the cost of merging sub-trees increases as the bits per pixel increase. Consequently, the performance of parallel algorithms decreases drastically if 32-bit integer or floating point representations are used for the input image. To efficiently deal with XDR images, a different approach is proposed in [9] to implement the merging of sub-trees in a two-step parallel algorithm. In the first phase of the parallel hybrid algorithm, a quantized version $\bar{f}$ of the input image f is computed. In this quantized image, the number of bits per pixel is reduced to the number of threads used in the parallel program. This allows for a so-called *pilot* max-tree to be built using any existing parallel method, like the one shown in Algorithm 3. Every thread builds its local max-tree using root-to-leaf flooding. These local max-trees are then merged using the CONNECT procedure as shown in Algorithm 4. The complexity of this CONNECT operation grows exponentially with bit depth, which is the reason a *pilot* max-tree of the quantized image is computed first. The *pilot* max-tree can then be ”refined” into the complete max-tree. This entire process is described as a hybrid algorithm since it uses root-to-leaf *flooding* in the first stage and leaf-to-root *merging* in the refinement stage.

3.1.1 The *Pilot* Max-Tree

The pilot max-tree is simply the max-tree of the quantized image $\bar{f}$. It is important to note that the max-tree of the quantized image does not show any new peak components that were not present in the original image. In Figure 2a, we see a signal representing a 16-bit 1D image, after quantization, the original image is mapped to 4 intensities, any peak component in the original image is flattened to its nearest lower intensity value in the quantized image. This fact is exploited and a relation is enforced where the level root nodes of the *pilot* tree are a subset of the level root nodes of the refined tree. A stricter definition of a level root has to be made to ensure this correspondence. The level root is no longer an arbitrary pixel, instead, it is chosen to be the pixel with the lowest coordinate among the pixels belong to the same component. The original flooding algorithm is adapted slightly in [9] to keep the level root with the lowest coordinate in each component.

3.1.2 The Refinement Stage

The next step is the refinement stage, where the *pilot* max-tree is ”refined” until the final max-tree is computed. The *pilot* max-tree is not modified, so all threads can safely access it. The image is partitioned according to its intensities. Each thread of the refinement stage uses the sequential Berger algorithm on the pixels of its partition. The Berger algorithm uses a sorted array of pixel intensities and processes them from high to low. A parallel version of Radix sort is used to sort the pixel coordinates in the original image based on their intensities, from low to high[4]. The final tree is created in parallel using K threads, working on the original pixel values of f. Each thread receives a partition S_i of the sorted pixels corresponding to the same quantized intensity and processes them in descending order. A parallel stable radix sort is used, meaning the pixels with equal intensity retain their original order, i.e. from lowest to highest coordinate. This ensures the last processed pixel of every con-

Algorithm 3 Concurrent Construction of the Max-Tree for thread th on K threads [9].

```
1:  procedure PARELLEL HIERACHICAL QALGORITHM(Thread th, Image partition P)
2:    $m \leftarrow argmin(P)$
3:    Add $m$ to the $Queue$ at level $P(m)$
4:    $isVisited[m] \leftarrow true$
5:    $levelroot[P(m)] \leftarrow m$
6:    FLOOD($P(m)$, $P$, $0$)
7:    $i \leftarrow 1; q \leftarrow th$
8:    while $(th + i < K) \wedge (q\%2 = 0)$ do
9:      Wait to glue with right-hand neighbour
10:     for all Edges(u,v) b/w partition $P_{th}$ abd $P_{th+i}$ do
11:       CONNECT($th, i, (u, v)$)
12:     $i \leftarrow 2 * i; q \leftarrow q/2$
13:   if $th \neq 0$ then
14:     Notify left-hand neighbour
15:     Wait for Thread 0
```

nected component in the original image f, will be the one with the lowest image coordinate. This fact ensures that the level roots of the *pilot* max-tree are a subset of the level roots of the refined tree.

Algorithm 4 Code of the CONNECT Procedure. Symbol $\perp$ is defined as the root node of every sub-tree and $f(\perp) = -\infty$ [9]

```
1:  procedure CONNECT(Thread $th$, Edge$(u, v)$)
2:    $area \leftarrow 0; areatemp \leftarrow 0$
3:    $x \leftarrow$ GETLEVELROOTOF(u)
4:    $y \leftarrow$ GETLEVELROOTOF(v)
5:    if $f(x) < f(y)$ then
6:      Swap$(x, y)$
7:    while $x \neq y \wedge x \neq \perp$ do
8:      $z \leftarrow$ GETLEVELROOTOF($node[x].parent$)
9:      if $f(z) \geq f(y) \wedge z \neq \perp$ then
10:       $node[x].Area \leftarrow node[x].Area + area$
11:       $x \leftarrow z$
12:     else
13:       $areatemp \leftarrow node[x].Area + area$
14:       $area \leftarrow node[x].Area$
15:       $node[x].Area \leftarrow areatemp$
16:       $node[x].parent \leftarrow y; x \leftarrow y; y \leftarrow z$
17:   if $y = \perp$ then
18:     while $x \neq \perp$ do
19:       $node[x].Area \leftarrow node[x].Area + area$
20:       $x \leftarrow$ GETLEVELROOTOFNODE($node[x].parent$)
```

3.2 Distributed Boundary Max-Tree Algorithm

For the boundary tree approach [5] the image is again cut up into as many tiles as there are processors available. Each processor computes the max-tree for the tile of the image it received. In order to merge the tiles back together a boundary tree is used. The boundary of a tile is the one-pixel wide contour of that tile. The boundary tree is a sub-tree of the max-tree, consisting of only the nodes that touch the boundary of a tile. When merging two tiles back together, only their respective boundary trees have to be merged. The changes during the merging in the boundary trees need to be propagated back to the original max-trees of the tiles. Doing so results in max-trees that are corrected for adjoining tiles, while not having to merge their full max-trees. This reduces the memory complexity of each message to $O(G\sqrt{N})$, where sending the full max-tree would have a memory complexity for each message of $O(N)$. Wilkinson et al. state in their paper that "for an 8 bit-per-pixel $40,000^2$ tile this is a savings of at least a factor of 78 in communication and memory overhead" [8].

The max-tree of each tile is computed using the algorithm as proposed in Wilkinson et al. [16]. Each processor does this sequentially for the tile it received. This means the memory is distributed and processors need to communicate with each other to take the adjoining tiles into account. To do so each processor creates a boundary tree for the tile it received. A boundary tree consists of a one-dimensional array with all the max-tree nodes that touch the border of the tile in it, as well as their parents.

To build the boundary tree, each edge is added in the following order: north, east, south, and west. Afterward, the parents of all the nodes in the edges of the tile are added. To make sure every node is only added once, they are flagged when they get added. Now that the boundary trees have been generated, the tiles can be merged. This is done by traversing the tree until the bottom is reached for each node in the border. Every time there are three options for a node that needs to be merged:

1. The parent of the current node is the root node, in which case this branch is done.

2. The parent of the current node is not in the merged tree yet. Add the node by accumulating the area of the node and point to its parent. Then continue with that parent.

3. The parent of the current node is already in the merged tree. Add the accumulated area of this node to it and point to it. Continue with its parent to add the accumulated area.

This has to be repeated until all tiles are merged back together into a single boundary tree. This means that the max-tree of each tile has been corrected as well.

3.3 Hybrid-Memory Halo Approach

In the *halo* approach [7] to parallelize connected component labeling, the image is cut up into tiles, while having a 1-pixel wide boundary or "*halo*" overlap with adjoining pieces. Each tile is assigned to an MPI process and generates a max-tree by using the shared-memory parallel flooding algorithm discussed based on Algorithm 1. A detailed explanation can be found in the paper by Götz et al. [7]. Each max-tree only has the information of its tile, which means they can be incomplete. The processes need to communicate with each other to correct each tree and take all the tiles into account. In order to join the tiles correctly, after each processor has finished computing their max-tree, two types of tuples are created. These are communicated and resolved simultaneously across all processes using Open-MPI communication techniques. The first set of tuples is for the locally determinable edges of the boundary trees in the halo zone. The second set of tuples is for the information about components that have been split due to the image division and their canonical points. These two sets of tuples are resolved simultaneously using MPI communication techniques and distributed to their corresponding MPI process so the tile's max-tree can be corrected accordingly. This means each tile now has a max-tree that has been corrected for the entire image, thus their collection is the connected (distributed) component forest.

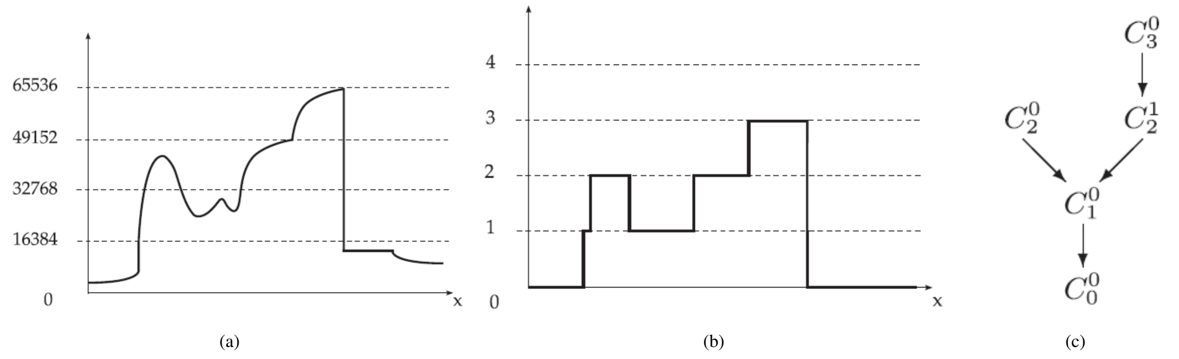


Fig. 2: (a) Represents a 16-bit integer 1D image, (b) is a quantized image after mapping the original intensities onto 4 intensities (c) illustrates the max-tree structure of connected components of the quantized image shown in (b). Originals from [9]

4 Comparison

We refrained from using actual speed-up numbers since the different algorithms have been tested on different supercomputer clusters and different images.

In order to compare the efficiency of the different algorithms explained in the previous sections, we will a consider number of import aspects. Speed-ups being the obvious one, both compared to the sequential algorithm and compared to the other parallel algorithms. Speed-up is generally defined as $S_p = t_1/t_p$. The execution time on a single process divided by the execution time on p processing cores. One can say that the algorithm with the best speed-up will be the best choice. However, another important feature to look at is memory use. When we have giga- and tera-scale images, it may become impossible to store the entire max-tree in memory. Another pertinent issue to look at is an algorithm's ability to deal with images with a gray-level depth of 32 bits per pixel or higher.

We will compare the three algorithms discussed in this paper here. Two of which use a distributed component tree algorithm to compute the max-tree. The boundary-tree approach[8][5] and the halo approach [7] respectively. The halo approach also uses shared memory for the local max-tree computation of each process. The remaining algorithm uses a two-stage shared-memory max-tree approach with a *pilot* max-tree [9].

4.1 Memory Efficiency

Looking at the distributed case, both the boundary tree and the halo approach perform extremely well in regards to memory efficiency. As the number of processes increases, the memory usage per process scales down linearly, for images up to 16 bits per pixel. This allows for the distributed storage of a forest of max-trees of a giga- or tera-scale image which can then be filtered and the final output image can be reconstructed. If we look at the *pilot* algorithm, a shared-memory approach, the total memory cost of constructing the final max-tree will be $9N$ [9], with N being the size of the original image. This is $3N$ more than the sequential Berger algorithm, this is due to the need to store the quantized image and the *pilot* max-tree in memory. Consequently, for a considerably large image, it might be impossible to calculate the max-tree due to a lack of available memory.

4.2 Speed-ups and Dynamic Range

Both the halo approach and the boundary tree approach have a quasi-linear speed-up, using images with 8 bits per pixel, with the speed-up increasing, for up to at least 256 cores. Accounting for the fact that these algorithms were tested on different images and different machines, both absolute execution time and speed-up are about equal for images of 8 bits per pixel. Once we observe the results to 16 bits per pixel [7] [5], we see that the halo approach has considerably better execution time and speed-up.

The issues with these algorithms come with images having a higher dynamic range of 16 bits per pixel and up. With the boundary tree approach, the size of the boundary trees increases when the bit depth increases resulting in a sharp decrease in speed-up as the number of processes increase. This is due to the large overhead of having to communicate all the trees between the cores. Similarly, with the halo approach, the number of tuples increases sharply as the dynamic range increases. The speed-up curve flattens at 64 and 16 cores for 16 and 32 bits per pixel images respectively.

In the paper by Götz et al. [7], a comparison is drawn between their proposed *halo* algorithm and the shared-memory *pilot* max-tree algorithm by Moschini et al. [9]. While they only tested the *pilot* max-tree algorithm for up to 24 threads, it is quite clear that for images of 8 bits per pixel, the hybrid-memory halo approach outperforms the *pilot* max-tree algorithm in terms of execution time and speed-up as the number of threads increases. However, looking at images of 16 bits per pixel, the performance is quite similar, at least up to 24 threads. They make no comparison on images of 32 bits per pixel, as Moschini's algorithm did not terminate, likely due to a lack of available memory. The image they used was a 9 Gigapixel image of the Milky Way. Moschini uses a crop of the same image to show speed-up numbers of approximately five-fold greater than the halo approach for 32 threads. Where the speed-up of the halo algorithms plateaus after 32 threads for images of 32 bits per pixel, the *pilot* max-tree approach shows an increase in speed-up for at least up to 64 threads.

5 Conclusion

In this paper, we discussed different techniques implementing parallel max-tree algorithms for connected component labeling. It became clear that the three main approaches discussed focus on different aspects of the process. The *pilot* max-tree approach focuses on extreme dynamic range and uses shared memory. The boundary tree approach focuses on parallelization of giga-pixel images of 8-bit and 16-bit per pixel, while the halo approach is able to handle image up to floating point precision. Making the halo approach the better choice between the two for processing giga-pixel images in parallel. The halo approach has similar speed-ups with 8bits per pixel and it can handle a gray-level depth of up to 32bits per pixel with significant speed-up. The paper by Markus Götz et al.[7] concludes that their approach has better performance than any state-of-the-art shared-memory algorithm, we disagree with this statement, as it is clear that the *pilot* max-tree algorithm by Ugo Moschini et al. [9] is better suited for XDR images, especially for 64-bit images. Moschini's *pilot* max-tree algorithm is not as fast as the other algorithms for giga-pixel images with low bit depths, so it should only be used for images with a high bit depth. A significant disadvantage of Moschini's two-stage algorithm is memory consumption, if we are dealing with gigapixel images, the available memory might be insufficient when using this approach.

While we could have tested the algorithms ourselves on similar images, it was infeasible in the given time frame. A future project would include testing all the current state-of-the-art parallel max-tree algorithms on identical images and high production clusters.

REFERENCES

[1] C. Berger, T. Geraud, R. Levillain, N. Widynski, A. Baillard, and E. Bertin. Effective component tree computation with application to pattern recognition in astronomical imaging. In *2007 IEEE International Conference on Image Processing*, volume 4, pages IV – 41–IV – 44, Sep. 2007.

[2] E. Carlinet and T. Géraud. A comparative review of component tree computation algorithms. *IEEE Transactions on Image Processing*, 23(9):3885–3895, Sep. 2014.

[3] E. Carlinet and T. Géraud. A comparative review of component tree computation algorithms. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 23, 07 2014.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[5] S. Gazagnes and M. Wilkinson. Distributed component forests in 2-d: Hierarchical image representations suitable for tera-scale images. *International Journal of Pattern Recognition and Artificial Intelligence*, 33(11 SI), 10 2019.

[6] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 2014.

[7] M. Götz, G. Cavallaro, T. Géraud, M. Book, and M. Riedel. Parallel computation of component trees on distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 29(11):2582–2598, Nov 2018.

[8] J. Kazemier, G. Ouzounis, and M. Wilkinson. Connected morphological attribute filters on distributed memory parallel machines. In J. Angulo, S. Velasco-Forero, and F. Meyer, editors, *Mathematical Morphology and Its Applications to Signal and Image Processing*, Image Processing, Computer Vision, Pattern Recognition, and Graphics, pages 357–368. Springer International Publishing AG, 2017.

[9] U. Moschini, A. Meijster, and M. Wilkinson. A hybrid shared-memory parallel max-tree algorithm for extreme dynamic-range images. *Ieee transactions on pattern analysis and machine intelligence*, 40(3):513–526, 3 2018.

[10] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 15:3531–9, 12 2006.

[11] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4):555–570, April 1998.

[12] H. Samet and M. Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):579–586, July 1988.

[13] P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, Berlin, Heidelberg, 2 edition, 2003.

[14] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, Apr. 1975.

[15] M. Wilkinson, H. Gao, W. Hesselink, J.-E. Jonker, and A. Meijster. Concurrent computation of attribute filters on shared memory parallel machines. *Ieee transactions on pattern analysis and machine intelligence*, 30(10):1800–1813, 10 2008. Relation: https://www.rug.nl/informatica/organisatie/overorganisatie/iwi Rights: University of Groningen. Research Institute for Mathematics and Computing Science (IWI).

[16] M. H. F. Wilkinson. A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In *2011 18th IEEE International Conference on Image Processing*, pages 1021–1024, Sep. 2011.

university of groningen

faculty of science and engineering

computing science