university of groningen

faculty of mathematics and natural sciences

computing science

SC@RUG 2014 proceedings

# 11th SC@RUG 2013-2014

Rein Smedinga, Michael Biehl and Femke Kramer (editors)

www.rug.nl/research/jbi

# SC@RUG 2014 proceedings

Rein Smedinga
Michael Biehl
Femke Kramer
editors

2014
Groningen

ISBN: 978-90-367-6316-5
Publisher: Bibliotheek der R.U.
Title: SC@RUG 2014 proceedings
Computing Science, University of Groningen
NUR-code: 980

# About SC@RUG 2014

## 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 eleventh time in the academic year 2013-2014. Students wrote a paper, participated in the review process, gave a presentation and chaired a session during the conference.

The organizers Rein Smedinga, Femke Kramer and Michael Biehl would like to thank all colleagues who cooperated in this SC@RUG by collecting sets of papers to be used by the students and by being an expert reviewer during the review process. They also would like to thank Agnes Engbersen for her very inspiring workshops on presentation techniques and speech skills.

## Organizational matters

SC@RUG 2014 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 this subject including the different approaches in the given papers. The paper should compare the theory in each of the papers in the set and include their own conclusions about the subject. Of course, own research was encouraged.
Two teams proposed their own subject.

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 of the paper through *Nestor*.

All papers could be rewritten and resubmitted, independent of the conclusions from the review. 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 can be found in these proceedings.

In her lectures about communication in science, Femke Kramer explained how researchers communicate their findings during conferences by delivering a compelling storyline supported with cleverly designed images. She also taught workshops on writing a scientific paper and on reviewing such a paper.

In another workshop, Michael Biehl showed how researchers review each other's papers.

Agnes Engbersen gave workshops on presentation techniques and speech skills that were very well appreciated by the participants. She used the 2 minute madness presentation as a starting point for improvements.

Rein Smedinga was the overall coordinator, took care of the administration and served as the main manager of *Nestor*.

Students were asked to give a 2-minute 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.

The conference itself was organized by the students themselves. In fact half of the group was asked to fully organize the day (i.e., prepare the time tables, invite people, look for sponsoring and a keynote speaker, etc.). The other half acted as a chair and discussion leader during one of the presentations. We had dual presentations for each paper. The audience graded both the presentation and the chairing and leading the discussion.

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

Students were graded on the writing process, the review process and on the presentation. Writing and rewriting counted for 35% (here we used the grades given by the reviewers and the re-reviewers), the review process itself for 15% and the presentation for 50% (including 10% for being a chair or discussion leader during the conference and another 10% for the 2 minute madness presentation). For the grading of the presentations we used the assessments from the audience and calculated the average of these.

In this edition of SC@RUG students were videotaped during their 2 minute madness presentation and during the conference itself using the new video recording facilities of the University and with thanks to the CIT crew (special thanks to Adri Mathlener for providing and operating a mobile recording kit during the conference). The recordings were published on *Nestor* for self reflection.

On 9 April 2014, the actual conference took place. Each paper was presented by both authors. We had a total of eleven presentations this day.

## Sponsoring

The student organizers arranged two keynote speakers this time and both the corresponding companies sponsored the event as well by providing lunch and drinks afterwards and payed for the additional costs like programme leaflets and such. We very much thank:

- Matthijs Vogt from ilionX. ilionX is a medium-sized IT company that implements solutions to customers in several fields like business intelligence, cloud solutions and consultancy. Matthijs Vogt is lead business intelligence consultant at Information Management ilionX north.
- Gert-Jan van Dijk from Targeet Holding. Target Holding is, as a partner in the Target Project, responsible of valorization of knowledge and offers solutions for storage, analysis, processing, archiving and searching in the area of large-scale intelligence. Gert-Jan van Dijk is the CEO of Target Holding.

**Thanks**

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

- André Sobieck
- Doina Bucur
- Frank Blaauw
- Apostolis Ampatzoglou
- Zengyang Li
- Michael Wilkinson
- Jasper van der Gonde
- Henk Bekker
- Alexander Lazovik
- Faris Nizamic

and all other staff members who provided sets of papers but were not needed in the review process.

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 this conference,
- *Target Holding* and *ilionX* for sponsoring lunch, drinks and providing a keynote speaker and
- *Agnes Engbersen* for providing excellent workshops on improving presentation skills.
- *Adri Mathlener* for providing and operating the mobile video recording kit during the conference

Rein Smedinga
Femke Kramer
Michael Biehl

Since the tenth SC@RUG last year we added a new element: the awards for best presentation, best paper and best 2 minute madness. Therefore, from last year's edition on, we will have a Hall of Fame:

**Best 2 minute madness presentation awards**

**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**

**2014**
Diederik Lemkes and Laurence de Jong:
*Pyschopathology network analysis*

**2013**
Jelle Nauta and Sander Feringa,
*Image Inpainting*

**Best paper awards**

**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 Review of Seamless Image Cloning Techniques

Lukas de Boer, Jan Veldthuis

**Abstract**— Image editing tasks concern either global changes (color/intensity corrections, filters, deformations) or local changes confined to a selection of the image. In this paper we are interested in achieving local changes that are restricted to a manually selected region. These changes range from removing slight distortions in images to replacing content in an image by novel content. Classic tools are available that achieve interactive cut-and-paste with cloning tools that are used for complete replacements of content, and image filters for slight changes. An example of this is the cloning stamp in Adobe Photoshop. However, these tools result in visible seams and distortions, which can only be partly hidden by feathering along the contour of the local selection. We have compared three alternative techniques to these classic tools, from which different tools for seamless editing and cloning can be derived: *a*) solving a linear system of poisson equations with Dirichlet boundary conditions; *b*) Mean-Value Coordinates to interpolate pixels along the boundary; and *c*) image inpainting to repair a damaged region of an image. We have described and compared these approaches based on applicability, ease of use, speed and quality of resulting images, using implementations readily available on the internet. Furthermore we have created a fixed set of images to facilitate the comparison of the different methods, highlighting the advantages and disadvantages of each method. We conclude that MVC cloning approaches the quality of Poisson cloning for only a fraction of the cost, and that image inpainting has limited usability compared to the other two approaches.

**Index Terms**—Interactive image editing, seamless cloning, Poisson equation, mean-value coordinates, image inpainting, stitching, scene transform, color transfer.

## 1 Introduction

Image cloning has many practical applications. Often a photograph needs to be manually edited for use in advertisements or other print. Besides editing of the color levels of an image, sometimes parts of an image need to be removed or added. In such a case, an artist would use a tool from a software package, for example the Cloning or Duplication tool in Adobe Photoshop. It can take the artist a considerable amount of time to remove or add an object without causing major visual artifacts such as seams.

An alternative to the manual approach is to use gradient domain image cloning. For such techniques, in general, a region is selected manually in a source image and placed manually onto a clone region in the target image. Next, a linear system like the Poisson partial differential equation has to be solved. One example of Poisson cloning, described by Pérez at al. [13], involves solving a Laplace equation, which is a Poisson equation with zero as the right hand side of the equation, with Dirichlet boundary conditions, which describes the values that the solution for the differential equation need to take at the boundary of the domain. The gradient inside the cloning region is taken from the target image while the Dirichlet boundary conditions come from the boundary of the cloning region with the target image. Poisson cloning then uses the gradient as a guidance field to smoothly interpolate the discrepancies along the boundary of the cloned region. This approach involves solving a large linear system which, whilst creating visually attractive results, makes it very slow for practical usage scenarios.

A second technique is described by Farbman et al. [6]. Their technique avoids solving a linear system and instead directly constructs a smooth interpolating membrane using Mean-Value Coordinates (MVC). This membrane is not identical to the membrane used for Poisson cloning, but produces visually indistinguishable results. Farbman et al. state that the advantages of MVC cloning as compared to Poisson cloning are that it is easier to implement, has a lower memory footprint and is faster and highly parallelizable.

A third and final technique called image inpainting by H. Li et al. [11] is described. This approach combines different image editing techniques in order to reconstruct a damaged image using an image database containing information that can be used to replace damaged regions of images. After the new information is cloned into the damaged image (scene transform), a color transfer function is applied in order to make the result more adapted to visual expectations. Furthermore, F. Li et al. [10] describe an image inpainting algorithm using Chambolles dual method that can even recover color data from an image with only some known color.

There are many image editing methods [2] [8] [14], where each one has its own restrictions and conditions, and in this paper we are going to discuss and compare three techniques for local image editing. For our research we used available implementations from the internet, however we did not manage to find an implementation of image inpainting. Therefore, our main comparison will be Poisson cloning vs. MVC cloning. We will discuss the advantages and disadvantages of the these two techniques and show the types of visual artifacts that can occur. Finally, we will compare these two gradient based cloning techniques with image inpainting on a theoretical level.

The remainder of this paper is organized as follows: Section 2 will describe the methods introduced in more detail. Section 3 describes the method of comparison we used in this paper. Section 4 displays the results of the methods described using a consistent set of images. Section 5 concludes this paper, and Section 6 shows a proposal for future work.

## 2 Descriptions of Methods

In this section we will give an outline of the theory and equations behind the gradient domain blending techniques, Poisson cloning and MVC cloning. We will also briefly discuss image inpainting and how it relates to gradient blending.

### 2.1 Poisson equations with Dirichlet boundary conditions

The Poisson equation has been used extensively in computer vision. It originates from fields like electrostatics, and is commonly used to model physical problems such as diffusion. At the core of this approach lies the Poisson partial differential equation with Dirichlet boundary conditions, which specifies the Laplacian of an unknown function over the domain of interest, along with the unknown function values over the boundary of the domain. The motivation for this choice is due to two reasons; psychologists have long known (Land [9], Palmer [12]) that the overlaying of slow gradients of intensity, which are suppressed by the Laplacian operator, can be done on an image without the human mind noticing a distinct difference. However, the second-order variations extracted by the Laplacian operator are a lot more significant perceptually. Secondly, an unknown

- *Lukas de Boer is a Master Student Computing Science at the RuG, e-mail: lukas@luqq.nl.*
- *Jan Veldthuis is a Master Student Computing Science at the RuG, e-mail: jan.veldthuis@gmail.com.*

scalar function on a bounded domain can be defined uniquely by two properties: the values on the boundary of the domain and its Laplacian in the interior of the domain. Therefore, the Poisson equation has a unique solution and this leads to a sound algorithm.

So, given methods for crafting the Laplacian of an unknown scalar function over some domain, and its boundary conditions, the Poisson equation can be solved numerically to achieve seamless filling of that domain. This can be replicated indepently in each of the channels of a color image. Solving the Poisson equation also has an alternative interpretation as a minimization problem: it computes the function whose gradient is the closest, in the $L_2$-norm, to some prescribed vector field - the *guidance* vector field - under given boundary conditions. In that way, the reconstructed function interpolates the boundary conditions inwards, while following the spatial variations of the guidance field as closely as possible.

### 2.1.1 Poisson Equation

Poisson cloning uses interpolants using a guidance vector field. Only scalar image functions are considered, since it is possible to solve the interpolant for all image color channels. Let S, a closed subset of $\mathbb{R}^2$, be the image definition domain, and let $\Omega$ be a closed subset of S with boundary $\partial\Omega$. Let $f^*$ be a known scalar function defined over S minus the interior of $\Omega$ and let $f$ be an unknown scalar function defined over the interior of $\Omega$. Finally, let $v$ be a vector field defined over $\Omega$.

The simples interpolant $f$ of $f^*$ over $\Omega$ is the membrane interpolant defined as the solution of the minimization problem:

$$\min_f \int\int_\Omega |\nabla f|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{1}$$

where $\nabla . = [\frac{\partial .}{\partial x}, \frac{\partial .}{\partial y}]$ is the gradient operator. The minimizer must satisfy the associated Euler-Lagrange equation

$$\Delta f = 0 \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{2}$$

where $\Delta . = [\frac{\partial^2 .}{\partial x^2}, \frac{\partial^2 .}{\partial y^2}]$ is the Laplacian operator. Equation 2 is a Laplace equation with Dirichlet boundary conditions.

## 2.2 A coordinate based approach using Mean-Value Coordinates

The Mean-Value Coordinates method, as described by Farbman et al. [6], introduces a coordinate-based approach that performs seamless cloning, as well as a number of other related operations in a direct manner, without ever having to form and solve systems of equations. In comparison with the aforementioned Poisson cloning, this approach is fast, straightforward to implement, and features a small memory footprint. Additionally, a large portion of the computation can be performed in parallel on, for example, the GPU to further increase computational performance. With the Poisson cloning method, the Poisson equation is solved, whereby the source patch determines the gradients inside the cloned region, and the boundary of the cloned region with the target image determines the Dirichlet boundary conditions.

Pérez at al. [13] described solving the Poisson equation as equivalent to solving the Laplace equation with the Dirichlet boundary conditions. One could say that Poisson cloning constructs a "membrane", specifically a harmonic interpolant, that smoothly spreads the discrepancies along the boundary between the source patch and the target image over the entire interior of the cloned region. The idea of using Mean-Value Coordinates is that, to avoid having to solve a large linear system, a different smooth interpolating membrane is created directly. This membrane does not have to be exactly equivalent to the membrane constructed by Poisson cloning, but the key is that the membrane is similar enough to provide results that are indistinguishable from Poisson cloning.

Furthermore, Farbman et al. [6] observed that it is not necessary to evaluate the membrane at every pixel inside the cloned region. After all, the most discrepancies in the membrane are present along the boundary, while the membrane away from the boundary is typically very smooth. In their approach, Farbman et al. construct an adaptive mesh and only only evaluate the membrane at the vertices of that mesh. The values at the other pixels can then be calculated by using linear interpolation which can be done very quickly on the GPU. A similar optimization was utilized by Agarwala [1] to solve large Poisson systems, such at those arising in gradient domain stitching, with a small memory footprint. Another important optimization introduced by Farbman et al. is the use of adaptive hierarchical sampling of the boundary.

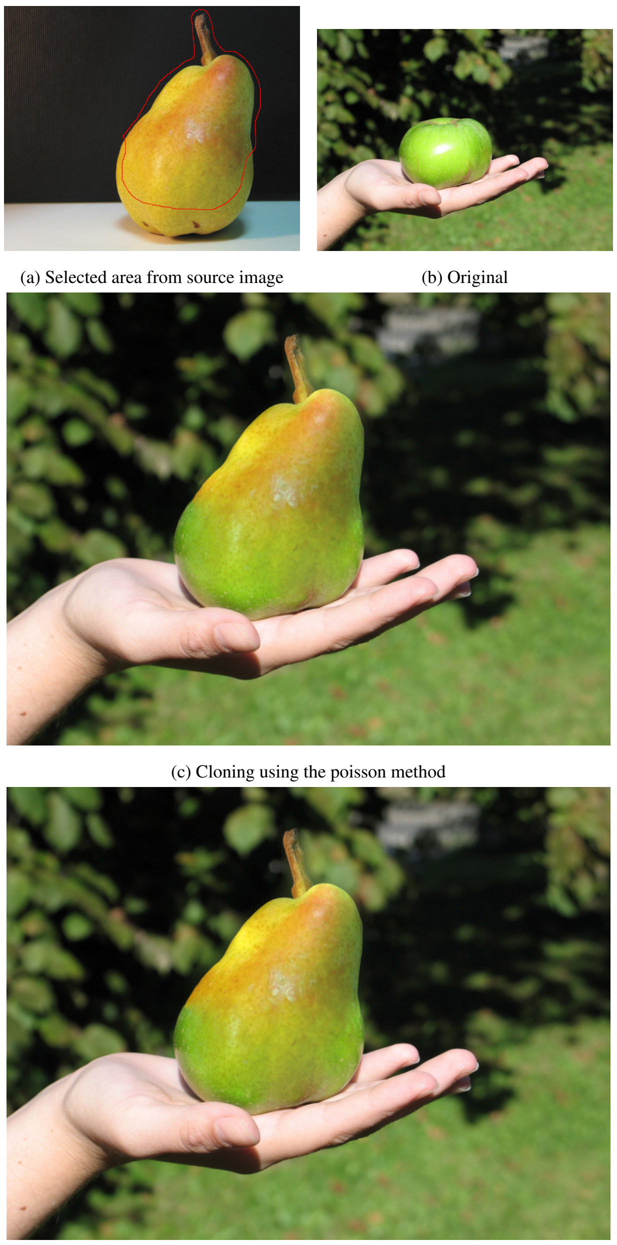
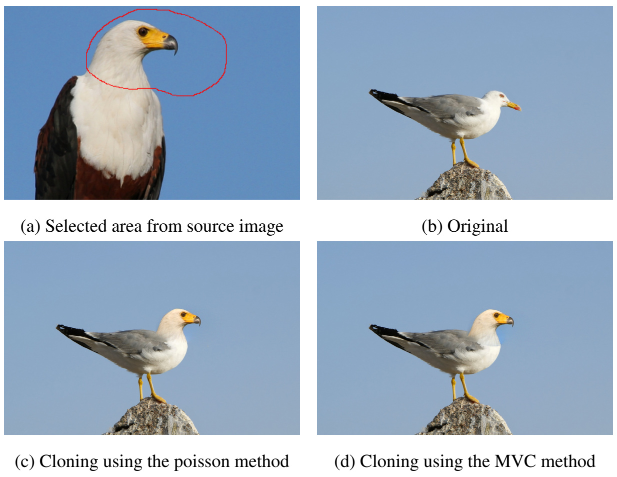

(a) Selected area from source image
(b) Original
(c) Cloning using the poisson method
(d) Cloning using the MVC method

Fig. 1: Changing an apple into a pear by cloning the top along with part of the texture. Source image courtesy of Carlos Paes, `http://www.rgbstock.com/gallery/wax115`. Target image courtesy of Sanja Gjenero, `http://www.rgbstock.com/gallery/lusi`.



(a) Selected area from source image (b) Original

(c) Cloning using the poisson method (d) Cloning using the MVC method

Fig. 2: Splicing the head of an eagle onto a seagull. Source image courtesy of Sias van Schalkwyk, `http://www.rgbstock.com/gallery/Seepsteen`. Target image courtesy of Javier Gonzalez, `http://www.rgbstock.com/gallery/Abyla`

#### 2.2.1 Mean-Value Coordinates

Floater [7] introduced Mean-Value Coordinates which are motivated by the Mean-Value Theorem for harmonic functions. These coordinates are well-defined over the entire plane for arbitrary smooth planar polygons without self-intersections. The use of MVC coordinates for this approach is novel and computationally attractive, as an alternative to solving the Poisson equation in certain image editing tasks.

Consider a closed 2D polygonal boundary curve, with counter-clockwise ordering, $\partial P = (p_0, p_1, ..., p_m = p_0), p_i \in \mathbb{R}^2$. The mean-value coordinates of a point $x \in \mathbb{R}^2$ with respect to $\partial P$ are given by

$$\lambda_i(x) = \frac{w_i}{\sum_{j=0}^{m-1} w_j}, i = 0, ..., m-1, \tag{3}$$

where

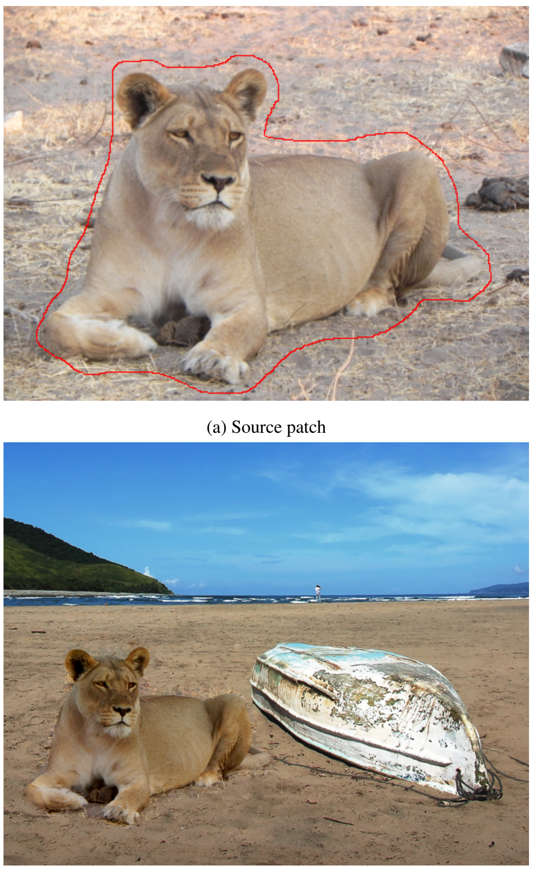
$$w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{||p_i - x||} \tag{4}$$

and $\alpha_i$ is the angle $\angle p_i, x, p_{i+1}$. Once computed, these coordinates may be used to smoothly interpolate any function $f$ defined at the boundary vertices.

### 2.3 Image inpainting based on scene transform and color transfer

Image inpainting techniques aim to fill in the damaged regions of images with new information in a way such that it is hard to find the image has once been damaged. Most image inpainting approaches reconstruct the damaged image using information contained in the damaged image, which may cause structure distortion and erroneous matching blocks if multi-class objects are covered in the damaged area. Examples of such algorithms are presented in Criminisi 2003 [3], Drori 2003 [4], Wilczkowiak 2005 [16], and have many limitations. H. Li et al. [11] state that therefore, given a damaged image, methods of repairing damaged areas in images using information from other pictures will become a new direction of the field of image inpainting.

Color transfer is a growing problem in the field of digital image processing. The process can be described as creating a new image from two different images, using color information from one image, and shape information from the other image. We call the first image the color image and the second image the shape image. The algorithm presented by H. Li et al. is called ”stct-inpainting”, which means *scene transform color transfer inpainting*, where the introduction of the color transfer algorithm can make the inpainting results adaptable to the visual expectation of the human mind. The ”stct-inpainting” technique is described as follows:



(a) Source patch

(b) Result using Poisson cloning

Fig. 3: Cloning a lioness onto a beach. Source image courtesy of Stella Bogdanic, `http://www.rgbstock.com/gallery/stellab`. Target image courtesy of Ariel da Silva Parreira, `http://www.rgbstock.com/gallery/arinas74`.

1. Extract the texture, color and structure information of damaged images from the shape image, and find the most similar style of image in the image database.
2. Define the range of 40 pixels near a damaged line as a transition band, the total size of the damaged region and the transition band as a mask, choose the proper scene in the source image and adjust the sideline using a cost function.
3. Apply the color transfer algorithm based on clustering to deal with the result of the last step, making the inpainted region in accordance with its surroundings.

## 3 Methods

For our research we did not implement Poisson cloning, MVC cloning or image inpainting, but used readily available implementations. For Poisson cloning we used the Poisson Image Editing software by C. Tralie [15] and for MVC cloning we used an implementation by J. Elinson [5]. As stated before, unfortunately we could not locate an implementation of image inpainting.

We will not discuss any usability difference between these applications that originate from the implementation, for example the user interface, however we will compare algorithmic differences like speed and accuracy. Because the Poisson cloning and MVC cloning are different applications, and both applications require manual selection of the source patch using the mouse, we will not be able to use exactly the same source patch selection for both resulting image, however the difference will be kept to a minimum by making the same rough selection shape.

In order to compare the two gradient based cloning methods we will use various images that illustrate situations where one or both algorithms work well or were they fail to create a convincing image. A convincing image is defined as having no obvious graphical artifacts that would make a viewer doubt the legitimacy of the image; only a close look should reveal it as being a splice of two images.

Cloning has multiple use cases: *a*) adding a complete object to the target image from a source image; *b*) replacing part of an object in the target image with part of an object in the source image; *c*) replacing a texture in the target image with a texture from the source image. We will illustrate examples of these use case. We will also compare the speed efficiencies of the two algorithms.

Finally we will compare the gradient based cloning methods image inpainting. Image inpainting has a more specific use than the gradient based cloning methods. With image inpainting, part of an image is deemed "damaged" and these areas will be repaired using textures from other images. We will discuss a theoretical approach in which gradient based cloning methods can be used to get similar results as image inpainting, and we will discuss the difference between these approaches.

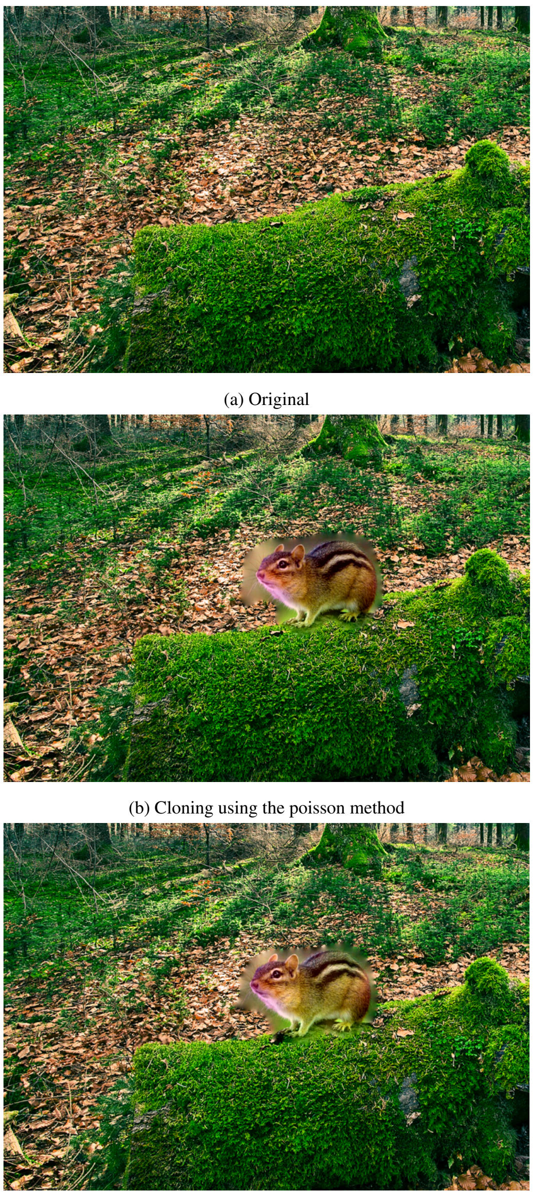
## 4 Results

### 4.1 Cloning objects

Using either technique, it is easy to clone entire objects from a source image into a target image. Some care has to be taken to select appropriate source and target images, which will be discussed later, but in general the results are more convincing than a naive cut and paste of a selection. Even the hue, brightness and saturation from the source image is automatically correctly modified to fit in the target image.

In Figure 3 a lioness has been cloned onto a beach. Some visual artifacts can still be observed around the boundary of the lioness, but on first inspection the result looks very convincing. Furthermore, the amount of time required from the user to create such a composition is practically none; the user simply needs to roughly cut out the lioness and place the source patch at the desired location. In comparison a professional cloning job would have resulted in less visible artifacts, but would also have taken more time to perform.

In Figure 1 part of a pear is merged with part of an apple. The background of the apple image is out of focus, but still detailed. Both MVC and Poisson cloning manage to smoothly transition from the apple to the pear. However, the dark part of the stem of the pear is blended so that it appears translucent in the resulting images. Another visual artifact is the striped background of the pear source image. In the resulting images, these stripes are still slightly visible, however this is simply a matter of choosing better source images. In comparison, a similar clone was done in Figure 2, where the head of an eagle is cloned on the body of a gull, but the background is mostly a single shade of blue. With a smooth background these types of visual artifacts are not visible.


(a) Original

(b) Cloning using the poisson method

(c) Cloning using the MVC method

Fig. 4: Placing a chipmunk on top of a mossy tree trunk. Source image courtesy of Ken, `http://www.rgbstock.com/gallery/wildarts`. Target image courtesy of Andreas Krappweis, `http://www.rgbstock.com/gallery/krappweis`.

### 4.2 Cloning textures

Both Poisson cloning and MVC cloning are good at cloning detail from the source image to the target image. This does not only mean that an object can be cloned from the source to the target, but a texture on the target image can be replaced with a texture from the source image too. An example of this is shown in Figure 5, where a brick wall texture is cloned from the source image onto the wall of the target image. A similar case can be seen in Figure 1, where part of the pear texture is copied over the apple's surface.

### 4.3 Background texture (mis)match

In general, both techniques work best when the background of both the source image and the target image are similar. For example, the best results can be gotten when either the background in both images is smooth, for example the sky, water or if the background is out of focus, or when the background in both images has the same type of detail, for example if the background in both images is grass, wood, etc. Obviously, when the backgrounds of the source and target image are similar the cloning is more successful. However, both Poisson cloning and MVC cloning produce inadequate results when the background of the source and target images do not match.

The first case is when the background of the target image is detailed, while the background of the source image is blurry. In such a case, there is a perceptible smudge around the cloned area, due to the cloning technique attempting to smooth the boundary of the cloning selection and the target image into the cloned source. This approach does not work because the background is not smooth itself: as a result detail from the background behind the cloned region is lost and this lack of detail is very noticable. An example of this can be seen in Figure 4. Here a chipmunk is cloned onto a mossy tree trunk, however the tree truck and the leaves on the ground behind it are highly detailed,

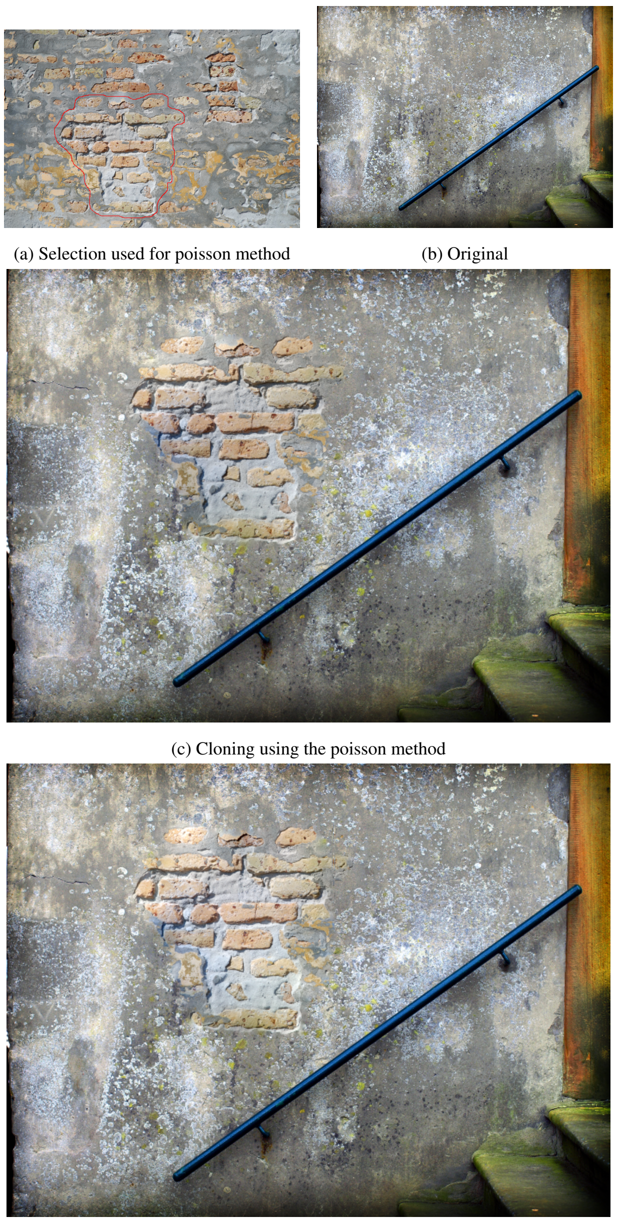

(a) Selection used for poisson method
(b) Original
(c) Cloning using the poisson method
(d) Cloning using the MVC method

Fig. 5: Placing a brick texture onto a wall. Source image courtesy of Lars Sundström, `http://www.rgbstock.com/gallery/sundstrom`. Target image courtesy of Jay Simmons, `http://www.rgbstock.com/gallery/jazza`.

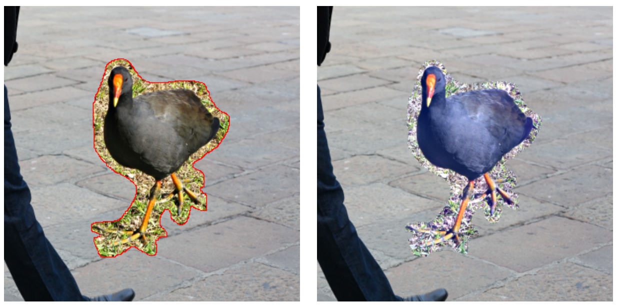

(a) Target blending region
(b) Result after blending using poisson method

Fig. 6: Cloning a moorhen from a source image with a detailed background onto a street with a relatively smooth surface. Source image courtesy of Adrian van Leen, `http://www.rgbstock.com/gallery/TACLUDA`. Target image courtesy of Mei Teng Wong, `http://www.rgbstock.com/gallery/MeiTeng`.

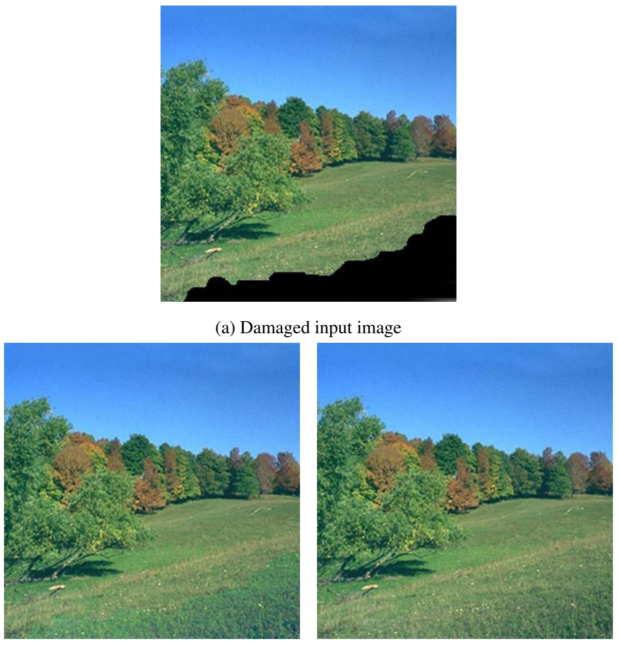

(a) Damaged input image
(b) Result after using coarse filling in the image in (a)
(c) Result after applying color transfer to the image in (b)

Fig. 7: The result of applying the image inpainting algorithm to a damaged image. Image source: H. Li et al. [11].

causing the area around the chipmunk to become blurred.

The second case is when the background of the target image is blurry, while the background of the source image is detailed. In this case, the detail from the source image is copied onto the background of the target image. For example, if the source image is an object on grass, a detailed background, and the object is cloned onto a stone surface in the target image, which is a relatively smooth background, then the stone in the area around the object will get a grassy texture. The resulting image does not look natural. This situation is shown in Figure 6.

The paper by Pérez et al. described a technique called mixed gradients, where part of the gradient of the target image is taken into account to solve this problem. The paper by Farbman at al. describes a technique with matting, where MVC is used to obtain a monochrome matte of the source image that describes the parts of the image that are foreground and background, white and black respectively. This matte is then used as the alpha value for the source patch to avoid cloning background pixels. Neither of the applications we used for our research implemented these techniques, and thus we were not able to compare these results.

## 4.4 Comparison with image inpainting

Image inpainting is different from the gradient based techniques, however cloning can be used to try and get the same results. Image inpainting replaces damaged areas with textures from other images. An example of the process is shown in Figure 7. Poisson and MVC cloning could also be used to place textures from other images onto the damaged areas. However there are a number of disadvantages compared to image inpainting.

The first difference is that image inpainting automatically repairs the damaged regions, while the other two approaches require manually selecting fitting source pathes. This gives image inpainting an advantage similar to the advantage that the other two approaches have over manually blending the cloning patch. Manual work takes more time and effort than automatic work.

The second difference is that when repairing a damaged region via blending, part of the boundary of the cloning patch is the damaged region. This means that the damaged region interferes with the boundary constraints of the blending. For example, if a red letter is removed from an image the user has to take care that the red is not part of the boundary, or else the red part of the letter will be blended with the cloned area.

## 5 Conclusion

The resulting guidance field used by Poisson cloning and MVC cloning differ quite a lot, however visually it is difficult to say which method is better. MVC attempts to approximate the guidance field used by Poisson cloning in order to achieve the same visual results while being computationally faster. It certainly succeeds at this: the qualitative difference between the resulting images from MVC and Poisson cloning are not significant but the speed of MVC is orders of magnitude larger. The paper by Farbman et al. [6] describes their algorithm being fast enough to show the user what the result of cloning would be in real-time. The implementation of MVC that we used did not achieve this, but it would give MVC cloning an even greater advantage over Poisson cloning.

In our paper we have used a set of images to compare the results of MVC cloning and Poisson cloning. From the various resulting images it can be seen that while there are slight differences, it is not easy to say which image is preferrable to the other. With this set of images we have also tested the restrictions on these two methods: both methods fail to produce seamless images when the texture of the background differs significantly between the source and target images.

Image inpainting is a different approach to this problem which results in smooth seamless pictures, but it has the necessity of an image database as a drawback. Also, if there is no image in the database that has similar properties to the damaged region, stct-inpainting will not be able to reconstruct a smooth image.

## 6 Future Work

There is drastic need of an open-source application that implements the techniques we discussed, possible alongside other techniques. It would allow the user to easily make the same selection from the source image and try out the various cloning methods to select the best one for the situation without having to switch between applications. It would also solve the lack of a readily available, tried-and-tested implementation of the three techniques we discussed.

Another point of improvement may be an objective evaluation function of cloning results, so objective results can be obtained to compare the different algorithms.

## References

[1] A. Agarwala. Efficient gradient-domain compositing using quadtrees. *ACM Transactions on Graphics*, 26(3):94, July 2007.

[2] P. J. Burt and E. H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2(4):217–236, Oct. 1983.

[3] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–721–II–728 vol.2, June 2003.

[4] I. Drori, D. Cohen-Or, and H. Yeshurun. Fragment-based image completion. *ACM Trans. Graph.*, 22(3):303–312, July 2003.

[5] J. Elinson. `https://github.com/jelinson/MVC/`.

[6] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, and D. Lischinski. Coordinates for instance image cloning. In *SIGGRAPH 09*, Aug. 2009.

[7] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, Mar. 2003.

[8] S. Jeschke, D. Cline, and P. Wonka. A gpu laplacian solver for diffusion curves and poisson image editing. *ACM Trans. Graph.*, 28(5):116:1–116:8, Dec. 2009.

[9] E. H. Land and J. J. McCann. Lightness and retinex theory. *Journal of the Optical Society of America*, 61(1):1–11, Jan. 1971.

[10] F. Li, Z. Bao, R. Liu, and G. Zhang. Fast image inpainting and colorization by chambolles dual method. *Journal of Visual Communication and Image Representation*, 22(6):529 – 542, 2011.

[11] H. Li, S. Wang, W. Zhang, and M. Wu. Image inpainting based on scene transform and color transfer. *Pattern Recognition Letters*, 31(7):582–592, May 2010.

[12] S. E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, May 1999.

[13] P. Pérez, M. Gaangnet, and A. Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, July 2003.

[14] A. Sobiecki, A. Telea, G. A. Giraldi, L. A. P. Neves, and C. E. Thomaz. Low-cost automatic inpainting for artifact suppression in facial images. In *VISAPP (1)*, pages 41–50, 2013.

[15] C. Tralie. `https://github.com/ctralie/PoissonImageEditing/`.

[16] M. Wilczkowiak, G. J. Brostow, B. Tordoff, and R. Cipolla. Hole filling through photomontage. In *16th British Machine Vision Conference 2005 - BMVC'2005, Oxford, United Kingdom*, pages 492–501, July 2005.

# Android Applications — Evaluating Tools to Secure the Android Market

Hessel B. van Apeldoorn & Mark Hoekstra

**Abstract**—An Android application is given access to resources based on what its implementers have specified. Users then have to allow access to these resources when they want to install an app on their mobile device. Often more permissions are granted than strictly necessary for the functionality of a certain application. Even apps not requesting more permissions than needed, are potentially abusing these granted permissions. Several tools have been developed to analyse these permissions granted to Android applications. These tools check if an application is granted permissions it does not need or abuses the granted permissions.
Our research verifies which of these tools is the most useful in applying a security check to Android apps before these apps appear on the Android market. When an app is not accepted by one of these tools, it should be marked as untrustworthy, otherwise it is allowed to be published.
For some of the tools developed to analyse apps, we check if they provide a valuable contribution to secure the Android market. Different tools have different approaches such as looking at applications' permissions or the usage of these permissions for malicious ends. Our research evaluates these tools with regard to this subject. We have concluded that all evaluated tools provide a useful contribution to the security of the Android markets. Furthermore, we elaborate on the benefits of reducing permissions and malware for the privacy of Android users.

**Index Terms**—Android, security, privacy, permissions, applications, malware, Android markets.

## 1 Introduction

Over the past few years, the number of people using Android devices has increased rapidly. To illustrate; around 310 million devices were sold in 2012 that were running on the Android operating system. This number is expected to increase to 540 million in 2015. The total number of sold open OS devices is expected to be 1.1 billion, meaning that Android holds about 50% of this market [1].

Software products for the Android platform are called applications (abbreviated as apps) and can be installed from the Android market. With Android market we refer to all the existing Android markets. The Android OS allows third parties to develop apps. The number of third parties developing software for Android has risen, totalling 700,000 apps in Q1 2013 and 1.1 million apps in Q1 2014 [2]. Third party developers create applications using the Android API. This API provides a set of functions that allows applications to access the phones hardware (e.g., the GPS) or certain phone settings and information (e.g., contacts, messages). Several API calls can only be used when the right permissions have been granted. A permission is declared in the app by the developer [3]. Before installing third party software, users must accept these permissions (e.g., accessing phone status, internet). Figure 1 shows an overview of some permissions and the API calls they invoke. From this figure it can be seen that the source code and documentation is used to determine sources (start of a dataflow) and sinks (end of a dataflow) of data. Currently Android requests the user to accept all the permissions and it is not clear what the permissions are used for.

Similar to every operating system, malware has been spread on Android. Malware is malicious software that negatively influences an operating system [5]. Malware can leak information to advertisers, negatively affect the user experience and abuse functionalities of the infected device. There are many more forms of malware, but the previously named possibilities occur the most often on Android.

Android's increasing share in the market and the increasing number of malware applications for Android go hand in hand. The Cisco 2014 Annual Security Report states that 99 percent of mobile malware targeted the Android platform in 2013 [6]. A research has been done on dissecting malware resulting in identifying 1260 malware samples in 2012 [7]. From these 1260 samples it was discovered that 86% of them consist of altered versions of legitimate software by adding malicious code.

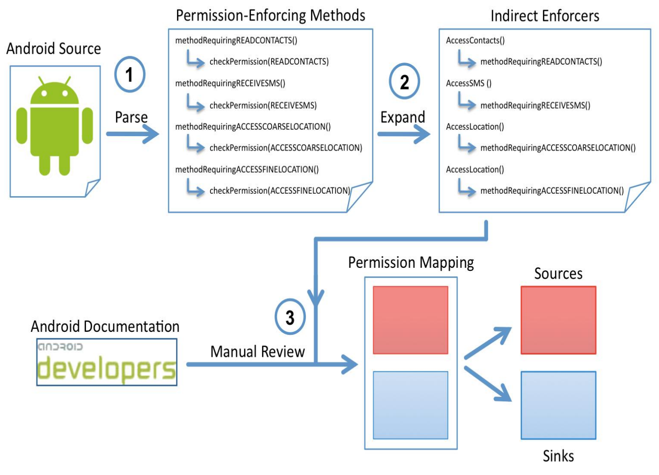

Android Source
1
Parse
Permission-Enforcing Methods
methodRequiringREADCONTACTS()
checkPermission(READCONTACTS)
methodRequiringRECEIVESMS()
checkPermission(RECEIVESMS)
methodRequiringACCESSCOARSELOCATION()
checkPermission(ACCESSCOARSELOCATION)
methodRequiringACCESSFINELOCATION()
checkPermission(ACCESSFINELOCATION)
2
Expand
Indirect Enforcers
AccessContacts()
methodRequiringREADCONTACTS()
AccessSMS ()
methodRequiringRECEIVESMS()
AccessLocation()
methodRequiringACCESSCOARSELOCATION()
AccessLocation()
methodRequiringACCESSFINELOCATION()
Android Documentation
developers
3
Manual Review
Permission Mapping
Sources
Sinks

Fig. 1. Mapping between API methods and Permissions [4]

Permissions are standing at the root of security and privacy on the Android OS. A permission gives an app the opportunity to insert a virus on our smartphone or to leak our personal information to ad providers. A total of 940 apps have been analysed by Stowaway [8]. One third of these apps were overprivileged. This means that an app uses more permissions than it strictly needs. The opposite of overprivilege is least privilege, which applies to apps requesting the exact number of permissions it uses. An app should always use the least privileges it needs.

As Android is a relatively new and rapidly expanding platform, it still has some problems. One of the biggest issues concerns privacy and security. About 24,000 applications have been analysed where a total of around 57,000 leaks have been found [4], showing that privacy and security are real issues on the Android platform. As such, several tools and papers have been written about the malware and pri-

- *Hessel B. van Apeldoorn is a computing science student at the University of Groningen, E-mail: h.b.van.apeldoorn@rug.nl*
- *Mark Hoekstra is a computing science student at the University of Groningen, E-mail: mark.hoekstra@rug.nl*

vacy leaks on Android. Also, these papers contain solutions on how to resolve these viruses and leaks. In this paper we will identify and analyse a selection of these tools.

Our research focuses on evaluating several tools which are able to detect overprivilege (An app requests more permissions than needed), privacy leaks or even malware. These tools were selected based on 3 quality measures:

- **The author:** If an author is an employee of either Google or a smartphone manufacturer that uses Android, he will have the most recent knowledge. This is important as Android is a young and thus quickly changing field of research.
- **Release of the paper describing the tool:** Conferences which have a reputation of releasing good papers attract the best scientists. In turn this will cause future papers of these conferences to be of good quality.
- **The number of citations:** This is a general metric to measure the quality of a paper. A paper that has been cited very often, is in general a good paper.

The tools are evaluated based on common metrics available in all their papers. In this paper we use the Google Play Store as the reference app store. Manufacturers and third parties have also brought their own markets to Android. Conclusions and arguments of this paper do also apply to these smaller markets. Our evaluation is based on evaluating tools to secure the Android market, which can be used to decrease overprivilege and decrease the number of apps with privacy leaks or malware in the Android market.

The rest of this paper is organised as follows: Section 2 gives an overview of the current security in Android. Section 3 describes the tools evaluated in this paper. Section 4 describes the different approaches used in the tools. Section 5 gives an overview of the advantages and disadvantages of using these tools in the Android market. In Section 6 the results are discussed, followed by a conclusion in Section 7. Finally, Section 8 describes the future work that needs to be done to integrate these tools in the Android market.

## 2 Current Security Android

Currently Android offers a few security features [9]:

- **Sandboxing:** An application runs within its own sandbox. It can not access another application's data. This concept is further illustrated in Figure 2.
- **Permissions:** A developer has to specify in the manifest which resources (access contacts, send SMS, etc.) an application is allowed to access. The user accepts the requests for using these resources upon installing an app. Also, this application cannot access a resource that is not specified in the manifest.
- **Malware removal:** If the Android OS on a device is modified by an app, that app is removed. Furthermore, Google can remotely remove malicious apps.

In February 2012 Google introduced a service called Bouncer [9]. This service checks for malware, spyware and trojans. It is only used in the Google Play Store. Other app markets do not include this service. It is possible to circumvent Bouncer, however [11]. As such, adding tools to secure the Google Play Store should help.

The Google Play Store seems to have better security than other Android markets. As it has a large number of apps and users and a low number of malware (0.1% as of February 2014 [12]). Besides reducing the amount of malware in the Google Play Store we also suggest to use the tools evaluated in this paper to reduce malware in other markets.

In theory, securing all the Android markets should be enough to prevent malicious apps from being installed on android devices. In practice however it is not possible to fully secure all the Android markets. Besides the tools evaluated in this paper, several other applications could be installed which continuously monitor a user's device.

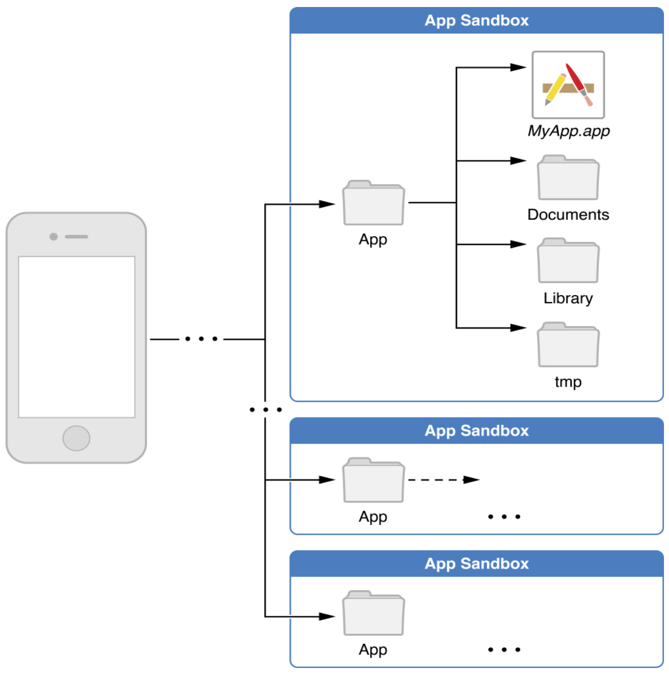


Fig. 2. Illustration of sandboxing in Android [10].

Examples of such applications are Malwarebytes Anti-Malware Mobile [13], which monitors sensitive data during runtime. AVG could be installed on a smartphone to scan for viruses [14] and Bitdefender provides a layer of security as well [15].

## 3 Tools

In this research paper we have analysed several tools. The following sections give a short introduction to each of these tools. The first four tools, TaintDroid, Stowaway, SAAF and AndroidLeaks, are analysing applications before these apps are being published on the Android markets. The last tool discussed, TISSA, is installed on a user's smartphone.

### 3.1 TaintDroid

TaintDroid [16] is used on third-party applications. It tracks when sensitive data leaves the system. The data that has left the system (meaning that it is sent over a network to a third party) is then logged by TaintDroid.

TaintDroid provides system-wide taint tracking. Taint tracking refers to the process of following the flow of a data stream. A data stream is tainted as a first step (adding a flag to the data) to make sure TaintDroid can follow the data stream. It can simultaneously track multiple sources of sensitive data. TaintDroid provides realtime analysis by leveraging Android's virtualised execution environment.

### 3.2 Stowaway

Stowaway [8] detects overprivilege in compiled Android applications by analysing the source code. This source code is extracted from Dalvik Executable (DEX) files using the Dedexer tool [17]. In order to detect overprivilege, Stowaway determines a set of API calls used in an app and then maps those API calls to permissions. The creation of a permission map, gives insight into the used permissions. Comparing an app's used permissions with this app's requested permissions can result in the unused permissions.

### 3.3 Static Android Analysis Framework (SAAF)

SAAF [18] is a framework which analyses smali code, a disassembled version of the DEX format used by Android's Java Virtual Machine. The goal of SAAF is to analyse the data flow in an application by slicing the application. Program slices are useful for detecting malicious code regions in an automated way by tracking a user's personal information (e.g., phone numbers, messages).

### 3.4 AndroidLeaks

AndroidLeaks [4] is a framework for automatically finding potential leaks of sensitive information in Android applications. AndroidLeaks creates a call graph (step 2) of an application's code (obtained by step 1) and analyses whether sensitive information may be sent over the network (step 3) as seen in Figure 3. The creation of such a call graph is done using T. J. Watson Libraries for Analysis (WALA) [19], a program analysis framework for Java source and byte code.

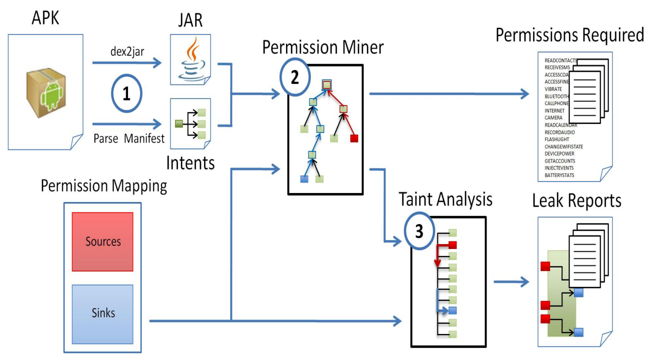


Fig. 3. Architecture of AndroidLeaks [4].

### 3.5 Taming Information-Stealing Smartphone Applications (TISSA)

TISSA [20] is a system which implements a new privacy mode on Android. This new privacy mode allows users to control the kind of personal information accessible to an application. The system also lets users change the granted access during runtime to better suit certain scenarios (e.g., different time or location). The components TISSA adds to Android can be seen in Figure 4 ("Privacy Setting Manager", "Privacy Setting Content Provider", "Privacy Policy Database").

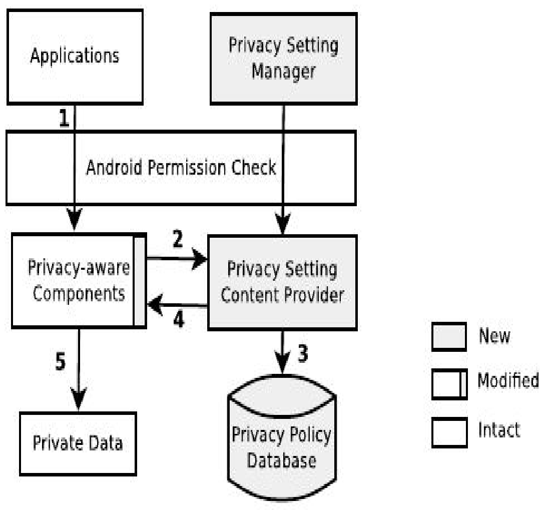


Fig. 4. Architecture of TISSA [20].

## 4 Analysis approaches

Different tools have different approaches to identify overprivilege, malware or reduce an app's access to resources. This section gives an overview of the different approaches used in the tools discussed in this paper.

### 4.1 Static or dynamic analysis

Static and dynamic analysis are two different methods of analysing an application.

Static Static analysis is done on the source code without executing a program [21]. From this code the analysis tool could extract methods which can possible be used in malicious applications. However, further processing must be done to determine the actual use of the method. An example of such a method is a method called "sharePhoneNumber" which is capable of sharing a user's phone number. Stowaway is an example of a static analysis tool, as Stowaway extracts the API calls of an application from the source code.

Dynamic Dynamic analysis, on the other side, is analysing the properties of a running program [22]. Dynamic analysis is useful for tracking the data flow of an application. An example of a piece of such data that could be tracked is a user's phone number which could be shared to third parties. TaintDroid follows the flow of sensitive data and is therefore a dynamic analysis tool.

The static analysis tools evaluated in this research comprise Stowaway, SAAF and AndroidLeaks. These static tools analyse an application's code in order to detect potentially malicious apps. The only dynamic analysis tool included in this research is TaintDroid which analyses the data flow of an application. Lastly, the system called TISSA is neither a dynamic nor a static analysis tool as this system is not analysing an application but is restricting an application's permissions. An overview of the type of tools just described, regarding a static or dynamic approach, is given in Table 1.

| Type / App | Dynamic | Static | OS Redesign |
|---|---|---|---|
| TaintDroid | X | | |
| Stowaway | | X | |
| SAAF | | X | |
| AndroidLeaks | | X | |
| TISSA | | | X |

Table 1. Overview of the type of tools analysed regarding a dynamic, static or another approach

### 4.2 Automatic or dynamic analysis

Besides categorizing a tool in static and dynamic types, a tool can be automatic or manual.

Automatic Automatic tools can operate without human intervention and can analyse a lot of applications relatively fast.

Manual Manual tools need people to analyse an application and to ultimately decide whether or not an app is malicious.

The benefits of automatic over manual tools is that automatic tools save significant amounts of time and money. However, a manual tool is more reliable when deciding whether an app is using permissions for malicious ends or not.

The automatic analysis tools evaluated in this paper comprise TaintDroid, Stowaway and SAAF. AndroidLeaks is a tool which finds malicious apps both automatically and manually. Firstly, AndroidLeaks automatically finds applications which potentially leak sensitive information. Secondly, the apps which potentially leak information are manually checked on leaking information. TISSA is neither automatic nor manual as it is not an analysis tool, as mentioned before. To summarise, the type of tools regarding a manual or automatic approach is presented in Table 2.

| App \ Type | Automatic | Automatic/Manual | User |
|---|---|---|---|
| TaintDroid | X | | |
| Stowaway | X | | |
| SAAF | X | | |
| AndroidLeaks | | X | |
| TISSA | | | X |

Table 2. Overview of the type of tools analysed regarding an automatic, manual or user controlled approach

| Tool | Number of analysed apps |
|---|---|
| TaintDroid | 30 |
| StowAway | 940 |
| SAAF | 142,100 |
| AndroidLeaks | 24,350 |
| TISSA | 24 |

Table 3. Overview of the number of apps used by the analysed tools

All tools have been tested on a number of apps. The total numbers are summed up in Table 3. These numbers deviate quite a lot. SAAF has evaluated a total of 140,000+ apps [18] where TISSA has only evaluated 24 apps [20]. TaintDroid, on the contrary, is tested on 30 popular apps [16] whereas Stowaway evaluated 940 randomly selected apps [8]. AndroidLeaks, tested on 24,350 apps, is the only tool which evaluated apps from different Android markets [4]. Other tools have only considered applications from the Google Play Store.

The difference between the numbers of apps evaluated is mainly caused by the fact that these tools analyse apps differently. SAAF checks apps for malware automatically [18]. The developers of SAAF could thus let SAAF run on multiple apps at the same time without having to conduct actual analysis on each of these apps individually. The developers of TISSA, however, had to analyse each app individually since the tool is operated manually. Furthermore, all apps in TISSA were selected by the researchers manually where the developers of SAAF used a crawler to load all these apps into their tool [18, 20]. Unfortunately, the success rate of these tools can not be determined. Most papers do not show such success rates. Also, the tools analyse apps in different ways. This means there is no possibility to create one uniform success rate.

## 5 Usefulness for the Android market

In this section we evaluate the usefulness of the different tools for the Android market. The usefulness of a tool is based on the following metrics: scalability, architecture and performance. These metrics are selected based on their availability in all papers that describe an evaluated tool.

### 5.1 Scalability

In Table 3 we showed the number of apps that have been used for test purposes for each of the tools. The Google Play store contains 1.1 million apps in Q1 2014 [2]. As such, scalability is an important issue for the tools.

#### 5.1.1 TaintDroid

TaintDroid has been tested on 30 popular applications. This tool is dynamic, meaning it has to be ran on an Android operated device at the same time as that the to be tested application is running. This adds the restriction that the application itself actually has to be executed. Just checking the source code of the application is not enough. This could impose scalability problems upon the Android markets.

#### 5.1.2 Stowaway

Stowaway analysed a set of 940 applications. Contrary to TaintDroid, this tool performs static analysis on applications. Hence scaling this tool to be used on all applications in the markets should not pose a problem in terms of resources. Furthermore, this tool shows for each app whether it is overprivileged or not. This is done automatically, allowing this tool to be used on many apps simultaneously.

#### 5.1.3 SAAF

SAAF is the tool that has the highest number of tested apps, totalling 140,000+. A crawler that searches for apps in the Play Store was used to find all these apps. The high number of tested apps and the use of an automated crawler shows that SAAF is well equipped to handle the scale of the number of apps in Android markets.

#### 5.1.4 AndroidLeaks

AndroidLeaks has been tested on a total of 24,350 apps. This tool is partially automated. It checks for potential data leaks through static analysis. An auditor has to do another check on these leaks to see if they are in fact harmful leaks. This somewhat limits the scalability of this tool. AndroidLeaks should be combined with a tool that performs dynamic analysis to greatly increase scalability [4].

#### 5.1.5 TISSA

TISSA is a tool that has to be controlled by the user instead of being used by a market. As such, it has only been tested on 24 apps. TISSA allows the user to manually set the resources that an app is granted access to. This tool has to be installed on a user's android device before it can be used. This tool is then continuously running in the background. All of the above statements reduce scalability of TISSA.

### 5.2 Aim of analysis

Different tools have different views on how to detect malicious applications. One tool targets an application's privileges while another tool tracks a data flow. We describe the different architectures used in the tools and the advantages and disadvantages of these tools.

#### 5.2.1 Information Flow Analysis (IFA)

TaintDroid, AndroidLeaks and SAAF are analysis tools which are performing data analysis on the data flow in an application [16, 4, 18]. Dynamic IFA has an advantage over static IFA because static IFA has to assume that all code paths in a program can be executed. These assumptions could be wrong as the paths may never be travelled, resulting into false positives. Dynamic IFA's drawback is that it has a significant runtime overhead [23], but it can also be possible that certain parts of the code are not reached while analysing an app.

#### 5.2.2 Overprivilege

Stowaway is detecting applications which request more permissions than used [18]. Detecting overprivileged apps does not directly help with identifying malicious apps. However, making sure apps follow least privilege helps with reducing permission warnings on installation and reducing bugs or vulnerabilities.

#### 5.2.3 Privacy mode

TISSA is a system which lets Android users decide what personal information can be used in an application through implementing a privacy mode [20]. The advantage of this tool above other tools is that the user is in control over the use of personal data in an application, but it cannot be used in an Android market. When TISSA forbids a potentially malicious app's access to certain information, the user can use this app without leaking this information. Although, having the user in control can be considered as an advantage, it can also be a disadvantage. Users have to install and configure the app themselves as this is not done automatically.

### 5.3 Performance

In terms of performance, the tools can be split into 2 groups; the tools that evaluate applications before they are published in the market and the tools that have to be installed on a user's device. TISSA is installed on the user's device, the other 4 tools are executed on pre-published applications.

As TISSA is continuously running in the background, it is important that it poses no significant overhead. According to the developers of TISSA, it does not [20].

The other group, containing the other 4 tools, can be split into 2 groups. Those that perform dynamic analysis (TaintDroid) on an application and those that perform static analysis (Stowaway, AndroidLeaks, SAAF) on an app. Dynamic analysis requires the app in question to be executed, while static analysis only requires the source code of the app in question. This means that dynamic analysis is only as fast as the tested app, where static analysis is only dependent on the available computing power. Static analysis thus has an edge in performance.

## 6 Discussion

The Google Play Store seems to have the best security. As it has a large number of apps and users and a low number of malware (0.1% as of February 2014 [12]). Yet Google has announced new safety measures for this market [24]. The fact that just 0.1% of the malware comes from the Google Play Store implies that other markets are less well secured. These markets would thus benefit even more from utilizing the evaluated tools in their markets.

As TISSA is not used by the markets, but rather by the users, it does obviously not need integration with the markets. Malware is less an issue if most Android owners actually secure their devices with TISSA. The challenge would thus be to convince Android owners to actually use TISSA.

TISSA is the most different from the other tools as it is deployed on a user's electronic device rather than on a market. Since security is still low on the markets, especially considering the large numbers of malware in Android apps, TISSA is a good provisional solution. It provides the user full control over what an app can and cannot do. This is however too time-consuming for the user to be a good final solution. Ultimately, the parties that own the markets should have a sufficiently good filter such that users should not need antivirus software or TISSA-like solutions to protect themselves against malware.

The 4 remaining tools can be split into dynamic- and static analysis tools. Dynamic analysis tools require apps to be executed and thus take up more time to analyze. The owners of the Android markets should however have enough resources to allow for the usage of a dynamic analysis tool. Furthermore, TaintDroid has the advantage that it can analyse apps at runtime, where static analysis tools cannot. TaintDroid would thus be a valuable tool for the Android markets. The requirement to have an auditor check an app after TaintDroid has flagged it as malicious, is still an issue though.

Stowaway, AndroidLeaks and SAAF apply static analysis. Stowaway detects overprivileges where AndroidLeaks and SAAF both check analyse flows. AndroidLeaks and SAAF are therefore the only 2 tools that are not advisable to use together. All the other tools utilize different techniques with different end goals and are, as such, complements instead of substitutes.

## 7 Conclusion

We have analysed 5 different tools to be used on the Android markets. These tools are Stowaway, TaintDroid, AndroidLeaks, SAAF and TISSA. All tools provide additional security and/or privacy for the user of an Android operated device. Although not all of these tools provide it in the same way. As such, a combination of tools could be used on an application before it is published on a market. All tools seem feasible. Stowaway, TISSA and TaintDroid may pose scaling and reliability issues though. Especially considering the low number of apps they have been tested on.

Finally, we consider all tools to be useful for protecting security and privacy on the Android OS. The Google Play Store does already have a decently steady security, considering the low number of malware compared to the number of apps. These apps will thus be the most useful for the least secure markets.

## 8 Future work

Most of the reviewed tools require some form of manual labour. Also, the markets are currently not completely transparent in the security they have for checking apps that are in their stores. The most important future work is thus to check how these tools can be integrated into the markets.

## Acknowledgements

The authors wish to thank Dr. D. Bucur for reviewing this paper as an expert in the field of the Android OS. The authors would also like to thank their fellow students for reviewing draft versions of their paper.

## References

[1] Gartner. URL `http://www.gartner.com/newsroom/id/1622614`. Last access on Mar. 4th, 2014.

[2] Appbrain. URL `http://www.appbrain.com/stats/number-of-android-apps`. Last access on Mar. 4th, 2014.

[3] Google. URL `http://developer.android.com/guide/topics/manifest/manifest-intro.html`. Last access on Mar. 6th, 2014.

[4] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12, pages 291–307, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-30920-5. doi: 10.1007/978-3-642-30921-2_17. URL `http://dx.doi.org/10.1007/978-3-642-30921-2_17`.

[5] Techterms. URL `http://www.techterms.com/definition/malware`. Last access on Mar. 10th, 2014.

[6] Cisco. URL `https://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf`. Last access on Mar. 6th, 2014.

[7] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4681-0. doi: 10.1109/SP.2012.16. URL `http://dx.doi.org/10.1109/SP.2012.16`.

[8] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046779. URL `http://doi.acm.org/10.1145/2046707.2046779`.

[9] Android Hiroshi Lockheimer, VP of Engineering. URL `http://googlemobile.blogspot.nl/2012/02/android-and-security.html`. Last access on Mar. 7th, 2014.

[10] Daniel Eran Dilger. URL `http://appleinsider.com/articles/13/03/04/samsung-adds-security-layer-to-android-to-gain-enterprise-credibility`. Last access on Mar. 10th, 2014.

[11] Ryan Whitwam. URL `http://www.extremetech.com/computing/130424-circumventing-googles-bouncer-androids-anti-malware-system`. Last access on Mar. 7th, 2014.

[12] Brad Reed. URL `http://bgr.com/2014/03/05/android-malware-google-play/`. Last access on Mar. 8th, 2014.

[13] Malwarebytes corporation. URL http://www.malwarebytes.org/mobile/. Last access on Apr. 4th, 2014.

[14] AVG Technologies. URL http://www.avg.com/nl-nl/antivirus-for-android. Last access on Apr. 4th, 2014.

[15] Bitdefender. URL http://www.bitdefender.com/solutions/mobile-security-android.html. Last access on Apr. 4th, 2014.

[16] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1924943.1924971.

[17] Gabor Paller. URL http://dedexer.sourceforge.net/. Last access on Apr. 4th, 2014.

[18] Johannes Hoffmann, Martin Ussath, Thorsten Holz, and Michael Spreitzenbarth. Slicing droids: Program slicing for smali code. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1844–1851, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1656-9. doi: 10.1145/2480362.2480706. URL http://doi.acm.org/10.1145/2480362.2480706.

[19] T.J. Watson Libraries for Analysis (WALA). URL http://wala.sourceforge.net/. Last access on Mar. 4th, 2014.

[20] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W. Freeh. Taming information-stealing smartphone applications (on android). In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, TRUST'11, pages 93–107, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21598-8. URL http://dl.acm.org/citation.cfm?id=2022245.2022255.

[21] Ba Wichmann, Aa. Canning, D. L. Clutterbuck, L A Winsborrow, N. J. Ward, and D. W. R. Marsh. Industrial perspective on static analysis. *Software Engineering Journal*, 1995.

[22] Thomas Ball. The concept of dynamic analysis. In Oscar Nierstrasz and Michel Lemoine, editors, *Software Engineering ESEC/FSE 99*, volume 1687 of *Lecture Notes in Computer Science*, pages 216–234. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66538-0. doi: 10.1007/3-540-48166-4_14. URL http://dx.doi.org/10.1007/3-540-48166-4_14.

[23] Wes Masri and Andy Podgurski. Using dynamic information flow analysis to detect attacks against applications. In *Proceedings of the 2005 Workshop on Software Engineering for Secure Systems&Mdash;Building Trustworthy Applications*, SESS '05, pages 1–7, New York, NY, USA, 2005. ACM. ISBN 1-59593-114-7. doi: 10.1145/1082983.1083216. URL http://doi.acm.org/10.1145/1082983.1083216.

[24] JR Raphael. URL http://blogs.computerworld.com/android/23590/google-android-security. Last access on Mar. 8th, 2014.

# Tracking Communities in Dynamic Social Networks

Marc Holterman and Arjen Zijlstra

**Abstract**— Social networks are networks of individual actors that are related by any sort of social interaction, just as is known from Facebook and Twitter. There are specific regions or subsets within these social networks that are particularly interesting, namely the regions that are more densely connected than others, known as communities. A lot of research has been done on identifying different communities within social networks, however, most of them involve static graph analysis that does not take time based evolution into account. Since real-time social networks tend to change very rapidly over time, the existing identification strategies for detecting communities fall short and new methods having this time constraint need to be developed.
Social networks that change quickly over time are called dynamic social networks. One of the key problems in dealing with dynamic social networks is volatility. Detecting communities within static graphs is already very expensive, so recomputing communities after every change is not an appealing option. Many attempts have been made to make use of heuristics to accurately identify communities in dynamic social networks. Furthermore, one of the possibilities is to identify the community structure only once and to adaptively update this structure based on the activities that occur.
In this paper, we describe different approaches for the identification and tracking of communities in dynamic social networks. We critically look at heuristics that can be used to track communities over time. Furthermore, we will explore an adaptive way of updating communities according to events that occur during the evolution of a social network.

**Index Terms**—Dynamic Social Networks, Community Identification, Dynamic Graphs

## 1 Introduction

A *social network* is a structure that models the relations within a certain collection of individuals. These social networks contain interesting regions called *communities*, which are groups of individuals with denser connections within the group and fewer connections between groups. These groups play an important role in information flow in a social network, so identifying these groups could give insight in controlling this flow of information.

A social network can be modelled using a graph in which the actors are represented by nodes and the relations between actors are represented by edges. This results in a static graph representing the state of a social network at a given moment in time, which can easily be used for analysis. The downside of a static graph is its inability to cope with time dependent data, that is the addition and deletion of interactions between individuals and thereby the creation of a new graph of the current state of the network. These time dependent social networks are called *dynamic social networks*. We need more than just nodes and links, since time needs to be modelled as well. A dynamic social networks is essentially represented as a sequence of static graphs for certain moments in time, thereby effectively modelling different observations of the network and capturing the changes that occur within the network between time stamps. This way, it is useful to represent a dynamic social network using the definition of a static social network and extending it by the addition and deletion of links per time step.

There is a wide variety of methods available for analysing static social networks. The main task of these algorithms is detecting communities, which basically comes down to the identification of meaningful clusters within the given graph. An example of such a structure of communities in a small network is given in figure 1. The majority of social analysis methods that are currently available focus on these static graphs as their input and they are in general not capable of handling social networks that undergo changes [4]. When time passes, social networks evolve, since new social interactions between individuals develop and others are lost. The new graphs that arise can be re-analysed using the same methods; however, given that this is a very time-consuming process (the optimal solution is NP-hard [12]) it is far from ideal to reapply these methods on every new state. Given that the new state is probably fairly similar to the previous one, it would be far more efficient to have methods that can cope with time-dependent data.

An algorithm able to deal with dynamic social networks could make use of this definition by computing communities at a certain point in time and using the updates of the social interactions to adaptively adjust the communities to this [10]. This saves a lot of computation time by avoiding to recompute from scratch, which makes it feasible to keep track of communities in a social network that develops in real-time, which can be of considerable usefulness to many practical applications. One might think of the dissemination of information or spread of diseases within a population; also some more abstract applications such as tracking changes within links between internet pages or being able to route information through a shorter path by making use of the community someone belongs to. Being able to identify these communities in real-time can help to understand these disseminations and help us to improve, for example, routing in communication networks.

The remainder of the paper is structured as follows. In the next section we provide a small conceptual of preliminaries in which we introduce terms, notation and other tools used in this paper, continuing in the third section by describing the methods that can be used to find these communities adaptively. In the fourth section we critically discuss the findings and finally we conclude in section 5.

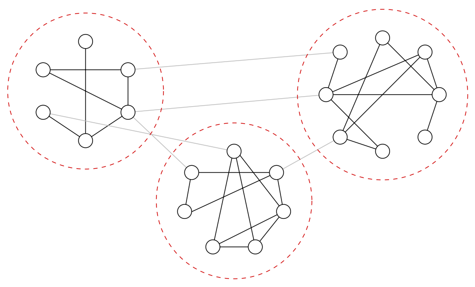


Fig. 1. Three communities in a small network example. Communities are denoted with the dashed circle. [9]

• *Marc Holterman, Master student Computing Science, Software Engineering and Distributed Systems at the University of Groningen.*
• *Arjen Zijlstra, Master student Computing Science, Intelligent Systems at the University of Groningen.*

## 2 Preliminaries

In this section, we describe the notation and models used in this paper. These consist of the representation of (dynamic) social networks as graphs and also the objective function, which measures the quality of a community structure. All of these are needed while describing the methods discussed in section 3.

First of all, note that terms as *node*, *vertex* and *individual* as well as *edge*, *link*, *connection*, *interaction* and *relation* are used interchangeably. Also, a *snapshot* is the same as an *observation*, however a *group* at a certain point in time is not necessarily also a *community*. A group just looks at one specific observation in time (like a 'static' community), while a 'dynamic' community takes a longer sequence of observations into account.

## 2.1 Social networks

Social networks can be represented as undirected graphs, in which nodes represent individuals and edges represent social interactions. Let $G = \langle V, E \rangle$ be such an undirected graph representing a social network with $n = |V|$ individuals and $m = |E|$ social interactions. Communities can be represented as sets of individuals, where $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is the partition of communities in $G$. For each vertex $v$ let $d_v$ denote its degree, $C(v)$ the community it belongs to, $N(v)$ the set of adjacent vertices and $NC(v)$ the set of its adjacent communities. Furthermore, for each $S \subseteq V$ let $m_S$ denote the number of links inside S, $d_S$ the total degree of the vertices in $S$ and $e_S^v$ the number of interactions from $v$ to other individuals in $S$.

## 2.2 Dynamic social networks

Dynamic social networks can be represented by binding an undirected graph to the specific time which it represents in the evolution of the social network. Let $G^s = \langle V^s, E^s \rangle$ be an observation of a dynamic social network at time $s$. Now let $\Delta G^s = \langle \Delta V^s, \Delta E^s \rangle$ be the change of the social network at time $s$, where $\Delta V^s$ are the added nodes to the network and $\Delta E^s$ are the added links to the network. Using this definition, the update of an observation at time $s$ can be written as $G^{s+1} = G^s \cup \Delta G^s$. Now, a dynamic social network is represented as a sequence of these observations $\mathcal{G} = \left(G^0, G^1, \ldots G^s\right)$ and $\mathcal{D} = \left(\mathcal{C}^0, \mathcal{C}^1, \ldots \mathcal{C}^s\right)$ the corresponding community structures. Using this, an adaptive algorithm $\mathcal{A}$ which makes use of $C(G^t)$ and $\Delta G^t$ to be able to find $C(G^{t+1})$ is graphically shown in figure 2.

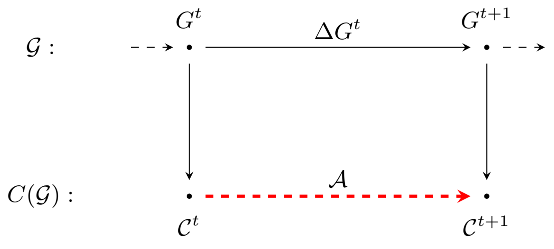


Fig. 2. Times $t$ and $t+1$ of dynamic social network $\mathcal{G}$ and the set of communities at these times $C(\mathcal{G})$. The adaptive algorithm $\mathcal{A}$ computes the communities at time $t+1$ from the communities at time $t$, $\mathcal{C}^t$ and the changes at time $t$, $\Delta G^t$. [10]

## 2.3 Tracking

When tracking the communities in a social network over time, several different events can happen with respect to the evolution of a community.

**Birth** A community that cannot be associated with an already existing dynamic community in $\mathbb{D}$ is found.

**Death** A dynamic community that is not observed for a given number of consecutive time steps.

**Merging** Two dynamic communities observed at time $t-1$ are matched to a single community observed at time $t$.

**Splitting** Two dynamic communities observed at time $t$ match to a single community observed at time $t-1$, i.e. the opposite of *merging*.

**Expansion** A dynamic community grows ($> 10\%$ growth) from one time-step to the next.

**Contraction** A dynamic community shrinks ($> 10\%$ reduction) from one-time step to the next.

This definition is used by Green et al. [4] to describe the evolution of communities over time in terms of events. Examples of these life-cycle events are shown in figure 3.

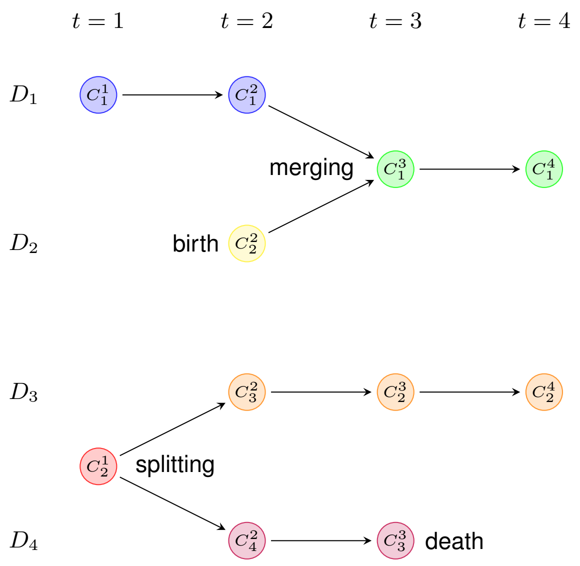


Fig. 3. Example of four dynamic communities tracked over four time steps featuring continuation, birth, death, merging and splitting life-cycle events. [4]

## 2.4 Optimising Cost

Given the definition of a community, we can make the following assumptions about individuals:

- At each time, every group represents a different community. There is a reason if two groups are separated.
- An individual can only belong to exactly one community at a given time.
- An individual can change community over time.
- An individual does not change its community very frequently.
- If an individual changes community often, it is oscillating between a small set of groups.
- An individual is frequently in the group which represents its community.

It is possible to find communities by using the above stated properties to define a colouring problem. This can be written as a function $f : V \rightarrow \mathbb{N}$ for individuals, and $f : \mathcal{P}(V) \rightarrow \mathbb{N}$ for groups, where the natural numbers represent the set of colours used. The colour of a vertex $v^t \in V^t$, represents $v$'s community affilition at time $t$. Similarly, the colour of a group $C \subset V$ denotes what community $C$ represents at time $t$. To measure the quality of a community, violations of the properties listed above are penalised. These are separated in three types of penalties, namely individual, group and colour penalties, abbreviated as *i-cost*, *g-cost* and *c-cost* [12]. These are parametrised to be able to give different importancy to the properties; $\alpha$, $\beta_1$, $\beta_2$ and $\gamma$ are used for this.

**i-cost** is defined by the value of $\alpha$ whenever an individual changes community. i.e. $f(v^t) \neq f(v^{t+1})$.

**g-cost** is defined by the value of $\beta_1$ whenever an individual does not have an edge to his group. i.e. $f(v^t) = f(u^t)$ and $v \notin C, u \in C$ with $C \subset V$ and by the value of $\beta_2$ whenever an individual has an edge to a group of a different colour i.e. $v \in C$, $u \notin C$ and $f(v^t) \neq f(u^t)$.

**c-cost** is defined by the value of $\gamma$ for each colour an individual uses after getting his first colour.

By varying the parameters $\alpha$, $\beta_1$, $\beta_2$ and $\gamma$ it is possible to increase or decrease the importance of properties of the communities found. This way, it is for example possible to let individuals be really loyal to its community by increasing $\alpha$ and/or $\gamma$.

### 2.5 Modularity

When a structure of communities is found in a social network, it would be very useful to know how good this structure is. To quantify this, Newman [9] uses *modularity* as a measure on the quality of a community. In equation 1, modularity ($\mathcal{Q}$) is defined as the fraction of the edges within a given community $C$ minus the expected fraction if edges would have been distributed randomly.

$$\mathcal{Q} = \sum_{C \in \mathcal{G}} \left( \frac{m_C}{m} - \frac{d_C^2}{4m^2} \right) \tag{1}$$

A higher modularity means a better partition of the network in communities. Using this, the goal is to find a set $\mathcal{C}$ of communities such that $\mathcal{Q}$ is maximised. Modularity has some disadvantages. Maximising modularity has two main biases: it tends to merge small clusters into bigger ones and also to split large clusters into smaller ones [8]. However, it is very useful in the methods used in this paper because of its close agreement to real-world networks.

### 2.6 Problem Definition

The problem of finding communities is defined as the problem of colouring individuals according to the community they are affiliated with. This can be done by maximising the modularity of the social network as done by Newman et al. [9] or by minimising the total cost using the i-cost, g-cost and c-cost as done by Tantipathananandh et al. [12]. An example of the colouring of a real-life example is shown in figure 4. As can clearly be seen, communities are nicely separated and they can be clearly distinguished from each other.

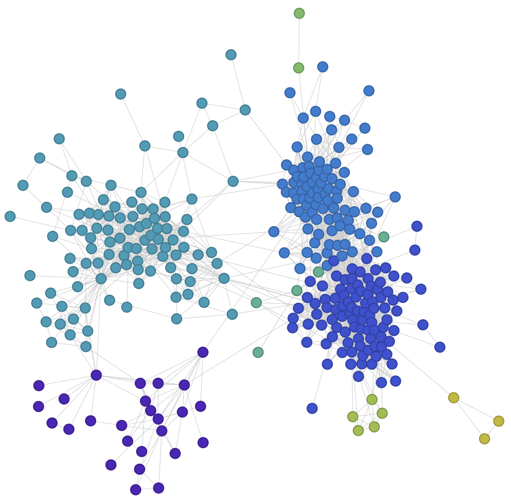


Fig. 4. Static communities identified in a Arjen Zijlstra's Facebook network using Mathematica's `FindGraphCommunities[g]`, making use of modularity-based clustering. Individuals in the same community have the same colour [14, 15]

Now, the central problem is; given a dynamic social network $\mathcal{G}$ consisting of observations $G^0, G^1, \ldots, G^s$ and differences $\Delta G^0, \Delta G^1, \ldots, \Delta G^s$, identify the network community structure at any point in time by using the information about the previous observation combined with the evolution of the network. This problem is further described during the remainder of this paper.

## 3 Methods

In this section we describe different methods for the identification and tracking of communities in dynamic social networks. The first methods are the ones described by Tantipathananandh et al. [12] and Green et al. [4] that describe the tracking of communities by making use of heuristics to identify semi-optimal communities. The last one is an algorithm by Nguyen et al. [10] which updates the community structure by using the previous observation and the changes that occurred at that moment in time.

### 3.1 Tracking Communities

Since the problem of finding an optimal colouring of the vertices per community is known to be NP-hard [12], it is useful to look at heuristics to approximate the optimal solution. But before looking at heuristics, an optimal solution based on exhaustive search and dynamic programming is presented, which is used as a basis for the heuristics. Once it is shown that this optimal solution actually helps in understanding the communities in a social network, the question is whether the solutions produced by heuristics are less informative when run on a real-world example.

#### 3.1.1 Optimal Solution

Once the colouring for a group of vertices has been found, an optimum colouring for the individual vertices can be found using dynamic programming. The problem is then reduced to, given a colouring of the group vertices and find the minimum cost colourings for each of the individuals. The total cost is defined as the sum over all individuals, their edges and the i-costs, g-costs, and c-costs.

Given a group colouring, the minimum cost of colouring the individual $i$ at a given time $T$ is [12]:

$$\min_{S \in \Phi(T), x \in S} \Gamma(T, S, x). \tag{2}$$

Where $\Phi(t) = \{S \subseteq C \mid 1 \leq |S| \leq t\}$ denotes the collection of all possible subset of colours used between time step 1 and $t$, and $\Gamma$ is the recurrence for the minimum cost of colouring $i$ in time step $t$ with colour $x \in S$.

$$\Gamma(t, S, x) = G(t, x) + \min_{\substack{R \,\in\, \Phi(t-1),\ y \,\in\, R, \\ R \,\cup\, \{x\} = S}} \left(\Gamma(t-1, R, y) + I(t, x, y) + C(x, R)\right) \tag{3}$$

$$\Gamma(1, \{x\}, x) = G(1, x) \tag{4}$$

Where $G(t, x)$ is the g-cost of colouring individual $i$ at time step $t$ with colour $x$, $I(t, x, y)$ is the i-cost of colouring individual $i$ at time steps $t$ and $t-1$ with colours $x$ and $y$ respectively, and $C(x, R)$ is the c-cost of using colour $x$ with $R$ as the set of colours of previous steps. In other words, given a group colour the minim cost of colouring individual $i$ at time $t$ comes down to the sum of the g-costs and the minimum of the sum of the i-costs and c-costs.

To avoid the exponential time constraint heuristics are investigated. Once the heuristic has found a group colouring, dynamic programming still applies to colour the individuals.

#### 3.1.2 Bipartite Matching Heuristic

Bipartite Matching Heuristic is largely based on the intuition that a group colour is good if most of the individuals keep the same colour from one step to the next one. This avoids either i-costs or g-costs to increase. For all groups $g \subset V^t$ and $g' \subset V^{t+1}$ at time steps $t$ and $t+1$ respectively, an edge between $v_g^t$ and $v_{g'}^{t+1}$ is added with weight $|g \cap g'|$. Then, using standard flow techniques a maximum weight bipartite matching among the group vertices for those time steps is found [12, 13]. This can be improved by enumerating many maximal matchings and choosing the best one based on the actual cost. This method aims at minimising i-costs rather than g-costs.

#### 3.1.3 Greedy Heuristic

So, the bipartite matching algorithm is focused on preserving as much similarity as possible from one step to another. Here, similarity is normalised to the interval $\langle 0, 1\rangle$, where disjoint groups have similarity 0 and identical groups have similarity 1. The measure used for similarity is Jaccard's index [5], adjusted to give more weight to similar groups that exist in closer temporal proximity (JacD).

$$\text{Jac}(g, g') = \frac{|g \cap g'|}{|g \cup g'|} \tag{5}$$

$$\text{JacD}(g, g') = \frac{\text{Jac}(g, g')}{|t - t'|} \tag{6}$$

Where $g$ and $g'$ are two groups occurring at times $t$ and $t'$ respectively, with $t \neq t'$.

The most basic version of greedy heuristic works similar to Kruskal's Minimum Spanning Tree algorithm [6]. It works by repeatedly selecting a pair of groups $g$, $g'$ with the highest similarity and assigning them with the same colour. When the algorithm visited all edges this way, the algorithm gives each component of groups its own colour and terminates.

This algorithm can be used in reversed order, by searching for colours by looking at earlier time step. At time 1 all groups have a unique colour. At time $t$, a group $g$ is coloured by colouring it with the same colour as group $g'$ at time $t'$ with $t' < t$ of largest similarity. This algorithm is called the *Backward Greedy* algorithm.

By requiring that links between groups point as least into the past as possible, a more restrictive version of the Backward Greedy algorithm is obtained. When considering group $g$ at time $g'$. This can be done by selecting the latest time $t' < t$ such that there exists a group $g'$ at time $t$ with similarity higher than 0 with group $g$. The colouring is done the same as in the Backward Greedy algorithm. This new algorithm is called *Least Delay Greedy* algorithm.

### 3.2 Adaptively Updating Communities

In this section, we describe an adaptive method which focusses on maximising and keeping the modularity of a given dynamic social network intact. To identify an initial community structure, one can make use of one of the widely available static algorithms to find communities, for example the ones described in Blondel et al. [1], Clauset et al. [2] or Newman et al. [9].

#### 3.2.1 Community structures

Vertices can only be included in one of the communities for a given graph $G^i$. Communities are linked together by edges or relations as known in a social networks. There are two different kinds of edges in the community structure, which are called *intra-community* links and *inter-community* links [10]. *Intra-community* links are edges that have both ends within the same community, while *inter-community* links are edges that have both ends in different communities and hence are being referred to as bridges that connect two communities. For each community $C$ in $G$, the number of intra-community links is much higher than the number inter-community links. Therefore, the community $C$ is much more densely connected inside than it is outside. A community is a region in a social network in which the nodes are more densely connected to others within this region as to others outside the region. In other words, the number of intra-community links is way higher than the number of inter-community links.

During the flow of time, relations between nodes tend to change. A certain vertex $v$ can create a new relation thus creating a new edge $e$ which can be either an intra- or inter community link. This affects the community in multiple ways; if this $e$ would be an inter-community link the vertex might be pulled out of the community because the connection with other vertices might become more powerful or heavier. There are four possible events that mark a transition, namely a graph $G^t$ might differ from $G^{t+1}$ because it has a new node, there is a new edge or there might have been a deletion of a certain node or edge.

These four events either strengthen or weaken a community structure. The details are as follows.

- $newNode(V + u)$ : A new node $u$ is added to the set of vertices $V$ thereby introducing new possibly edges $e_k$ connecting $u$ to $G$.
- $deleteNode(V - u)$ : Node $u$ is removed from the set of vertices $V$ thereby removing all its adjacent edges $e_k$.
- $newEdge(E + e)$ : A new edge $e$ is added to the set of edges $E$ connecting two nodes $(v, w)$ together.
- $deleteEdge(E - e)$ : Edge $e$ is removed from the set of edges $E$ thereby deleting a relation between $v$ and $w$ in $G$.

All these events have an effect on the community structure since they can either strengthen or loosen nodes with respect to their corresponding community. The newly added edge might be another intra-community link which tightens the bond within a community or it might be that the addition changes two communities by handing over an affected vertex, effectively adopting the vertex into a new community.

#### 3.2.2 Adaptive Algorithm

Within a network that has an initial community structure $\mathcal{C}$ obtained from running the static algorithms on $G^0$, each vertex represented within the graph is related to two forces. The forces indicate whether a vertex belongs to a certain community. First, there is a force $F_{in}$ that keeps a vertex bounded to a certain community and second, there is the opposite force $F_{out}$ that originates from all other communities and tries to pull the vertex out of the network towards another community. This battle between these two different forces eventually decides to what community all the different vertices belong and thus creates a partition $\mathcal{C}$ which makes up $G$.

Creating new consecutive states $G^t$ is computationally easy since it is simply another snapshot for a certain moment in time. However, creating the corresponding community structure $\mathcal{C}^t$ for this same snapshot, requires considerably more computational power since it involves redoing the entire calculation used to compute $\mathcal{C}^0$. The adaptive algorithm for community identification computes from a combination of the four events listed in section 3.2.1 a new $C^{t+1}$. Let us consider these four events one by one;

*NewNode*: The addition of a new node $u$ in the network has a few implications based on the number of associated connections it comes with. Namely, when the $u$ does not have any connections with the network it is simply a community of its own and it leaves the modularity of the network intact. The interesting case is when it does have connections with the adjacent edges with the current community structure. In this case, we need to determine where to put the new node $u$ such that the overall modularity of the network stays maximal. We do this by computing the out-force ($F_{out}$) of every adjacent community and determine which of the adjacent communities is pulling the hardest on the new node $u$. The community that has the biggest force on $u$ adopts the node to its community structure. The method to perform this calculation is shown in algorithm 1.

*NewEdge*: The addition of an edge $e$ we have to consider two possibilities. The edge $e$ can be an intra-community connection (which means that both the endpoints, nodes $u$ and $v$, are members of the same community) or the edge $e$ can be an inter-community connection (which means that both the endpoints, nodes $u$ and $v$, are members of different community). In the former case nothing changes since it merely strengthens the community structure. The latter case however, is more interesting since we now have to calculate the changes in modularity. We will only move a vertex to another community if this increases the total modularity of the network. We have three different situations. First of all, the new edge does not cause any chances in the community structure and the edge is added. Secondly, the new edge causes vertex $u$ to join $C(v)$ or the other way around; the vertex $v$

**Algorithm 1:** NewNode

```
    Input: New node $u$ with associated links; Current structure $\mathcal{C}^t$
    Output: An updated structure $\mathcal{C}^{t+1}$
1   Create a new community of only $u$;
2   for $v \in N(u)$ do
3       Add community $C(v)$ to $NC(u)$;
4   end
5   for $C \in NC(u)$ do
6       Find $F^C_{\text{out}}$;
7   end
8   Let $C_u \leftarrow \arg\max_C \{F^C_{\text{out}}(u)\}$ ;
9   Update $\mathcal{C}^{t+1}$ : $\mathcal{C}^{t+1} \leftarrow (\mathcal{C}^t \setminus C_u) \cup (C_u \cup u)$;
```

joins $C(u)$. The effects of these changes are computed and the final decision is made upon which of them results in the highest modularity of the network. The method to perform this calculation is shown in algorithm 2. Here, $\Delta q_{u,C(u),C(v)}$ denotes the change in modularity when placing node $u$ from its current community to the one from $v$.

**Algorithm 2:** NewEdge

```
    Input: Edge $\{u, v\}$ to be added; Current structure $\mathcal{C}^t$
    Output: An updated structure $\mathcal{C}^{t+1}$.
1   if $C(u) == C(v)$ then
2       $\mathcal{C}^{t+1} \leftarrow \mathcal{C}^t \cup \{u, v\}$;
3   else if $C(u) \neq C(v)$ then
4       if $\Delta q_{u,C(u),C(v)} < 0$ and $\Delta q_{v,C(u),C(v)} < 0$ then
5           return $\mathcal{C}^{t+1} \equiv \mathcal{C}^t$;
6       else
7           $w = \arg\max\{\Delta q_{u,C(u),C(v)}, \Delta q_{v,C(u),C(v)}\}$;
8           Move $w$ to the new community;
9           for $t \in N(w)$ do
10              Let $t$ determine its best community;
11          end
12          Update $\mathcal{C}^{t+1}$;
13      end
14  end
```

*DeleteNode*: When a node is removed from the network, it is inherently removed from the corresponding community as a consequence. This has a few complications because the resulting community might be very complex [10]. Consider the two extremes; where you either remove a node with only one connection to his community or the node that has the most relations within the community. If the targeted vertex has only one link to the corresponding community the community structure stays the same since there are no other influences. However, removing the most popular node from a community is deadly. This can result in a complete split of the community as is shown in 5.

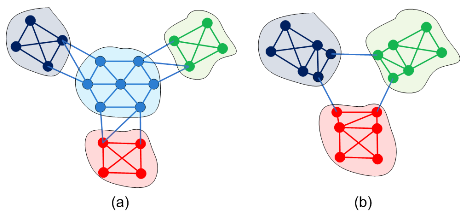


Fig. 5. a) The original network with four communities b) After the highest degree node is removed, the leftover nodes join in different modules, forming a new network with three communities [10].

To deal with this immense task something called the clique percolation method [11] is used. This methods makes use of the properties of complete graphs in order to regroup the leftovers from the previous community. It selects the complete graph $k_3$ and uses this to bind other nodes until no other node is able to join. The procedure for joining is similar to the addition of a node discussed previously. Figure 6 shows the deletion of the most popular node $g$ and the recovery using the clique percolation method.

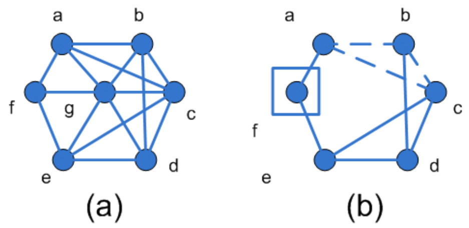


Fig. 6. a) The original community b) When the central node $g$ is removed, a 3-clique is placed at $a$ to discover $b$, $c$, $d$ and $e$. $f$ is assigned singleton afterwards [10].

The remaining communities are left to choose their best community to merge with. The method upon node deletion is shown in algorithm 3.

**Algorithm 3:** DeleteNode

```
    Input: Node $u \in \mathcal{C}$ to be removed; Current structure $\mathcal{C}^t$.
    Output: An updated structure $\mathcal{C}^{t-1}$
1   $i \leftarrow 1$;
2   while $N(u) \neq \varnothing$ do
3       $S_i$ = {Nodes found by a 3-clique percolation on $v \in N(u)$};
4       if $S_i == \varnothing$ then
5           $S_i \leftarrow \{v\}$;
6       end
7       $N(u) \leftarrow N(u) \setminus S_i$;
8       $i \leftarrow i + 1$;
9   end
10  Let each $S_i$ and singleton in $N(u)$ consider its best communities;
11  Update $\mathcal{C}^t$;
```

*DeleteEdge*: When deleting an edge $e = (u, v)$ there are four possible cases which have to be addressed. Three of which do not significantly affect the community structure. The first two cases concern either or both $u$ and $v$ having degree one. When both have degree one the result is two singleton groups $u$ and $v$ and it does not have any effect on the community structure as a whole. When either of the two has degree one the effects on the network community structure is yet again negligible because it results in the previous structure plus a group with only $u$ or $v$. When $e$ is an inter-community link the removal of $e$ will strengthen the community structure and therefore makes no changes. The last case, the case that $e$ is an intra-community edge is the most tricky one. Figure 7 shows the removal of an edge which results in the split of the blue community.

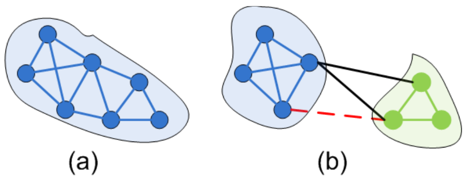


Fig. 7. a) The original community b) After an edge (in dotted line) is removed, the community is broken into two smaller communities [10].

The intra-community edge $e$ that is about to be removed, leaves behind a host community that is less densely connected than before. The only possibility is to test all the ‘quasi-cliques’ [10] and find the maximal cliques similar to the one described in [11]. The quasi-cliques as well as the singleton nodes can be asked to find their best suited community. This is the most time consuming operation of all events listed above. The method to perform this operation is shown in algorithm 4.

**Algorithm 4:** DeleteEdge

```
   Input: Edge $(u, v)$ to be removed; Current structure $\mathcal{C}^t$
   Output: An updated clustering $\mathcal{C}^{t+1}$
1  if $(u, v)$ is a single edge then
2      $\mathcal{C}^{t+1} = (\mathcal{C}^t \setminus \{u, v\}) \cup \{u\} \cup \{v\}$;
3  else if Either $u$ (or $v$ is of degree one then
4      $\mathcal{C}^{t+1} = (\mathcal{C}^t \setminus C(u)) \cup \{u\} \cup \{C(u) \setminus u\}$;
5  else if $C(u) \neq C(v)$ then
6      $\mathcal{C}^{t+1} = \mathcal{C}^t$;
7  else
8      % Now $(u, v)$ is inside a community $C$ %
       $L = \{$Maximal 'quasi-cliques' in $C\}$;
9      Let the singletons in $C \setminus L$ consider their best communities;
10 end
11 Update $\mathcal{C}^{t+1}$;
```

The four aforementioned events are combined in the main algorithm QCA described by Nguyen et al. [10] that quickly updates a dynamic social network. QCA is shown in algorithm 5.

**Algorithm 5:** Quick Community Adaptation

```
   Input: $G \equiv G^0(V^0, E^0), \mathcal{E} = \{\mathcal{E}^1, \mathcal{E}^2, \ldots, \mathcal{E}^s\}$ a collection of
          simple events
   Output: Community structure $\mathcal{C}^t$ of $G^t$ at time $t$.
1  Use Blondel et al. [1] to find an initial community structure $\mathcal{C}^0$ of
   $G^0$;
2  for (t ← 1 to s) do
3      $\mathcal{C}^t \leftarrow \mathcal{C}^{t-1}$;
4      if $\mathcal{E}^t = newNode(u)$ then
5          New_Node($\mathcal{C}^t, u$);
6      else if $\mathcal{E}^t = newEdge((u, v))$ then
7          New_Edge($\mathcal{C}^t, (u, v)$);
8      else if $\mathcal{E}^t = removeNode(u)$ then
9          Remove_Node($\mathcal{C}^t, u$);
10     else
11         Remove_Edge($\mathcal{C}^t, (u, v)$);
12     end
13 end
```

## 4 Discussion

First of all, we would like to point out that there is no widely accepted unique definition for the word community. One might argue that a community is a group of nodes sharing properties that draws them together and thus a densely connected group of vertices. Others claim that the word community is for groups that share these properties over a longer period over time.

Tantipathananandh et al. [12] found that, when evaluating the heuristics described in section 3.1 on small synthetic datasets with known embedded communities and on real-life datasets, the communities identified by the heuristics were similar to the communities identified by the algorithms that find the optimal solution. Furthermore, the resulting costs of the heuristics are not significantly higher compared to the optimal costs. However, they do not consider any large social networks, nor do they look at the performance over long time periods.

Green et al. [4] focussed more on the scalability of these and other heuristics, by generating bigger synthetic networks containing the different event types; birth, death, splitting and merging, based on the techniques proposed by Lancichinetti & Fortunato [7]. Furthermore, they evaluated a large real-world dataset (on which the heuristics performed very well), but they did still not consider a very long period of time.

All of these heuristics still largely depend on the traditional static methods for finding communities. They use these to find communities at a certain moment in time and try to colour these to the ones found in the previous time step. The adaptive algorithm described by Nguyen et al. [10] has our preference over the heuristics since they do not heavily rely on static algorithms to detect communities.

## 5 Conclusion

In this paper we have looked at different methods for tracking communities. We have seen an algorithm that uses heuristics and makes use of several cost properties to measure the similarities between groups at different moments [12]. This approach is not fully investigated on the scalability and therefore probably less promising than the adaptive approach. But since these heuristics can easily be adjusted by varying the cost parameters, they might provide more possibilities when tested on a broader scale.

Furthermore, we described an approach that adaptively updates communities found at a given moment in time. This is done by tracking the changes and adapting the structure while keeping the modularity maximised [10]. This approach is very effective in identifying high quality community structures and updating it according to the changes that occur in large rapidly changing social networks. We are interested in this method being further developed and tested on large real-life dynamic social networks.

## Acknowledgements

The authors wish to thank Doina Bucur for the great support on both theoretical as well as writing issues.

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

[3] L. C. Freeman. Finding social groups: A meta-analysis of the southern women data. *Dynamic social network modeling and analysis*, pages 39–97, 2003.

[4] D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, pages 176–183. IEEE, 2010.

[5] P. Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[6] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[7] A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.

[8] A. Lancichinetti and S. Fortunato. Limits of modularity maximization in community detection. *Physical Review E*, 84(6):066122, 2011.

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

[10] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai. Adaptive algorithms for detecting community structure in dynamic social networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 2282–2290. IEEE, 2011.

[11] G. Palla, P. Pollner, A.-L. Barabási, and T. Vicsek. Social group dynamics in networks. In *Adaptive Networks*, pages 11–38. Springer, 2009.

[12] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726. ACM, 2007.

[13] T. Uno. *Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs*. Springer, 1997.

[14] S. Wolfram. Data science of the facebook world. Website, April 2013. blog.stephenwolfram.com/2013/04/data-science-of-the-facebook-world/.

[15] Wolfram Mathematica. FindGraphCommunities. Website, 2014. reference.wolfram.com/mathematica/ref/FindGraphCommunities.html.

[16] Z. Ye, S. Hu, and J. Yu. Adaptive clustering algorithm for community detection in complex networks. *Physical Review E*, 78(4):046115, 2008.

# Applying link analysis algorithms to psychopathology symptom networks

Laurence de Jong *CS: SE&DS, University of Groningen*, Diederik Jan Lemkes *CS: SE&DS, University of Groningen*

**Abstract**— A recent discovery in the field of psychopathology is that the co-occurrence of two mental disorders *(comorbidity)* is prevailing. The fact that comorbidity is prevailing might be due to traditional psychometric approaches where symptoms are used as a measurement for mental disorders. However, a more causal relationship between symptoms seems to be plausible. In previous research a network has been constructed from the symptoms of mental disorders described in the *Diagnostic and Statistical Manual of Mental Disorders-IV* (DSM-IV).
In our paper we study the applicability of the *link analysis algorithms* PageRank, HITS and SALSA. The objective is to analyze which algorithm will theoretically yield the best results when it comes to identifying symptoms which will lead to comborbidity.
We created a thought experiment in which the algorithms have been applied to two types of symptom networks. The first type is the network created from the DSM-IV, the second symptom network type consists only of causal relationships between symptoms and does not exist for the full DSM-IV symptom spectrum. We assess properties of the networks and algorithms on their impact on identifying important symptoms.
The first and foremost result that emerges is that in order for link analysis to be of any added value, a directed graph should be preferred over an undirected graph. In the case of HITS, an undirected graph would not even yield sensible results. The second result is that the damping factor of PageRank can be interpreted in multiple ways which each have their own problems. The best results are expected from SALSA because this algorithm combines parts of PageRank and HITS with an improvement in overcoming a problem called *Tightly Knit Communities*. We conclude that in theory the best results are obtained by application of SALSA and HITS on network type 2, although we advise that all three algorithms should be put to practice to verify our expectations.

**Index Terms**—Psychopathology, link anlysis, web graph, PageRank, HITS, SALSA, DSM-IV

## 1 Introduction

The co-occurrence of two mental disorders (comorbidity) has recently been discovered to be not exceptional but rather prevailing [12]. Moreover, comorbidity has been consistently linked to a poorer prognosis, greater interference with activities of daily living, higher suicide rates and an increased need for professional help [1, 18]. This, and the fact that almost half of the people who have been diagnosed for one mental disorder also get diagnosed for a second mental disorder [11], has lead to extensive research into comorbidity.

In traditional psychometric approaches, observable symptoms have been treated as latent variables for diagnosing a disorder (condition). In other words, symptoms are explained by their disorders, and accordingly, symptoms are used as measurements in rating a persons qualification for a disorder, for example, a panic disorder could cause observable symptoms like panic attacks. Until recent years symptoms have been treated as being individual, mutually exclusive entities. However, in clinical psychiatry a more causal relationship between symptoms seems to be plausible. When looking at figure 1, one can imagine that a Major Depressive Episode (MDE) may co-occur with Generalized Anxiety Disorder (GAD) because of possible causal relationships between symptoms. Sleep deprivation may cause fatigue, which may cause concentration problems to emerge. In the end, irritability may be caused by the concentration problems itself [4].

The insight of possible causal relationships between symptoms indicates that there may not be only one root cause of mental disorder comorbidity, but that there may be a chain of symptoms that, because of the direct relationship between the symptoms, will cause multiple mental disorders to surface.

The possibility that symptoms may have a direct relationship raises the question on how to use these causal relationships optimally and interpret symptoms and mental disorder comorbidity. In recent research a network approach is used to encompass the possible causal relationship between symptoms [8, 4, 3]. Modeling symptoms and the

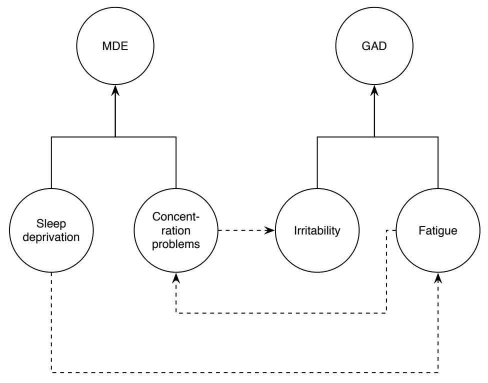


Fig. 1. Possible causal relationships between symptoms related to Major Depressive Episode (MDE) and Generalized Anxiety Disorder (GAD)

interaction between symptoms in a network can provide insight on a mental illness.

The network in figure 3 is created from the symptoms and disorders described in the *Diagnostic and Statistical Manual of Mental Disorders-IV* (DSM-IV) [2]. The DSM-IV is a handbook for psychotherapists, which was initially created for psychiatric organizations to create a standard for classifying mental disorders. However it is currently also used by, among others, psychiatric drug regulation agencies, health insurance companies, the legal system for standardization and as a common language for classification of mental disorders. The DSM-IV describes the various symptoms related to a mental illness and aids therapists to make closely aligned diagnoses.

The application of link analysis algorithms to symptom networks may provide an answer to crucial questions surrounding the relations between symptoms. Amongst the crucial questions are: How important are symptoms that overlap between two disorders as sources of comorbidity? Can we identify symptoms of a disorder that put someone at more risk of developing a second disorder compared to other symp-

• *Laurence de Jong, e-mail: research@ldej.nl.*
• *Diederik Jan Lemkes, e-mail: research@djlemkes.nl.*

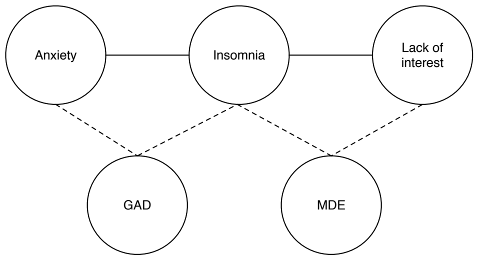


Fig. 2. Indirect connection in network type 1 as defined by Borsboom et. al [4]

toms? Is there an order in which people generally develop one particular disorder first and another disorder second? Knowledge about causal relationships could also help with detecting *feedback loops*. An example of a feedback loop is the fear of fear example. The fear of fear case, first introduced by McNally [16], describes a person that develops a fear of an existing fear the person already has. For instance, being afraid of the anxiety that comes with not being able to sleep. Identifying causal relationships between symptoms could help preventing and breaking out of these feedback loops.

Our goal is to evaluate the link analysis algorithms PageRank, HITS and SALSA for their applicability on symptom graphs. To make the analysis, we developed a theoretical thought experiment to determine the possible effects of the algorithms on the graphs. To our knowledge, such an experiment has not been done before.

In the next section we describe two symptom networks, how they are created and which properties they have. After that the link analysis algorithms PageRank, HITS and SALSA each have their own section (in order of mentioning) in which they will be briefly introduced and, following the thought experiment, will be applied to the described symptom networks. From there on we proceed to a conclusion which will summarize the main results, followed by the discussion and a notion for future works.

## 2 Network structure

As stated in the introduction, a theoretical thought experiment on the applicability of an algorithm will be conducted for every algorithm described. These experiments will be run using network type 1 and network type 2 as inputs.

### 2.1 Type 1

The first network we describe has been introduced by Borsboom et. al [4] and was constructed by representing each symptom of DSM-IV as a node. A connection between two nodes is established if they either have a direct or indirect connection. A direct connection is present if both symptoms are indicators for the same mental disorder. An indirect connection is present in the case of shared symptoms, or bridge symptoms, between two nodes. As can be seen in figure 2, anxiety and lack of interest do not have a direct connection because they are not shared indicators for any of the mental disorders described in DSM-IV. However, because they both do have a connection to insomnia, we say that the anxiety and lack of interest nodes are indirectly connected. In the example in figure 2, we define insomnia as being a bridge symptom. The bridge symptom definition is an important (and potentially dangerous) property as will be shown during the algorithm evaluations. The most important note to take from this network structure is that a cluster of directly connected symptoms is used to conceptualize a disorder.

If the network structure that has just been setted out is applied on the full DSM-IV symptom spectrum, the result as depicted in figure 3 is obtained. The most prominent property one can immediately identify when looking at this figure is the giant component that covers 47.8% of all symptoms in the network. Within the giant component, any of the 208 symptoms can be reached by traveling along a path consisting

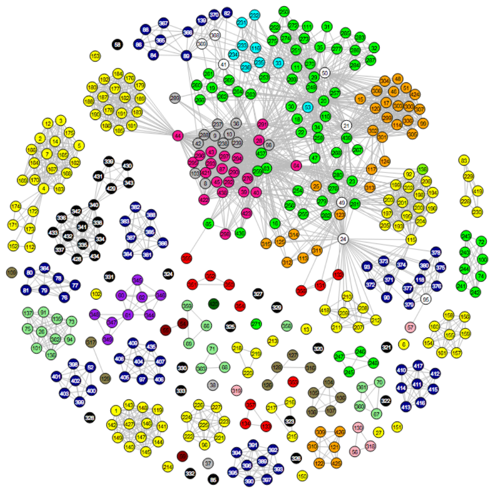


Fig. 3. Network structure 1. First introduced by Borsboom and Cramer [4]. Every node represents a symptom. A link is present when the nodes being connected by the link are both indicators for the same mental disorder.

of one or more links between other symptoms. Even more interesting is the fact that the giant component conforms to the properties of a small world. A component within a network can be defined as being a small world if the average number of hops between two randomly chosen nodes within the component grows proportionally with the logarithm of the number of nodes in the network and is small compared to a randomly constructed network of the same size (both nodes and edges) [20]. Like with the bridge symptom property, the giant component being a small world may be indicators for great care to be taken if a patient shows signs of a symptom within the giant component.

When studying the link structure of figure 3, we can see that undirected, unweighted links are used to connect two nodes. Because the link analysis algorithms being studied in this paper need directed networks (as the internet is a directed network as well), we consider the undirectedness of the network to be bi-directional instead.

### 2.2 Type 2

In practice, a fully bi-directional network as defined by network type 1 will not exist. Also, the link analysis algorithms evaluated in this paper do not produce optimal results for bi-directional graphs. Therefore, we need to construct a symptom network that contains directed links without having the precondition that every link also has to have a bi-directional counterpart. In the domain of symptom networks, such a structure would imply that there is a causal relationship between two symptoms. In other words: such a network would have the knowledge of the symptom at the source of the link causing the symptom at the destination of the link.

Although proving causal relationships between symptoms in the domain of psychopathology is very hard, recent research by Bringmann et. al [6] has shown that applying Experience Sampling Methodology (ESM) can provide insight in causal relationships between symptoms. ESM works by analyzing participants experience and affect over the course of time.

To our best knowledge, current research focusing at mapping causal relationships has not been able to construct a network covering the full DSM-IV symptom spectrum. However, we think that such a network should be possible for at least one type[1] of individuals in the foreseeable future. Therefore, we consider network type 2 as being the directed counterpart of network type 1 with the rejected precondition of every link having to have a bi-directional partner. A very elementary example of a type 2 network is depicted in figure 2.2.

[1]type based on: gender, age, ethnicity, income .

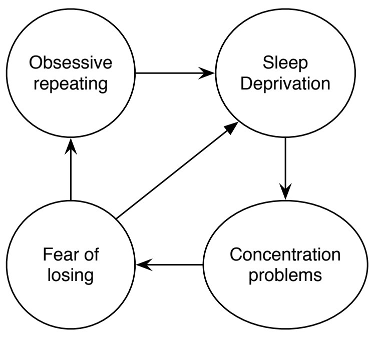


Fig. 4. Elementary example of network type 2. Note that the relations in this figure are based on non-proven assumptions.

### 2.3 Naming conventions

In order to circumvent any indistinctness on naming conventions of networks and its properties, we indicate the following definitions as being the same:

- A network equals a graph
- A symptom is the same as a node or a web-page
- A link is an edge and a relation, and may be directed or undirected

## 3 PageRank

*PageRank* is a link analysis algorithm proposed by Larry Page and Sergey Brin who founded Google [17]. The algorithm is used to measure the relative importance of web pages. While the main purpose of the algorithm is to rank web pages, many other fields of research have found PageRank to be beneficial [9].

### 3.1 Theory

Page and Brin describe the importance of a website as the amount of links that point to this website. However, not only the amount of links pointing to the website matters, also the importance of the website that contains the link is taken into account. Intuitively, pages that are important are more likely to guide surfers to better sources.

A brief description of how the algorithm works is as follows. A random surfer is at a random page. He or she clicks on a link and goes to the next page and simply keeps clicking links at random, going from one page to another page. The resulting PageRank value for a certain page is equal to the relative probability that the surfer visits that page when he would be surfing around to the end of time. The more frequently the page is visited, the higher the importance of the page, the higher the PageRank value will be.

The PageRank of a page is dependent on three factors:

1. Incoming links: the number of pages that link to the page. The more pages that link to a page, the more important this page probably is.
2. Outgoing links: the number of pages that the page is linking to.
3. The PageRank values of the linking pages. That is, pages with a higher importance add more value compared to pages which are less important. Therefore, in order to compute the PageRank of a page you need the ranks from other pages first.

The computation of the PageRank for a web page $u$ can be expressed as:

$$PR(u) = \sum_{v \in L(u)} \frac{PR(v)}{C(v)} \tag{1}$$

The PageRank of a page $u$ depends on the PageRank values for each page $v$ in the set of incoming links $L(u)$ to $u$, divided by the number of outgoing links $C(v)$ of page $v$.

There are two problems with the algorithm as described. The first problem is that when there is a web page with only incoming and no outgoing links, there is no way for the surfer to get out, the surfer will be trapped. This is called a dangling page or sink [17]. The second problem is a variation on the first problem. Instead of a single page with no outgoing links there is a closely connected set of pages, a strongly connected component, which only links to pages inside the component and not to any pages in the rest of the network. These problems have been solved by adding a variable $\alpha$, called the *damping factor*. The damping factor represents the chance that the surfer will jump to a randomly chosen page. A jump can be interpreted as a random surfer clicking links and browsing the web, but at a certain moment getting bored and randomly jumping to a (possibly unrelated) other page somewhere on the internet. In the original paper by Brin and Page the proposed damping factor is set to $\alpha = 0.85$ which means that it takes about 5 clicks for a random surfer to get bored [5].

When the damping factor is incorporated in equation 1, the PageRank for a page $u$ can be expressed as:

$$PR(u) = \alpha \left(\frac{1}{N}\right) + (1-\alpha) \sum_{v \in L(u)} \frac{PR(v)}{C(v)} \tag{2}$$

The question is if this algorithm can be applied to the symptom network and if so, if it will yield any sensible results. To answer this question we will have to look at the properties of the symptom graph and how this influences the results of PageRank. A desirable result would be to single out bridge symptoms to investigate their crucial role in the network as described by Cramer at al. [8].

### 3.2 The effect of the damping factor

The importance of the recursive effect of some symptoms cannot be understated. The fear of fear example as given above clearly shows the recursiveness and intensification of such a symptom [16]. In the symptom graph this translates to a symptom with an link to itself. What would be the effect of self-referencing on the PageRank of a symptom? In the actual implementation of PageRank the links from a page to itself are ignored and the PageRank value stays the same. For the fear of fear example however, the importance of the symptom increases because it reinforces itself. The fear of fear does not dissolve in thin air.

In case a symptom only has outgoing links or the only outgoing link is to itself, the symptom can be considered a sink. The sink problem has been described previously and has been overcome by the damping factor. Due to the damping factor, after a certain number of steps, a random jump will be done in the network. What would such a random jump mean in the symptom network? Does a jump relate to the chance of a person getting a random other symptom? If a jump occurs according to the chance that a symptom will surface, then a sensible value for $\alpha$ needs to be calculated. Mental disorders are influenced by numerous environmental factors like family, childhood, culture and work but also biological factors like genes. Even from person to person and from moment to moment the influences on the probability of getting a mental disorder can change. Therefore, giving a statistically reliable chance of getting a symptom or mental disorder is extremely hard if not impossible.

A second way to look at the random jumps in the symptom graph is by treating them as a kind of error term. It is impossible to measure every single thing on earth to calculate a damping factor. Therefore there is always an unknown predictor (randomness) in the network which has not been modelled. Hence, it is impossible to predict the occurrence of symptom. However, it might be possible to estimate the amount of randomness. Let us assume that there is a large set of time bound data about the occurrence of symptoms, that is, data about when symptoms occur and how they relate to other symptoms over time. If such a dataset exists, a estimation can be made to what the value of the damping factor, in our case the amount of randomness, should be.

In consequence, the damping factor can maybe be estimated to a value which will make sense, as it is derived from empirical data.

### 3.3 Network type 1

Let us assume, for now, that there is no sensible damping factor available and therefore we will set it to $\alpha = 1$. A damping factor of 1 means that there is no chance of randomly jumping to another symptom. If the network is time bound and could include several symptoms at a certain moment, but not at the other, then the damping factor of 1 would make sense as there is no chance to have other mental disorders at that moment in time. Although, since there is no time bound data available we will consider the non-time bound case. When looking at the symptom network of figure 3, a large component is visible. The large component covers roughly half of all symptoms. Apart from the large component there are some small clusters of symptoms and a couple of isolated symptoms. If the isolated symptoms were web pages and the random surfer would start on an isolated page, there is no way of getting out. The random jumps cannot happen because the damping factor is 1. The isolated symptoms end up not having a reliable PageRank. Does is matter that these isolated symptoms have an unreliable PageRank value? Recall that we are looking for important bridge symptoms. The isolated symptoms are not bridging between other symptoms, they can never be important bridge symptoms in network type 1. Therefore the value of isolated symptoms does not matter.

Clusters of densely coupled symptoms are visible within the large component. These clusters are generally called *Tightly Knit Communities* (TKC) and they hamper the ability to assign reliable PageRank values to web pages [15]. The TKCs in the symptom network are there due to the way the network is build up. All symptoms for a mental disorder are linked with all other symptoms of the disorder. When the random surfer would enter such a cluster, there is a good chance that quite a lot of the symptoms would be visited. This actually makes sense as the the symptoms linked to a disorder probably have a higher chance of surfacing then symptoms of another symptom. As an example, when a person is suffering from an elevated blood pressure, there is a higher risk of suffering from dizziness, confusion and eventually seizure than that the risk of developing intense sexually arousing fantasies is. As said before, TKCs are considered a hindrance when calculating PageRank values. Therefore, using PageRank on a type 1 symptom network presumably yields inferior results when looking for bridge symptoms.

### 3.4 Network type 2

The second network type, as described in section 2.2, is a directed network instead of an undirected network. However, the difference is not just in directedness but also that network type 2 is not a symmetric graph. In other words, there is not an equal an opposite incoming link for every outgoing link. Therefore, network type 2 has more resemblance with a web graph than type 1 and application of PageRank would presumably give bridge symptoms a higher PageRank than with type 1.

Although there is no actual network type 2 available some assumptions can be made. TKCs might be in the network as well because it seems plausible that symptoms for a mental disorder are likely to be influencing each other because in the end they are symptoms for the same disorder. On the other hand, some symptoms might be only surfacing in certain conditions while with network type 1 they are connected to all other symptoms of the same mental disorder. Therefore, when a type 2 symptom network is created the occurrence and effect of the TKCs should be analyzed carefully.

## 4 Hyperlink-induced Topic Search (HITS)

Around the same time PageRank was introduced, Jon Kleinberg proposed a similar web page ranking algorithm named Hyperlink-induced Topic Search (HITS) [13]. Instead of indexing nodes solely based on links and their weights, HITS is based on the notion of having *hubs* and *authorities* on the web that point to each other. *Hubs* point to authorities and can be described as ‘the person who knows someone that can explain it’ and *authorities* are the actual sources of information who *may* know zero or more hubs. Currently, the HITS algorithm is used at the Ask.com[2] search engine and at the ‘Who To Follow’ service at Twitter [10].

The notion of having hubs and authorities can be best described by the example of a person who wants to buy a car [19]. If that person is going to buy a car based on the best model of that year within a specific price range, a possible query for a web search could be: ‘best car model 2014 under 40k’. Finding results in the web graph solely based on the occurrence of these words on every page will likely return low quality results. Some unrelated pages might incidentally contain many words mentioned in the query whereas highly related pages may go unnoticed because they use slightly different words such as ‘vehicle’ or ‘automobile’ instead of car. However, what does happen a lot at some web-pages, such as blogs or review sites, is the activity of linking to every source mentioned within that page. In this case, a blog containing a page covering the best car model of 2014 costing less than 40k would be a hub that links to all specific car model manufacturer pages that have been reviewed; the authoritative pages.

HITS works by assigning every page a hub and an authority score. Evaluating these hubs and authority scores at query time yield good quality results as we can sort on both authority or hub score, depending on if we want specific or clustered information. The basic rules that come with these scores are that a good hub increases the authority score of the pages it points to and a good authority increases the hub score of a page it is pointed from. Looking at Equation (3) and (4) we can see the recursive procedure that represents these rules [13].

$$authority(p) = \sum_{q:(q,p)\in E} hub(q) \tag{3}$$

$$hub(p) = \sum_{q:(p,q)\in E} authority(q) \tag{4}$$

Computing both scores for every nodes happens at query time by running the following basic procedure: within the graph being processed, we first perform a text-based search to find all nodes that are directly relevant for the query. The result of this search is called the root set. Following this, we construct the base set that extends the root set by also including all nodes that are linked to or linked from in the root set. The focused subgraph, which is the set of nodes and links within the base set, will function as the query data for HITS. The recursive hub and authority score procedure will be run and once HITS has converged or its maximum number of iterations have been reached, we know which symptoms act as hubs and which symptoms as authorities.

The main advantage of HITS over PageRank in a traditional web graph setting can be found in the fact that HITS is query based instead of the topology based query-independent PageRank [9] algorithm. As shown in the previous paragraph, query dependency implies that for every different query, HITS will yield different results optimized for that query. On the contrary PageRank will first filter on results and then sort on pre-computed PageRank of a result’s page. However, query dependency also implies that for every query, the results have to be computed at query time. This may significantly decrease query speed, especially in a dynamic environment as the web where (small) changes occur all the time.

Other disadvantages of HITS in a web graph environment are especially focused at misleading or spamming the algorithm by adding outgoing links to your own page; thus increasing your page’s hub score. Furthermore, picking the right base set deems to be very important because of the so called ‘topic drift’ problem of HITS. An example of topic drift is given by Cohn and Chang [7]: a search term such as ‘jaguar’ could result in ranking a football team as top result because of a newspaper writing many articles about them.

Considering HITS in the context of a DMS-IV symptom network containing about 500 nodes and a small-world component covering roughly 50% of these nodes, both query time execution and spamming

[2]Ask: www.ask.com

should not be considered as major disadvantages because of the limited network size and a base set that will be likely to cover the entire network.

Before evaluating HITS for both network type 1 and 2, a small extension to both network types should be made. Up and until now, we have considered that every node in both network types only contains the name of the symptom. However, because HITS is query and text-search based, the DSM-IV description of the symptom should also be included. If this would not be the case, constructing a reliable root and base set is nearly impossible.

### 4.1 Network type 1

As stated earlier, we consider the undirected network type 1 as being bi-directional for all connections it contains. If we evaluate the primary procedure within HITS which is computing hubs and authority scores, we can conclude from Equation (4) and (3) that in a bi-directional graph, both scores will be equal. This implies that no advantages will be present over a regular text based search of symptom descriptions. Hence, HITS will not yield interesting results for psychopathologists when it is applied on currently generated and validated symptom networks.

### 4.2 Network type 2

Continuing our thought experiment of applying HITS on symptom network type 2, we can see some possibly interesting results. Before going into these results, let us imagine being a psychopathologist researcher that is researching patients who are alcoholics and have sleep problems. If we would search in the symptom network using HITS, we would first construct a query that could contain the key-words 'alcohol' and 'trouble sleeping'. HITS would construct the root and base set using a text-based search and subsequently compute hubs and authority scores for all nodes in the base set. In the domain of psychopathology, these hub and authority scores could mean the following: a high hub score indicates a symptom being a bridge symptom that could lead to many other symptoms. On the other hand, authority scores indicate symptoms that are not likely to cause other symptoms.

Possibly interesting results of applying HITS to symptom network type 2 are mainly to be found in 'real-time' usage by psychologists. Having a symptom network type 2 at disposal for every patient enables a psychologist to assess a patient's condition using quantitative data. Based on a couple of key words on a patients condition, many possible outcomes of new states can be gathered within a few moments. Furthermore, a psychopathologist can also glance at the seriousness of a patients condition by indexing the HITS result on hub score; if a query returns a currently present symptom as a top hub, then it important to know where this could lead.

## 5 Stochastic Approach for Link-Structure Analysis

The third and last algorithm discussed is *Stochastic Approach for Link-Structure Analysis* (SALSA) and is a web ranking algorithm inspired by PageRank and HITS [14, 15]. SALSA computes both authorities and hub scores as HITS does, and like PageRank, the scores are obtained from the scores of other pages. The SALSA is very similar to HITS but has focused on a major problem of HITS, namely the bonding between hubs and authorities when it comes to the TKC effect.

A tightly knit community is a small but highly connected set of nodes. In the case of HITS these nodes will get a high score although they may be unauthoritative on the topic. The mutual reinforcement that these nodes have on each other are vulnerable for the TKC effect and may end up in uncalled-for high positions. The presence of TKCs in a subgraph is known to have negative results on the finding of authorities in the network when HITS is applied [15].

As HITS, SALSA is a query dependent algorithm. It computes an authority score which estimates how relevant the page is according to query. The hub score estimates whether the page contains valuable links to other authorities. Instead of using these values directly, SALSA weighs them according to the indegree and outdegree of the nodes.

When assessing SALSA for its applicability, we have to overcome the same problem as in the case of HITS. Because SALSA is query dependent, having just the name of a symptom will not be enough. The originally proposed symptom network needs to be extended with additional qualitative description for every symptom in the network as is proposed in the section on HITS 4.

The effect of SALSA on network type 1 will presumably yield better results than PageRank and HITS because of the focus on the TKC effect. However, we suspect that even better results will be obtained with network type 2 due to the increased resemblance with web pages in the internet when compared to network type 1.

## 6 Conclusion

In this paper we have built upon recent research that has shifted the psychopathological measurement of comorbidity from direct latent variable analysis to symptom network analysis. We considered an undirected (converted to bi-directional) symptom network (type 1) that was the result of previous research, and a more theoretical directed symptom network (type 2) that was built using several assumptions.

When applying PageRank, HITS and SALSA to these two network types, the first and foremost result that emerges is that in order for link analysis to be of any added value, a directed network type 2 should be preferred over the undirected network type 1; in the case of PageRank, there is the issue of random jumps that may not occur since that would not be realistic. When evaluating HITS, an undirected network would not even yield sensible results since both hub and authority scores for the symptoms would be equal. Also, the inability to correctly handle self-loops is unfortunate since this is a very interesting research topic.

If we consider pushing a query to network type 2, HITS can be used to gather insight in the current state and possible future state(s) based on a couple of keywords about a patient's current condition. PageRank and HITS could be used in combination to verify each other's results and provide additional quantitative grounds for treatment.

SALSA has the ability to overcome the tightly knit community (TKC) effect and is therefore proposed as an improvement over HITS. Especially in the psychopathology domain where TKCs exist in the small-world cluster of the initially introduced DSM-IV symptom network type 1 in figure 3, SALSA might offer valuable improvements over HITS.

We conclude that HITS and SALSA have the potential of offering interesting practical results which may actually assist a psychopathologist in discovering important bridge symptoms. Furthermore, HITS and SALSA may be able to support psychopathologists in validating their findings. We do not expect PageRank on itself to yield interesting results although it might be used to support HITS and SALSA findings.

## 7 Discussion and future

Continuing research on applying link analysis algorithms on symptom networks may be very useful since link analysis algorithms are able to provide insight into the importance of individual nodes in the network. However, in order to apply these link analysis algorithms, the first and foremost step now is to develop the ability to construct and validate symptom networks type 2 efficiently. Without having a validated symptom network type 2, link analysis will not be as useful as described in this paper. Even worse, when applied to symptom networks type 1, link analysis may return unreliable results because of the simple fact that they are developed to handle web graphs that are ultimately directed at all times.

After having created empirically validated symptom networks type 2, a next interesting step might be the creation of weighted networks where the weight could indicate the likelihood of the source symptom causing the destination symptom to emerge. The weight information could be included in, for instance, HITS to obtain more accurate hub and authority scores.

## Acknowledgements

The authors wish to thank Frank Blaauw for his role as advisor during the time span of this research. Without his enthusiastic contributions,

we would have gone mental. Also, the peer reviewers provided us with a lot of useful feedback that helped during the rewriting process.

## REFERENCES

[1] U. Albert, G. Rosso, G. Maina, and F. Bogetto. Impact of anxiety disorder comorbidity on quality of life in euthymic bipolar disorder patients: differences between bipolar i and ii subtypes. *Journal of affective disorders*, 105(1):297–303, 2008.

[2] A. P. Association, A. P. Association, et al. *Diagnostic and Statistical Manual-Text Revision (DSM-IV-TRim, 2000)*. American Psychiatric Association, 2000.

[3] D. Borsboom and A. O. Cramer. Network analysis: An integrative approach to the structure of psychopathology. *Annual review of clinical psychology*, 9:91–121, 2013.

[4] D. Borsboom, A. O. Cramer, V. D. Schmittmann, S. Epskamp, and L. J. Waldorp. The small world of psychopathology. *PloS one*, 6(11):e27407, 2011.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[6] L. F. Bringmann, N. Vissers, M. Wichers, N. Geschwind, P. Kuppens, F. Peeters, D. Borsboom, and F. Tuerlinckx. A network approach to psychopathology: New insights into clinical longitudinal data. *PloS one*, 8(4):e60188, 2013.

[7] D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In *ICML*, pages 167–174. Citeseer, 2000.

[8] A. O. Cramer, L. J. Waldorp, H. L. van der Maas, and D. Borsboom. Comorbidity: A network perspective. *Behavioral and Brain Sciences*, 33(2-3):137–150, 2010.

[9] M. Franceschet. Pagerank: Standing on the shoulders of giants. *Communications of the ACM*, 54(6):92–101, 2011.

[10] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 505–514, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[11] R. C. Kessler, P. Berglund, O. Demler, R. Jin, K. R. Merikangas, and E. E. Walters. Lifetime prevalence and age-of-onset distributions of dsm-iv disorders in the national comorbidity survey replication. *Archives of general psychiatry*, 62(6):593–602, 2005.

[12] R. C. Kessler, W. T. Chiu, O. Demler, and E. E. Walters. Prevalence, severity, and comorbidity of 12-month dsm-iv disorders in the national comorbidity survey replication. *Archives of general psychiatry*, 62(6):617–627, 2005.

[13] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[14] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Computer Networks*, 33(1):387–401, 2000.

[15] R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Transactions on Information Systems (TOIS)*, 19(2):131–160, 2001.

[16] R. J. McNally. Anxiety sensitivity and panic disorder. *Biological psychiatry*, 52(10):938–946, 2002.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[18] R. A. Schoevers, D. Deeg, W. Van Tilburg, and A. Beekman. Depression and generalized anxiety disorder: co-occurrence and longitudinal patterns in elderly patients. *The American journal of geriatric psychiatry*, 13(1):31–39, 2005.

[19] R. Tanase and R. Radu, 2009.

[20] D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

# A Survey on Design Patterns and Software Quality Attributes

Wytse Visser, Pascal Bouwers

**Abstract**—Design patterns are commonly used to solve software design problems. In the process of selecting design patterns, the effect on software quality attributes is often overlooked. The main reason for this is that the common believe is that the negative influence on software quality is negligible. The software development field lacks knowledge on the real effect of design patterns on software quality and the results of different researches on this subject are contradicting. Our research is focused on the design patterns of the Gang of Four (GoF) from the viewpoint of software developers. Additional information was needed to improve the selection process of design patterns. Improving this process will increase the software quality.
We have performed an empirical research by conducting a survey among software developers from open source projects. Given our results, we have verified the research results from the existing papers and generated additional new information on this subject.
We have given more clarity on the rationale behind using design patterns, by validating that they are mainly used because they provide working solutions to recurring problems and enable easy communication. In addition, we have provided information on the popularity of individual design patterns and the importance of quality attributes. Lastly, we confirmed that design patterns can have a negative impact on quality attributes. However, our results suggest a less negative impact than other studies.

**Index Terms**—Gang of Four, design patterns, software quality attributes.

## 1 Introduction

By nature, software engineering is an empirical subject, although in practice it is often based on ”expert opinion” instead of empirical evidence [14]. Several authors over the past two decades have observed this and proposed a more empirical approach. Whitley noted that in software engineering research, the development of system-building techniques is often advocated without evidence [20]. Kitchenham et al. proposed to adopt evidence-based software engineering in order to ”provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software” [15]. In [18], Sjøberg et al. envision a more empirical approach as well and outline how to address current challenges on this matter.

An example in need of more empirical evidence is the design of object oriented systems based on software design patterns. The large amount of literature about design patterns suggests that they are valued by software developers. However, most of this literature seems to be based on experience. The best known book on design patterns is by Gamma et al. [8], which has the same problem that it lacks evidence on where and when to use a certain patterns. Like many other software engineering practices, it seems to be based on experience rather than on empirical evidence.

Design patterns are commonly used to solve software design problems. In the process of selecting design patterns, the effect on software quality attributes is often overlooked. The main reason for this is that the common believe is that the negative influence on software quality is negligible. The software development field lacks knowledge on the real effect of design patterns on software quality and the results of different researches on this subject are contradicting [2].

The goal of this paper is to provide software developers with more information regarding the relation between design patterns and the software quality attributes they influence. This will improve the selection process of design patterns and thereby increase software quality.

- *Wytse Visser is a MSc. Computing Science student at the University of Groningen, E-mail: w.m.visser.1@student.rug.nl.*
- *Pascal Bouwers is a MSc. Computing Science student at the University of Groningen, E-mail: p.bouwers@student.rug.nl.*

### 1.1 Patterns

Patterns have first been documented by Christopher Alexander in 1977, when he described reusable architectural proposals for producing quality designs [1]. The field of computing science adopted this idea and created pattern languages aimed at their own types of patterns. An important type of patterns are architectural patterns, which aims at providing solutions to recurring problems related to software architectures. Architectural patterns have been documented in a wide variety of catalogs, such as those by Buschmann et al. [6] or Fowler [7] and are now a respectable part of software engineering research and practice.

Design patterns are another type of patterns, introduced by Gamma et al. [8] (nicknamed the ”Gang of Four” or GoF — as we will refer to them). Design patterns are narrower and of a lower abstraction level than architectural patterns. However, in some cases it can be hard to define a specific pattern as either an architectural or a design pattern [4]. According to the GoF, ”these patterns solve specific design problems and make object-oriented designs more flexible, elgant, and ultimately reusable” [8]. The use of design patterns is also considered as an advantage as it eases communication by providing a common vocabulary between software engineers [17].

Software Architects need to make trade-offs between strengths and liabilities of specific architectural patterns to meet the desired quality attributes of a system. For instance, Harrison and Avgeriou have evaluated a number of architectural patterns against different quality attributes [10]. For design patterns, the tradeoffs between quality attributes is far less known. This study tries to bridge this gap by providing the same level of knowledge on design patterns as exists for architectural patterns. We do this by means of a survey among software developers, as is explained in more detail later.

### 1.2 Quality Attributes

In this paper we will use the quality attributes from the ISO/IEC 9126 standard [11], which is a standard for the evaluation of software quality. The quality attribute model from the ISO/IEC 9126 standard is hierarchical. The first level contains six characteristics. Each of these six characteristics is divided into sub-characteristics on the second level. We decided to only use the first level set of characteristics, which consists of six quality attributes described by the standard as follows.

**Functionality:** A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

**Reliability:** A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

**Usability:** A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

**Efficiency:** A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

**Maintainability:** A set of attributes that bear on the effort needed to make specified modifications.

**Portability:** A set of attributes that bear on the ability of software to be transferred from one environment to another.

## 2 Related Work

Ever since the GoF introduced the concept of design patterns in 1994, many articles have been written on the use of them. In this section we give an overview of related work, which comprises secondary studies and surveys. This related work focuses on the relationship between patterns and software quality attributes.

Zhang and Budgen performed a systematic literature review on design patterns [21]. In this mapping study, they searched for papers published until 2009 with a strong empirical focus on the effectiveness of design patterns. The results of this study show that only minimal empirical evaluation of design patterns has been performed. However, they did find studies that claim evidence on advantages of using design patterns. These advantages include an improvement of the effectiveness of the communication between developers and maintainers and improvement of software quality. The latter, however, is also negatively influenced according to other studies they found.

A similar mapping study has been performed by Ampatzoglou et al. in [2]. This study is performed from a wide perspective, focusing on many aspects of design pattern research, both empirical and non-empirical. They identify ambivalent results in the effect of design patterns on particular quality attributes. Only a limited number of studies exists for the relationship between many individual quality attributes and specific design patterns.

Khomh and Guéhéneuc performed a survey among software engineers in [12]. They assessed the impact of all 23 design patterns on a number of software quality attributes. From this study can be concluded that some design patterns negatively influence several quality attributes and therefore should be used with caution. A complete discussion of that study is given in [13].

## 3 Methodology

We performed a survey among open source software developers to gather quantitative data on design patterns from the perspective of software engineers. Our research consists of a number of steps:

1. Research questions definition
2. Survey design
3. Data collection
4. Data processing

These steps have been performed sequentially. In this section, we elaborate on their outcome.

### 3.1 Research questions

This study mainly focuses on design patterns and software quality attributes. We tried to gain additional knowledge in the relationship between these concepts. In addition, we wanted more insight in the importance of each individual quality attribute from the perspective of software engineers. Lastly, our research targets to dissect the process behind selecting design patterns by gathering more practical information on the rationale behind this process.

The goal of our research is to contribute information on the impact of design patterns on software quality attributes to the field of software engineering. To achieve this goal, we defined a number of research questions. In order to keep the size of this study within reasonable limits, we performed our investigation on the most popular design patterns and the top-level quality attributes from the ISO/IEC 9126 standard. The research questions are as follows:

**RQ$_1$:** *What is the rationale behind using design patterns?*

**RQ$_2$:** *What is the importance of each quality attribute?*

**RQ$_3$:** *How often is each design pattern used?*

**RQ$_4$:** *How do design patterns influence software quality attributes?*

### 3.2 Survey Definition

The second step of our method consisted of designing the questionnaire. We used an online questionnaire and designed it conform to [3], which provides useful guidelines for online questionnaires. The final version of the survey can be downloaded from `http://goo.gl/ClbFcZ`. Initially, each respondent was asked about his professional / open source experience with software development in general. This could be answered with either `1`, `2`, `3`, `4`, or `5 or more` years. In the second question, we asked about the respondent's experience with design patterns in particular. We suggested 5 answer options ranging from `Not experienced at all` to `Very experienced`. By using the answers to these questions, we could filter results based on experience and knowledge.

#### 3.2.1 Pattern usage rationale

To give an answer to the first research question, our survey contained a question on the main reasons behind using a design pattern. We selected a number of possible answers of which multiple could be selected, based on usage reasons that were given in different articles ([19, 5]). These predefined answer options of which multiple could be selected consisted of: `A) No specific reason`, `B) To improve certain software quality attributes`, `C) To solve a problem with a proven solution`, `D) To gain working knowledge of a new pattern`, and `E) Easy communication with other developers and/or designers` In addition, it was possible to specify a custom reason.

#### 3.2.2 Quality attribute importance

The quality attributes were assessed in order to provide an answer to the second research question. This was done by rating the importance of the top-level quality attributes from the ISO/IEC 9126 standard in the respondent's software development practice. Five options could be selected: `A) Not Important at All`, `B) Somewhat Important`, `C) Important`, `D) Very Important`, and `E) Not Sure`. The first four answers are ranged from low to high. The last answer gave the respondents the possibility to indicate a lack of knowledge on a quality attribute.

#### 3.2.3 Design pattern usage

The original design patterns catalog by the GoF contains 23 different design patterns [8]. Researching all these patterns exceeds the size limits for this study, hence we limited the number of patterns. To provide the most valuable data, we selected the 10 most used design patterns. This selection was based on data from a quantitative study by Hahsler [9], in which he investigated 988 open source projects. By analyzing log messages of the version control systems, he was able to deduct data on design pattern occurrence in these projects. According to this study, the 10 most popular design patterns are Singleton, Builder, Adapter, Observer, Visitor, Factory Method, Facade, Composite, Bridge, and Decorator.

For each of these design patterns, we asked the respondents to indicate their familiarity with that pattern. Additional questions were asked to respondents indicating at least any familiarity with the pattern. These additional questions were not asked to respondents without familiarity, but they were redirected to the next design pattern instead. To provide an answer for the third research question, the respondents were asked to indicate how often they used the pattern. The five-point scale used as answer options ranged from `Never` to `Very Often`.

#### 3.2.4 Impact on quality attributes

Similar to the process performed in [12], a six-point Likert scale was used to evaluate the quality attributes [16]. We asked the respondents to determine the impact of the use of each design pattern on our set of quality attributes. This to generate data to answer the fourth research question. The given choices are `A) Very Negative`, `B) Negative`, `C) Not Significant`, `D) Positive`, `E) Very Positive`, and `F) Not Sure`. This question was only asked to respondents that indicated having at least any familiarity of the pattern.

### 3.3 Data collection

In the last week of February 2014 we posted our questionnaire to a number of different developer mailing lists of open source projects. The selection of mailing lists contained the Spring framework, OpenOffice, OpenStack, and several others. After a 7 day period we collected the results of the questionnaire.

We received 36 answers from which we took a selection of 26 based on two criteria. We only selected software engineers that have 3 or more years of experience in professional or open source software development, and who indicated that their design pattern experience was 'somewhat experienced' or better.

The question on how often a certain pattern is used was not asked to respondents without familiarity of that pattern. Therefore, we had to edit the results in order to get reliable results. If a respondent was unfamiliar with a certain pattern, we considered this as that respondent never having used that pattern. This is reasonable, because if a respondent has ever used a pattern, he or she would at least be somewhat familiar with it.

## 4 Result analysis

In this section we analyze the results from the questionnaire and answer the four research questions. First, we present the rationale behind using design patterns. Next, the importance of each quality attribute is discussed, followed by an analysis on the usage of each individual pattern. This section closes with an analysis of the relationship between patterns and quality attributes.

### 4.1 Pattern usage rationale

The first research question is about the rationale behind using design patterns. In the survey we asked the respondents why they were using design patterns. They could select multiple options from five predefined answers and were able to give an open answer as well. The results can be seen in Table 1. The table shows that the majority of the respondents uses design patterns to solve problems with proven solutions and to ease communication with other developers and/or designers. About half the respondents use design patterns to increase the software quality attributes. A small portion of the respondents indicated they use a certain design pattern to gain knowledge of that pattern. Two participants noted that design patterns often emerge naturally out of the software's design, noting that it is good to take the last step and call them by their name.

### 4.2 Important quality attributes

Ranking the importance of each quality attribute is the second research question. In the questionnaire we asked to label the importance of each of the six quality attributes. They could be labeled on a 4-point Likert scale ranging from `Not Important at All` to `Very Important`. The scale runs from 0 to 3 and the result with the calculated averages for each quality attribute can be seen in Figure 1. The graph shows that functionality is considered the most important quality attribute, 75% of the respondents labeled functionality

Table 1. Reasons to use design patterns

| Answer | Rate (%) |
|---|---|
| No specific reason | 0.0 |
| To improve certain software quality attributes | 46.15 |
| To solve a problem with a proven solution | 69.23 |
| To gain working knowledge of a new pattern | 19.23 |
| Easy communication with other developers and/or designers | 65.38 |
| Other | 7.69 |

as `Very Important`. Usability, maintainability, and reliability are shown to be a little less important than functionality, the majority voted between `Important` and `Very Important`. Another step down in importance is efficiency, where the average vote is `Important`. Clearly at the bottom is portability, where the majority voted between `Somewhat Important` and `Important`.

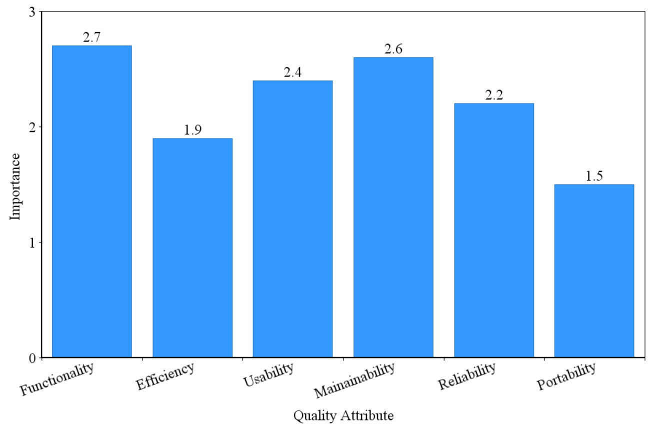


Fig. 1. Quality attribute importance

### 4.3 Pattern usage

$RQ_3$ focuses on how often each design pattern is used. In the survey we asked for each design pattern how often the respondent uses them. They could choose between 5 predefined answers on a Likert scale, ranging from `Never` to `Very Often`. The graph in Figure 2 shows the average usage of the answers, per pattern. The usage ranges from 1 to 5. The most often used design pattern is the Observer, the majority of the respondents said they use this pattern `Often` or `Very Often`. The Factory Method and Facade pattern are mostly used `Sometimes`, which is average. All the other patterns are very close to 2.5 usage, which translates to between `Rarely` and `Sometimes`. The exception to this is the Bridge pattern, where the majority of the respondents indicated they were not familiair with the pattern, which explains why it is almost never used.

### 4.4 The pattern quality attribute relationship

In this section, we will give the results regarding the fourth research question. First, we give an overall overview of the results. Subsequently, we describe the results in more detail by providing raw information.

The respondents evaluated the relation between design patterns and quality attributes in a scale that ranges from $-2$ to 2. Based on the results, we calculated the average result for each quality attribute, using the data given for all design patterns. The results are listed in Table 2. Most striking in these results is that all values are positive, which indicates that the respondents consider design patterns on the average as

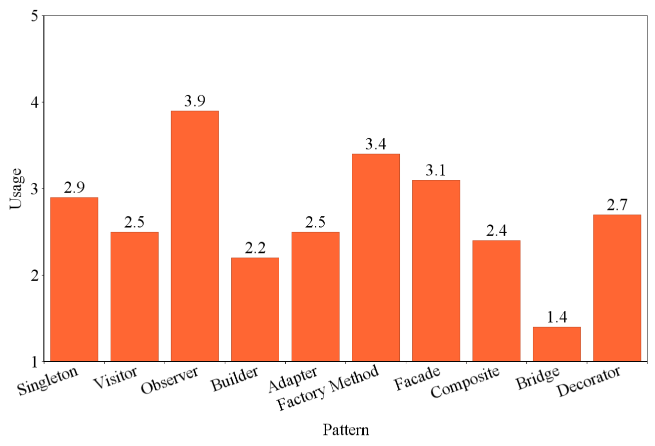


Fig. 2. Pattern usage

having a positive impact on quality attributes. On the other hand, there is a difference in the magnitude of that impact. For instance, functionality has a high impact value in comparison with the others. On the contrary, the values for efficiency and portability are rather low.

From the perspective of design patterns, we performed a similar approach by calculating the average impact for all quality attributes per design pattern. The results are listed in Table 3. Most striking is the negative average impact value of the Bridge pattern. All other impacts are positive, yet none is larger than 1.0. This means that on the average, the quality attributes seem to be positively affected, but not greatly.

The raw end results for the impact of design patterns on quality attributes are listed in Table 3. The values represent the impact of the design pattern in that row on the quality attribute in that column. We will take a detailed look on the results, both from the viewpoint of patterns as well as the quality attributes.

### 4.4.1 Pattern details

**Singleton:** According to the respondents, the Singleton pattern positively affects all quality attributes. However, the positive impact on maintainability is comparatively low (+0.20).

**Visitor:** The results show that using the Visitor pattern highly increases maintainability (+1.00) at the cost of efficiency (-0.18). All other quality attributes are positively affected, yet not as much as maintainability.

**Observer:** For the Observer pattern, all impacts are rated relatively high, with a peak in functionality (+1.64).

**Builder:** The impacts for the Builder pattern have a large difference. The efficiency is negatively affected (-0.17), while functionality and maintainability are affected highly positive (+1.33 and +1.00 respectively).

Table 2. Average impact per quality attribute

| Quality attribute | Average impact ($-2$ to $+2$) |
|---|---|
| Functionality | 1.07 |
| Efficiency | 0.22 |
| Usability | 0.54 |
| Maintainability | 0.54 |
| Reliability | 0.38 |
| Portability | 0.26 |

**Adapter:** The Adapter pattern is considered as having a significantly positive impact on functionality (+1.00), but does not increase the other quality attributes to that extent. It neither impacts efficiency positively, nor negatively.

**Factory Method:** All impact scales for the Factory Method pattern are positive. Especially, functionality (+1.29) and maintainability (+0.86) gain a large increase. Usability and portability are just slightly positively affected, having impact values of +0.34 and +0.17 respectively.

**Facade:** The variation in the impact of the Facade pattern on the studied quality attributes is high. Functionality (+1.67), usability (+1.00), and maintainability (+1.33) are affected vastly positive. On the contrary, efficiency is influenced slightly negative (-0.17). When using this pattern, there is increase in portability. This increase, however, is just slight (+0.20).

**Composite:** The highest advantage in using the Composite pattern lies in the functionality quality attribute (+1.20). The values for the remaining quality attributes are increased as well, but not as much. The efficiency, reliability and portability quality attributes are all increased by +0.40 when selecting this pattern.

**Bridge:** From the results can be concluded that the Bridge pattern does not yield any positivivity with regard to quality attributes. Even worse, maintainability and reliability are both influenced negatively (-0.50 for both). The remaining quality attributes do not seem to be impacted at all by this pattern.

**Decorator:** For the Decorator pattern the results are varied as well. The most striking positive aspect is the increase in functionality by +1.17. The costs for this pattern are in the maintainability and reliability quality attributes, which are decreased with -0.50 and -0.33 respectively. Efficiency (+0.33) and usability (+0.50) are positively affected. This pattern does not have any impact on the portability of the system in which it is used.

### 4.4.2 Quality attribute details

**Functionality:** The overall impact on functionality is largely positive. The impact on functionality is smaller than +1.00 for only three patterns: the Singleton pattern (+0.73), the Visitor pattern (+0.64) and the Bridge pattern (0.00).

**Efficiency:** The efficiency quality attribute is affected positively, neutrally, and negatively by different patterns. The patterns providing the most positive results are Singleton (+0.73) and Observer (+0.64). The neutral patterns are Adapter and Bridge. Finally, the Visitor pattern (-0.18) and the Builder and Facade patterns (both -0.17) all negatively influence efficiency.

**Usability:** Our results show that the usability of a system is not hugely affected by design patterns. The impact values range from 0.00 (Bridge pattern) to 1.00 (Facade pattern).

**Maintainability:** Some patterns have a rather positive impact on maintainability, such as the Facade pattern (+1.33) and the Visitor and Builder patterns (both +1.00). However, maintainability is affected with -0.50 if either the Bridge or Decorator patterns are used.

**Reliability:** The reliability of system is neither significantly increased nor significantly decreased by design patterns. By selecting the Bridge or Decorator pattern, the reliability is decreased however. For the first, this value is -0.50 and for the latter -0.33.

**Portability:** Portability is barely affected by the choice of design patterns. According to the respondents, the highest increase can be obtained by selecting the Observer pattern (+0.45). Both the Bridge and Decorator patterns have no influence at all on portability.

Table 3. Effect of design patterns on quality attributes

| | Functionality | Efficiency | Usability | Maintainability | Reliability | Portability | *Average* |
|---|---|---|---|---|---|---|---|
| Singleton | 0.73 | 0.73 | 0.80 | 0.20 | 0.67 | 0.43 | *0.59* |
| Visitor | 0.64 | -0.18 | 0.30 | 1.00 | 0.70 | 0.40 | *0.48* |
| Observer | 1.64 | 0.64 | 0.82 | 0.91 | 0.64 | 0.45 | *0.85* |
| Builder | 1.33 | -0.17 | 0.57 | 1.00 | 0.67 | 0.33 | *0.62* |
| Adapter | 1.00 | 0.00 | 0.50 | 0.33 | 0.33 | 0.20 | *0.39* |
| Factory Method | 1.29 | 0.57 | 0.34 | 0.86 | 0.57 | 0.17 | *0.63* |
| Facade | 1.67 | -0.17 | 1.00 | 1.33 | 0.67 | 0.20 | *0.78* |
| Composite | 1.20 | 0.40 | 0.60 | 0.80 | 0.40 | 0.40 | *0.63* |
| Bridge | 0.00 | 0.00 | 0.00 | -0.50 | -0.50 | 0.00 | *-0.17* |
| Decorator | 1.17 | 0.33 | 0.50 | -0.50 | -0.33 | 0.00 | *0.20* |

## 5 Discussion

The respondents to our survey considered most of the studied design patterns to have a positive impact on most of the quality attributes. However, the results suggest that different quality attributes are far from considered being equally impacted. Moreover, a few patterns have a negative influence on a small number of quality attributes.

In this section, we strive to give an explanation to the results for a selection of the most-used design patterns, displayed in Figure 2. For each pattern, we give a short description of its intent and describe how it affects certain quality attributes.

### 5.1 Observer

The Observer pattern provides the functionality for observers to be automatically notified of state changes in so called ”subject” classes. The observers in their turn can perform update functionality based on this state change. The results in Table 3 show us that the Observer pattern gives a positive impact on all quality attributes.

The main increase in functionality can be assigned to the fact that this pattern provides a notification mechanism between different classes. The improved maintainability is also not a surprise, since the pattern provides loose coupling between subjects and observers. This makes it easy to add or modify observers. A reason for an increase in efficiency could be that observers do not need to constantly check for updates with their subjects, but notifications are pushed to them. However, we had expected this impact to be much smaller. In a similar fashion, we can state that this pattern makes sure that observers are up to date and thus increasing reliability. Having up to date observers results in a consistent user-experience, which is probably why usability is considered to be affected positively.

### 5.2 Factory Method

The Factory Method is a design pattern to implement a concept called factories. This concept lets developers create objects without actually specifying the exact class of that object. According to the GoF the purpose of this pattern is to ”define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses.” [8].
The Factory Method increases the ease to maintain the code by decreasing duplicate code and increasing abstraction. This should lead to a positive impact on maintainability, which the survey shows according to Table 3. The pattern has close to no impact on usability and portability. This is not strange, considering the Factory Method does not directly influence them. The Factory Method increases efficiency by having a centralized lifetime management for objects, this is to ensure consistent behavior of the application, which increases reliability. The survey shows that this pattern has a moderately positive impact on both efficiency and reliability. We can imagine that this pattern allows for the creation of certain classes, so that it increases functionality. We are however not sure why it is between positive and very positive.

### 5.3 Facade

The Facade pattern is used to bundle multiple different interfaces from a subsystem into a single unified interface. This enables client classes to use functionality of a subsystem without knowing the exact details of the inner workings. The facade knows which class implements which functionality and how they can be invoked, but does not expose this information.

The huge increase in functionality is surprising, because a facade in itself is only an entry point to classes providing functionality (hence its name). We can understand why the maintainability of a system is improved, because using a facade significantly loosens the coupling between classes. For instance, the GoF suggest to use this pattern as entry point for layers in a layered system [8]. It is not clear on how using this pattern would increase usability. We had expected portability to be much higher than the results suggest, because of the loose coupling this pattern provides. In theory, having a looser coupled system makes it easier to reuse parts in another system. The results show a small negative impact on efficiency. This is due to the extra level of abstraction that this pattern puts between client and the implementation classes.

### 5.4 Decorator

The Decorator pattern is a design pattern that dynamically or statically allows behavior to be added to a specific object. It does this without having any influence of any other object from the same class. The survey answers in Table 3 show that this pattern has a clear positive impact on functionality. This makes sense considering the pattern allows to extend the functionality of a certain object, sometimes even at run-time, which allows for new flexible functionality. The main effect of the pattern is extending functionality.
The GoF says the following on the Decorator: ”A design that uses Decorator often results in systems composed of lots of little objects that all look alike. Although these systems are easy to customize by those who understand them, they can be hard to learn and debug.” [8]. According to the GoF the pattern makes a system hard to learn, this negatively influences maintainability. Our survey also shows a negative impact on maintainability. The book also says that the pattern often makes a system hard to debug, which implies the system will have more bugs, which negatively impacts reliabity. Our survey also shows a negative impact on reliability.
According to the survey, the pattern has no influence on portability. This is not strange because the Decorator doesn’t directly influence anything portability related. Efficiency and usability are positively impacted by the pattern. They are slightly positively impacted because new functionality can be combined at run-time, without having to create seperate classes for this.

## 6 Threats to validity

In this section, we discuss threats that can possibly invalidate the results of this study. The results of our study are based on a questionnaire among software developers. Therefore, they might not be considered as fully objective as the opinions of the respondents are never fully factual. Since we sent an open survey invitation to different open source groups, we might have attracted more respondents with a positive bias towards design patterns. This can lead to results which are more positive than they should be.

## 7 Conclusions

By performing this survey, we gained additional insight in design patterns and quality attributes. We provided information on the decision-making process of selecting design patterns, as listed in Table 1. As we expected, design patterns are chosen because they provide known solutions to recurring problems and provide easy communication between developers. The result of the importance of different quality attributes are shown in Figure 1 and show that functionality and maintainability are considered more important than portability and efficiency. We also provided data on the popularity of individual design patterns in Figure 2. Finally, we found evidence that design patterns can have a negative impact on quality attributes. However, our study gives more positive than negative results and on the average, design patterns are considered as having a positive impact on quality attributes. Moreover, the negative influence of design patterns is less significant than in [12] or [13].

We have investigated only a subset of all design patterns. To improve the knowledge on design patterns, future studies might focus on the remaining patterns.

## References

[1] C. Alexander, S. Ishikawa, and M. Silverstein. A pattern language: Towns, buildings, construction (center for environmental structure series). 1977.

[2] A. Ampatzoglou, S. Charalampidou, and I. Stamelos. Research state of the art on gof design patterns: A mapping study. *Journal of Systems and Software*, 86(7):1945–1964, 2013.

[3] D. Andrews, B. Nonnecke, and J. Preece. Conducting research on the internet:: Online survey design, development and implementation guidelines. 2007.

[4] P. Avgeriou and U. Zdun. Architectural patterns revisited–a pattern language. 2005.

[5] K. Beck, R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, L. Dominick, and F. Paulisch. Industrial experience with design patterns. In *Proceedings of the 18th international conference on Software engineering*, pages 103–114. IEEE Computer Society, 1996.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, P. Sommerlad, and M. Stal. Pattern-oriented software architecture, volume 1: A system of patterns, 1996.

[7] M. Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[9] M. Hahsler. A quantitative study of the adoption of design patterns by open source software developers. *Free/Open Source Software Development*, pages 103–123, 2005.

[10] N. B. Harrison and P. Avgeriou. Leveraging architecture patterns to satisfy quality attributes. In *Software Architecture*, pages 263–270. Springer, 2007.

[11] ISO/IEC 9126. Software engineering-product quality-part 1: Quality model. *Geneva, Switzerland: International Organization for Standardization*, 2001.

[12] F. Khomh and Y.-G. Guéhéneuc. Do design patterns impact software quality positively? In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, pages 274–278. IEEE, 2008.

[13] F. Khomh, Y.-G. Guéhéneuc, and P. Team. An empirical study of design patterns and software quality. *GEODES–Research Group on Open, Distributed Systems, Experimental Software Engineering, University of Montreal*, 2008.

[14] B. Kitchenham, D. Budgen, P. Brereton, M. Turner, S. Charters, and S. Linkman. Large-scale software engineering questions-expert opinion or empirical evidence? *Software, IET*, 1(5):161–171, 2007.

[15] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 273–281. IEEE, 2004.

[16] R. Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[17] S. McConnell. *Code complete*. " O'Reilly Media, Inc.", 2004.

[18] D. I. Sjoberg, T. Dyba, and M. Jorgensen. The future of empirical methods in software engineering research. In *Future of Software Engineering, 2007. FOSE'07*, pages 358–378. IEEE, 2007.

[19] P. Wendorff. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In *Software Maintenance and Reengineering, 2001. Fifth European Conference on*, pages 77–84. IEEE, 2001.

[20] K. N. Whitley. Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages & Computing*, 8(1):109–142, 1997.

[21] C. Zhang and D. Budgen. What do we know about the effectiveness of software design patterns? *Software Engineering, IEEE Transactions on*, 38(5):1213–1231, 2012.

# In depth with Technical Debt management

Djurre de Boer Joost Koehoorn

**Abstract**—Technical debt is the result of writing code that is complex, hard to extend and maintain, poorly documented or poorly tested. Unlike with financial debt, it is often not known what the impact of gaining technical debt has on a software project, since its need for change or impact on future requirements may not be known. Several methods have been developed to gain insight in the technical debt of software, in order to make guided decisions on how and when debt has to be paid. This helps software teams in finding a balance between implementing new requirements and keeping the code at high quality. Some of these methods may be used in every software project, whereas more extensive methods require continuous evaluation and parameterization. In this paper, we first give an introduction of the technical debt metaphor, after which three methods for technical debt management are discussed: basic static analysis metrics, risk management and the SQALE method. For all these methods we discuss their pros and cons and how easily they can be integrated into the development process of a software project. Based on these aspects, a comparison between the methods is established to find the optimum technique depending on the project and team.

**Index Terms**—Technical debt, static analysis, management, software risk, software maintenance, analysis, SQALE.

## 1 Introduction

Software is becoming more and more complex over the years and new features have to be implemented according to tight deadlines. Because of this, developers often have to take shortcuts and write code without adhering to company standards and architectural guidelines, or without updating documentation and writing new tests. These symptoms introduce *technical debt*, a metaphor coined in 1992 by Cunningham [4] to describe a situation in which long-term code quality is traded for short-term gain. Over time this causes problems with maintaining the software, as understanding of the code-base along with its flexibility to change decreases.

At the heart of this problem is scarcity of resources. Investing in the quality of code takes valuable time while not generating short-time revenue, and being mainly invisible to customers. Project managers and engineers frequently disagree about when and to what extend, if at all, these investments have to be made, even though the influence it has on long-term software sustainability is substantial [2].

To overcome such project management issues, methods have been developed to identify and measure technical debt. Such methods aim to help in keeping track of potentially problematic areas in software, along with the ability to evaluate whether it is worth addressing an issue or that it is more cost-effective to leave it as-is.

Because several approaches for technical debt management have been developed, it may be hard to find a suitable solution for a specific environment. For instance, team size, code-base size and experience of developers all play an important role in how effective a solution would be. The aim of this paper is thus to establish a comparison of three proposed methods we think are most valuable, and assess their effectiveness and cost with respect to the three mentioned environment aspects. These recommendations are based on the characteristics of the methods and data from earlier research, no case studies have been done.

In section 2, we first explain what technical debt is, along with a list of causes for technical debt and ways to identify it. This is followed by an overview of several methods regarding technical debt management which have been proposed in earlier research, in section 3. We then focus on what method is best fitted for which development environment, and formulate recommendations on managing technical debt for a variety of development teams and processes in section 4.

- *Djurre de Boer is a MSc. Computing Science student at University of Groningen, E-mail: djurredenboer@gmail.com.*
- *Joost Koehoorn is a MSc. Computing Science student at University of Groningen, E-mail: j.koehoorn.1@student.rug.nl.*

## 2 Technical Debt Metaphor

Similarly to financial debt, technical debt incurs interest payments in the form of future costs to further updates to the code-base. Thoughtless architectural decisions and quick and dirty fixes add to this debt, as does outdated documentation and incomplete tests. Building onto badly written code counts as interest on the debt and may eventually cause significant issues with maintenance [4]. Like financial debt, going into technical debt is sometimes necessary. Taking the time to fix the issues can be seen as debt repayment in order to avoid additional interest payments as time passes.

There are also differences with financial debt, as sometimes it is not clear up front when a project goes into debt. Decisions made regarding software architecture may prove to have been based on improper knowledge of the problem, or changes in requirements may be incompatible with the software in its current state. Similarly, if it is assumed that a particular module does not need to be modified in the future, deciding to not update its documentation will save time without any adverse consequences. However, it is often not known if a module will ever need modifications or if unexpected defects in the module may cause maintenance tasks which are unaccounted for. In 2011, Seaman and Guo [16] stated that this makes the problem primarily a matter of risk management and of making informed decisions about which delayed tasks need to be addressed, and when they have to be accomplished.

It is common that a software project incurs some debt during the development process, as this allows for higher productivity and gaining a better understanding of the problem [16]. It is however important that sooner or later the debt is repaid, otherwise it compromises system architecture or code quality. Besides an extra burden on the developers, it also complicates project management as decisions have to be made on how resources are spent on debt repayment. It is therefore helpful to project management when technical debt can be identified, measured and monitored so that informed decisions can be made.

### 2.1 Causes of technical debt

Technical debt may sometimes be intentionally incurred for tactical or strategical reasons, e.g. to meet a deadline or to increase the understanding of the problem. There are however also cases where a software system incurs unintentional debt, for instance because of lack of attention or missing test cases [12]. When system requirements are not well defined —causing inaccurate architectural decisions to be made— or when a problem is approached in the wrong way, perhaps by using unsuited tools, a project goes into a lot of debt from the beginning, unintentionally. This form of technical debt is the hardest to manage, as it is not observed until it may be too late to fix the issues without discarding a lot of earlier work. Fowler [6] has established

| | Reckless | Prudent |
|---|---|---|
| Deliberate | *“We don’t have time for design”* | *“We must ship now and deal with the consequences”* |
| Inadvertent | *“What’s layering?”* | *“Now we know how we should have done it”* |

Fig. 1. Categorization of technical debt causes, as given by Fowler [6].

a categorization as shown in Figure 1 to separate issues arising from recklessness —being unaware of issues or ignorant regarding them— from strategic decisions, e.g. to meet a deadline.

Besides code related debt, lacking succinct documentation and tests are also forms of debt, as are unclear requirements. Automated testing is an important aspect in verifying that the system is operating correctly, however if the tests have not been designed and written thoroughly, they may give an incorrect impression of the correctness of the software. During later refactoring this is an opportunity for subtle issues to sneak in, as the tests do not cover all use cases and hence harming the value and reliability of the system.

## 2.2 Properties of technical debt

Brown et al. [2] have established a, possibly incomplete, set of properties of technical debt, which can help to define the concept and characterizing types of technical debt:

- *Visibility.* Invisible debt, or known to only one person, causes significant problems to others who eventually have to pay for it.
- *Value.* The value is the economic difference between the system as it is and the system in an ideal state for the assumed environment.
- *Present value.* The present value of a system is the cost of the technical debt value subtracted from the potential value of its features. This includes a time-to-impact and uncertainty calculation, to create a more realistic estimation of the present value.
- *Debt accretion.* The total debt may not scale additively, but super-additively. Taking on too much debt may lead the system in an unrepairable state.
- *Environment.* In software engineering projects, debt is relative to a given or assumed environment.
- *Origin of debt.* A distinction between intentional and unintentional debt should be made.
- *Impact of debt.* The locality of elements that need to be updated to repay a debt, e.g. localized or widely scattered.

These properties have been indicative in earlier research to determine well-defined metrics to objectively quantify technical debt. Risks can be quantified by assigning values to above properties, so that risks with high impact and high probability of manifesting themselves may be dealt with accordingly, as to avoid having to pay high interest for them. Managing technical debt thus becomes a matter of risk management, for which a workflow is further outlined in section 3.2.

## 3 Methods

The technical debt metaphor has been helpful in finding ways to identify, measure and manage potential problems in software systems. Agile software practices such as Scrum, XP (Extreme Programming) and TDD (Test Driven Development) have become increasingly popular for software development and their goal of allowing for quick reaction to changing demands have helped software projects to become more modularized and expandable. While these paradigms promote writing better maintainable code, they do not directly address managing technical debt proactively. For this reason, more involved metrics and workflows have been studied to be able to make informed decisions on how technical debt is managed. In this section we first show how using static analysis of a code-base provides information on the code quality. We then focus on two methods we believe are most applicable to technical debt management, and most valuable for acquiring insight into the state of a software project. A risk management solution is discussed first, followed by the discussion of SQALE, a rich system of software metrics and indicators to help reduce technical debt.

Another approach we do not discuss would be to introduce periodic refactoring moments where e.g. once every two months one week is dedicated to refactoring. We think such an approach would indeed help in reducing technical debt, but it does not offer the opportunity for informed decision making and actively tracking of technical debt, which we think are vital aspects to actual management. The same can be said for standards such as ISO 9126 [8] which only define ways to assess software quality, not provide means to find the issues that are most important to solve first.

### 3.1 Static analysis methods

Although not directly designed with technical debt management in mind, static analysis tools may provide valuable information about the quality and complexity of code-bases. Such tools are nowadays widely available for a large range of programming languages. Some are even directly available in IDEs (Integrated Development Environments), and can be used to detect certain forms of *code smells*, which relate to technical debt in that they are a deviation from good programming practices, potentially causing maintainability issues. A common example of code smell is code duplication, which may be automatically identified by static code analysis. Another example of automated quality assessment is the lines of code (LOC) metric which determines software size, or cyclomatic complexity and fan-in/fan-out to give an understanding of the complexity, coupling, cohesion and testability of software.

Detecting code smells requires calibrated threshold values to correctly filter actual code smells from harmless code. *Derived smells* can be determined by combining quality metrics with threshold values and boolean operators. Marinescu et al. [9] proposed a set of criteria including thresholds to automatically detect such code smells. For instance, they proposed the following expression which determines if a class has too many responsibilities:

$$\text{WMC} > 47 \quad \wedge \quad \text{ATFD} > 5 \quad \wedge \quad \text{TCC} < 0.33$$

Here, WMC is the weighted methods per class, ATFD the number of foreign class data accesses and TCC the tight class cohesion. A problem with this approach is that defining the individual components such as WMC may pose problematic, since a weight factor has to be determined for each method. Additionally, the metrics and provided thresholds may need to be optimized for more accurate usage in a particular domain [7].

Another more involved metric is the C.R.A.P. index, as defined by Savoia in 2007 [15] and short for Change Risk Analysis and Predictions. Not only does it rely on code metrics, also test coverage is employed, assuming that well covered code imposes a smaller risk in future maintenance tasks. The C.R.A.P. index of a method $m$ is defined as follows:

$$C.R.A.P.(m) = CYCL(m)^2 \cdot (1 - \tfrac{1}{100}COV(m))^3 + CYCL(m)$$

| Technical debt item | Interest probability | | Interest amount | | Benefit | | Cost | | Decision | Total Cost |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 0.8 | $\times$ | 6 | = | 4.8 | $>$ | 4 | $\Rightarrow$ | Pay off | 4 |
| E | 0.5 | $\times$ | 4 | = | 2 | $<$ | 10 | $\Rightarrow$ | Delay | 4 |
| B | 0.5 | $\times$ | 13 | = | 6.5 | $>$ | 6 | $\Rightarrow$ | Pay off | 10 |
| A | 0.2 | $\times$ | 11 | = | 2.2 | $>$ | 2 | $\Rightarrow$ | Pay off | 12 |
| D | Low | | Low | | Low | | Low | | Delay | Low |

Table 1. Example of a debt list used for risk management by Seaman and Guo [16], slightly reordered and annotated for better readability. The items are sorted by interest probability and considered from top to bottom. Here, the maximum cost dedicated to debt payoff is 12, hence item $D$ is not evaluated.

In this formula, CYCL is the cyclomatic complexity of method $m$, and COV the test coverage in percentage. For methods with 100% test coverage the C.R.A.P. index measures just the cyclomatic complexity, whereas untested code is assigned an index of just over the squared cyclomatic complexity. This means that the index not only promotes high coverage, but also low complexity is preferred. A suggested threshold for when refactoring is in order is when the C.R.A.P. index exceeds 30 [14]. Some consider this threshold to be too high, as a method with an already fairly high cyclomatic complexity of 10 only needs 42% test coverage to not exceed the threshold value of 30 [14].

Even though static analysis may prove useful for limited insight into the state of a code-base, they only bring to the attention possible problematic code fragments. This is certainly a useful measure but touches only a rather small aspect of technical debt, e.g. there is no indicator for negligent documentation and such tools do not have knowledge over past experiences and future requirements. Moreover, many metrics determined by static analysis are only applicable to object oriented code —the concept of classes, methods, dependencies etcetera is specific to object oriented languages— and do not translate to e.g. functional languages.

In an effort to obtain more reliable results regarding defect detection, the history of files may be analyzed together with metrics obtained by static analysis. Zimmermann et al. [17] have established a method to acquire a mapping between bug reports and source code locations, by utilizing history data from versioning systems such as CVS. Their approach was successfully employed on historical data of the Eclipse project, which was reused in research by Moser et al. [13]. Moser et al. found that such "process metrics", extracted from file history and bug activities, provide a better insight in code defects than metrics obtained with static analysis do. The reasoning for this is that code metrics indicate the cognitive effort to understand the code, not the correctness of code. Frequent changes, especially to complex files, are more likely to cause defects than complicated code in itself.

Although employing historical data is an interesting approach to reason about correctness of code, we argue that it is less useful in managing technical debt, as defects only play a small role in technical debt. For better maintainable code, having a low cognitive effort is vital, in contrast to what Moser et al. [13] found for defect detection. The next method we discuss is a risk management approach, in which historical data is also used in establishing reliable estimates for e.g. risk probabilities.

## 3.2 Risk management

To remedy the shortcomings of static analysis, risk analysis may be employed. This form of analysis attempts to describe the identified risks, in order for stakeholders to decide what actions to take. A common set of attributes as given by Fairley [5] and Chapman et al. [3] is used to describe risk:

- *Class.* The type of the risk.
- *Cause.* The events that lead to the risk.
- *Scope.* The range in which the risk is considered.
- *Impact.* The severity of potential loss of the risk.
- *Probability.* The likelihood that the risk gets instantiated.
- *Valid time period.* The time frame in which the risk is valid.

Note that these attributes match some of the properties of technical debt as mentioned in section 2.2. Seaman and Guo use these attributes to draft a workflow for strategic management of technical debt [16]. For the probability and impact attribute, a simple scale consisting of low, medium and high is used. Even with such a coarse scale, assigning values is still highly subjective, e.g. dependent on experience. Better estimations of these attributes can be achieved using historical data that match current project characteristics. The approach taken by Seaman and Guo starts from a rough estimation of the attributes, then refines the estimation based on historical data, after which technical debt items are prioritized based on their probability and impact.

A list of technical debt items is reviewed and updated after each release, when items can be added and deleted from the list. This may happen at different points in the release process, depending on the type of debt:

- *Testing debt:* when the decision is made to skip tests or reviews for a particular release or module.
- *Defect debt:* when a defect is found but not addressed in the current release.
- *Documentation debt:* when a module is modified but its documentation is not updated
- *Design debt:* when e.g. code smell or violations of the system architecture are identified.

The estimated effort to address a debt item is based on historical data so that a more accurate estimation is achieved. The same is true for the probability of a debt item, coupled with a time frame in which the debt is estimated to be incurred. Establishing interest amount — the amount of extra work that will be incurred for not completing a debt item— is more complicated, a more detailed example of how this is accomplished is given in the original paper [16].

When a component is slated for updates in an upcoming release, all debt items regarding this component are selected from the debt list. These items then have to be reevaluated, as planned work may render debt items irrelevant or otherwise influence their effort. Then for the remaining items with high interest probability and interest amount, more precise numerical estimates have to be assigned. Table 1 is an example that shows the extracted items and assigned estimates. The benefit column as shown in the table is obtained by multiplying interest probability with interest amount, yielding the estimated benefit of paying off the debt. If this benefit outweighs the costs, then the item is handled and its cost is taken into account. This is done until a predetermined limit of costs has been reached, after which any remaining debt items are delayed.

## 3.3 SQALE

Another method specifically developed for managing technical debt is SQALE, an acronym for Software Quality Assessment based on Lifecycle Expectations. It was proposed by Letouzey and Ilkiewicz [11]

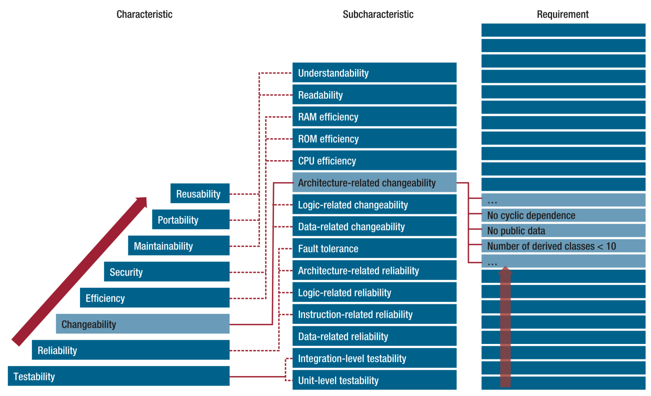


Fig. 2. Inheritance in quality model: characteristics have sub-characteristics, which are further subdivided into requirements [11].

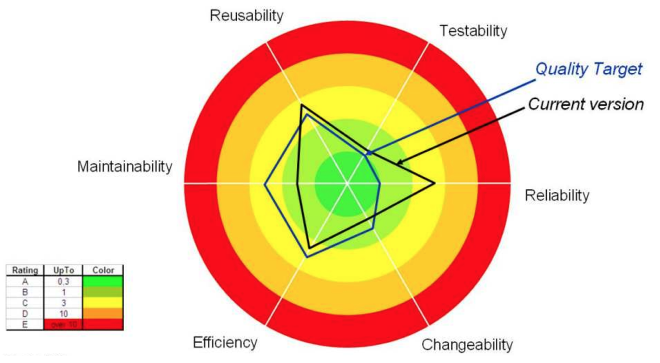
from the French company *inspearit* as a generic method, independent of programming language and analysis tools. SQALE allows to clearly define what creates technical debt and how to correctly estimate this debt, based on code quality metrics from code analysis methods.

SQALE consists of a quality model and analysis model. The quality model is the part of the SQALE process where the non-functional requirements related to code quality are gathered and classified into several categories. The quality model is composed out of three hierarchical levels: the characteristics, sub-characteristics and the source code requirements. The characteristics correspond with those defined in the ISO 9126 standard [8] with the addition of the characteristic "reuseability" [10]. The requirements describe the lowest level and are usually dependent on software context and programming language, to make SQALE a generic solution.

The characteristics of the quality model are stacked on top of each other to form a pyramid. This is done because characteristics depend on each other, e.g. without testability a component can hardly be made reliable as it is untestable. This is shown in Figure 2, where the pyramid of characteristics is shown on the left. The pyramid also establishes an order in which issues have to be taken on, e.g. it would nto make sense to spend time on fixing code duplication issues and then later having to completely rewrite the code to accomplish testability.

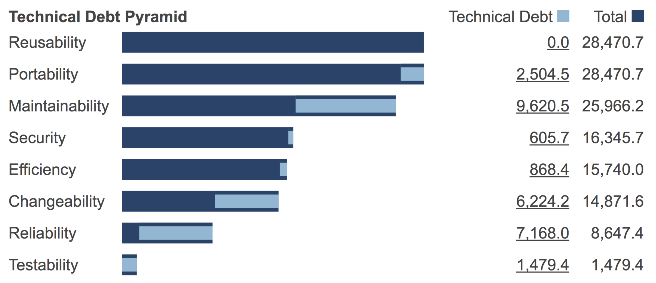
The analysis model defines on one hand rules that are used to normalize the code metrics, as outputted by the quality model. On the other hand, rules for aggregating these normalized values are defined. Normalizing the results from code analysis tools is accomplished by transforming them into remediation and non-remediation costs, by using a remediation factor for simple conversions or a remediation function to allow for more refined conversions. Having accurate remediation factors/functions for every requirement is essential, as the cost of remediation widely depends on the nature of an issue. The non-remediation costs are used for estimating what the penalty of not resolving an issue are, to allow for selectively addressing violations while not discarding issues that impose a high cost when they are not taken care of.

As the violations are categorized into characteristics, the state of a single characteristic may be summarized by taking the sum of all remediation costs of the requirements belonging to the characteristic. This produces eight SQALE indices, one for each characteristic. A global index which describes the total amount of technical debt in a system is obtained by adding all remediation costs from all requirements together, which results in the SQALE Quality Index (SQI). As these are absolute values and as such depend on code-base size, indices may not directly be compared between different projects. The definition of index densities aims to make this possible, by dividing the absolute indices by e.g. the number of lines of code to acquire a relative measure independent of code size. Such relative indices also allow for monitoring and evaluating the evolution of the technical debt as the code-base expands.



| Rating | UpTo | Color |
|---|---|---|
| A | 0,3 | 🟩 |
| B | 1 | 🟩 |
| C | 3 | 🟨 |
| D | 10 | 🟧 |
| E | over 10 | 🟥 |

Fig. 3. An example of the Kiviat indicator, which shows the current states of characteristics and their targets [10].



| Technical Debt Pyramid | | Technical Debt | Total |
|---|---|---|---|
| Reusability | | 0.0 | 28,470.7 |
| Portability | | 2,504.5 | 28,470.7 |
| Maintainability | | 9,620.5 | 25,966.2 |
| Security | | 605.7 | 16,345.7 |
| Efficiency | | 868.4 | 15,740.0 |
| Changeability | | 6,224.2 | 14,871.6 |
| Reliability | | 7,168.0 | 8,647.4 |
| Testability | | 1,479.4 | 1,479.4 |

Fig. 4. SQALE Pyramid indicator as provided by SonarQube [1]. It shows how the technical debt is distributed over all characteristics.

Another cost that is calculated is the business impact index, which is the impact violations are estimated to have on the development and maintenance. This index is calculated by summing the non-remediation costs from all violations, which quantifies the business impact and represents the business perspective of non-quality [11].

Since the calculated indices are just numeric values they may be hard to fully comprehend without visualizations. SQALE therefore has the notion of indicators, which are visual representations of the indices. An example of such an indicator is called the Kiviat indicator, as shown in Figure 3. The black line shows the current rating of a characteristic, whereas the target is indicated with a blue line to give a quick impression of where improvements have to be made.

Several tools, to name Squore[1] and SonarQube[2], have been developed which implement the SQALE method, which provide e.g. a dashboard for quick insights into the technical debt, sensible default configurations for easy integration and easy to grasp indicators. Figure 4 shows the technical debt pyramid as provided by SonarQube [1], which shows at a glance in which characteristics most of the debt is located. In contrast to the Kiviat this pyramid does not reveal objectives, however its indication of debt in a certain characteristic is more precise. Figure 5 is another example of how the issues are distributed over the characteristics, sub-characteristics and requirements, so that at a glance it can be seen which requirement violations are responsible for most of the debt.

## 4 Comparison of Methods

Now that we have discussed several technical debt management methods, this section focuses on comparing them regarding the effectiveness for various development environments with respect to team size, experience of developers and code-base size. Furthermore, the effort required to start using the methods and the cost of using and maintaining them once team members and managers are acquainted with them is discussed.

[1]http://www.squoring.com
[2]http://www.sonarqube.org/

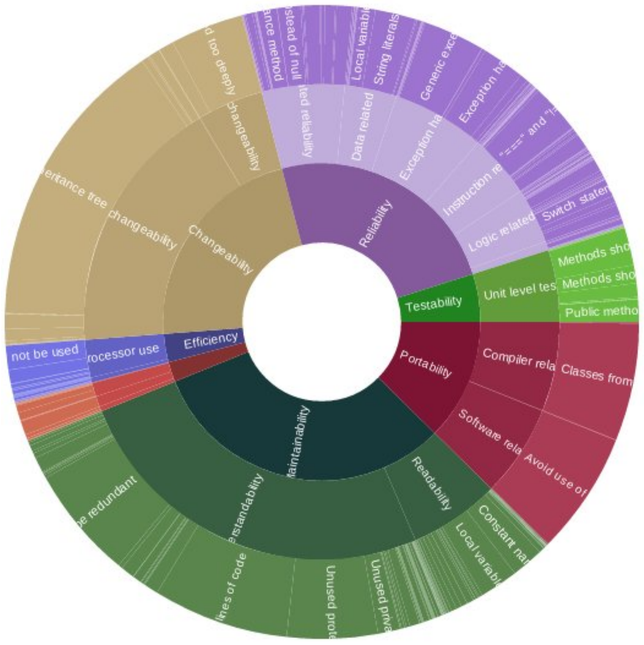


Fig. 5. Sunburst diagram of SonarQube, giving a clear representation of the distribution of the hierarchy [1].

## 4.1 Effectiveness

Basic static analysis results, such as cyclomatic complexity and fan-in/fan-out metrics, along with clone detection and metrics such as the C.R.A.P. index give some insight in the complexity and adherence to good programming practices, but they are only relevant for code-related technical debt. For large-scale, long-term projects only relying on static analysis results is inadequate, however for small companies who are utilizing Agile programming methodologies —to enforce project management with quickly changing demands in mind— it may be sufficient. We consider it inadequate for large-scale projects and organizations because static analysis in itself does not provide any means of actual management of debt, it only shows certain issues in the code-base.

Moreover, interpreting statical analysis results may be problematic for less experienced developers, as they provide no guidance on where to start addressing issues. When working on a project as a team, even when it is small, having insight in the state of a project at a glance by means of easy-to-understand graphs is essential in communicating and reasoning about the issues. This is what SQALE provides on top of the results from statical analysis, and because of the pyramid it becomes apparent which issues are most important to take care of early on.

The effectiveness of the risk management method mostly depends on being able to identify risks and the accuracy of the estimations made for them, so having historical data available is essential. This makes it less attractive for startups or for unexperienced teams, as they do not have this information. When actively and accurately updated with properly estimated risks, it does however offer a valuable way of actually reasoning about debt and making informed decisions about it, which we argue is more accurate than what is offered by SQALE. Risk management is more direct in this sense, the order in which issues have to be taken on is clearly defined by the interest probability assigned to the items.

SQALE also incorporates a part of the approach of risk management, as most attributes by Fairley [5] and Chapman et al. [3] as discussed in section 3.2 can be translated to SQALE. The *class* corresponds with the characteristic, the *cause* with the requirement that is violated, and the *impact* from applying the non-remediation cost. Risk *probability* and *valid time period* is not available in SQALE, so a specific violation can not be diminished when its probability of imposing interest payments is low. Note that in risk management, the interest probability is the first-order indicator of which issues have to be addressed first, so SQALE is missing an important measure in differentiating between two debt items from the same requirement. Risk management is more direct in this sense, the order in which issues have to be taken on is clearly defined by the probability of the debt.

## 4.2 Initial setup effort

Starting to incorporate statical analysis is arguably the easiest way to get started with technical debt management. Initial setup requires to define the metrics to be used and their threshold values, once this has been done it can easily be integrated e.g. in continuous integration processes.

To start using the risk analysis workflow as proposed by Seaman and Guo [16], an initial debt list has to be composed, and preferably historic data is also available or gathered. The debt list may either be fully reconstructed by analyzing the existing code, documentation and tests, or it may be decided to start collecting debt items from then on. The latter takes of course less effort, but then also the effectiveness decreases as already introduced debt with high interest probability and amount is not visual and may therefore cause problems with the effectiveness of the method.

Regarding SQALE, first of all the requirements to be put into characteristics have to be established, although this has to be done only once. Furthermore, remediation and non-remediation costs have to be determined for all requirements. By employing an existing tool such as SonarQube or Squore however, these initial efforts may be avoided as such tools provide sensible defaults based on findings from earlier research and applications in large organizations [11]. This makes SQALE easy to setup, as proven defaults are already in place. This does not mean that the method is restrictive, as changes can be made to e.g. the requirements or the remediation factors to make them applicable to anyones needs.

SQALE thus requires the least amount of effort to start using, but learning to make effective use of it also has to be taken into account. The basics of SQALE can be seen as convenient measures on top of metrics from static analysis methods, but their interpretation needs to be uniformly understood by the whole team. Everyone needs to be aware of which indicators are used to reason about debt and how these are used in planning the project. Letouzey and Ilkiewicz propose specific trainings for different roles, e.g. a one-day training for experts participating in workshops and a 45-minute awareness session for top managers [11].

## 4.3 Cost of maintaining

Once the initial setup has been performed, the time necessary to maintain the management method should not outweigh the benefits, otherwise employing the technical debt management method becomes obstructive. Analyzing static analysis results can be fully incorporated in e.g. a continuous integration setup, or on check-in into version control systems to achieve a fully automated setup. This does not require additional effort from any members of the team, so relying on static analysis is cheap.

As SQALE also aims to be an automated process in terms of acquiring measurements [11], once costs have been defined and fine-tuned it is not necessary to put manual labor into this. However, to keep the costs accurate and relevant, Letouzey and Ilkiewicz propose and annual review and maintenance of the models [11].

The risk management workflow is the most intensive in terms of maintaining, as the debt list has to stay up-to-date and relevant while also evaluating estimations based on historical data. This is similar to the backlog known in the Scrum methodology, but specifically tailored for technical debt management. If an Agile methodology such as Scrum is already in use, maintaining a debt list and evaluating it may be included during the general planning moments, so that no extra meetings are necessary.

## 5 Conclusion and Future Work

Our aim of this paper was to establish a comparison between technical debt management methods. Such methods have been developed for developers and project managers to make informed decisions on how technical debt can be identified, measured and dealt with for optimum use of resources. We have established that for unexperienced or small teams, static analysis with accurate thresholds may be sufficient for a basic insight in the state of a code-base, but that they do not offer any guidance on what order issues have to be addressed in. Also for larger teams this method becomes insufficient, as it lacks support for documentation related technical debt and more involved data such as estimated interest amount and probability. These estimates have to be available for precise planning of technical debt management, so that the break-even point can be approached. The risk management approach as proposed by Seaman and Guo [16] has been designed with this goal in mind and may be deployed effectively when historical data is available to base estimates on, albeit that it requires the most manual labor which we argue is only beneficial for large teams and projects. The SQALE methodology is also a valuable method for management of technical debt, although it may need some inspiration from risk management proposals in order to include data about architecture and documentation related debt, so that the importance of these aspects is not disregarded.

For future work, we are interested to see how SQALE can be extended so that it focuses more on non-code related aspects of technical debt, such as the state of documentation. We suspect a hybrid approach between risk management as discussed and SQALE to be an even more valuable approach for developers and managers to base their decisions on. This allows for full automation of code-related statistics as is already possible with SQALE, but also for manual input of architecture defects or documentation for even better tracking of technical debt.

## Acknowledgements

We would like to thank all reviewers for the time they took to provide us with valuable input on how to improve this paper, in particular Zengyang Li for sharing his expertise.

## References

[1] Online demo of the SonarQube tool. `http://demo.sonarqube.org/`, 2014. Accessed on: 03-10-2014.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 47–52, New York, NY, USA, 2010. ACM.

[3] C. Chapman, S. Ward, and S. Ward. *Project Risk Management: Processes, Techniques and Insights*. Project management / Wiley. Wiley, 2003.

[4] W. Cunningham. The WyCash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29–30, Dec. 1992.

[5] R. E. Fairley. Risk management for software projects. *IEEE Software*, 11(3):57–67.

[6] M. Fowler. TechnicalDebtQuadrant. `http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html`, 2009. Accessed on: 09-03-2014.

[7] Y. Guo, C. Seaman, N. Zazworka, and F. Shull. Domain-specific tailoring of code smells: An empirical study. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 167–170, New York, NY, USA, 2010. ACM.

[8] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[9] M. Lanza, R. Marinescu, and S. Ducasse. *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[10] J.-L. Letouzey. The SQALE method – Definition document. `http://www.sqale.org/download`, 2012. Accessed on: 09-03-2014.

[11] J.-L. Letouzey and M. Ilkiewicz. Managing technical debt with the SQALE method. *IEEE Software*, 29(6):44–51, 2012.

[12] S. McConnell. 10x software development – Technical debt. `http://www.construx.com/10x_Software_Development/Technical_Debt/`, 2007. Accessed on: 09-03-2014.

[13] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 181–190, New York, NY, USA, 2008. ACM.

[14] A. Savoia. The code C.R.A.P. metric hits the fan. `http://www.artima.com/weblogs/viewpost.jsp?thread=215899`, 2007. Accessed on: 09-03-2014.

[15] A. Savoia. Pardon my French, but this code is C.R.A.P. `http://www.artima.com/weblogs/viewpost.jsp?thread=210575`, 2007. Accessed on: 09-03-2014.

[16] C. B. Seaman and Y. Guo. Measuring and monitoring technical debt. *Advances in Computers*, 82:25–46, 2011.

[17] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, pages 9–, Washington, DC, USA, 2007. IEEE Computer Society.

# Theory and application of second generation connectivity and attribute filters

Herman Schubert, and Jeroen Lanting

**Abstract**—Connected operators are filtering tools that act by merging elementary regions. These operators cannot create new contours nor modify their position. Therefore, they are very attractive for filtering tasks where the contour information has to be preserved. In recent years, several works were devoted to the development of new theories of connectivity, among second generation connected operators, hyper-connectivity, and attribute-space connectivity. These connections extend beyond the standard connectivity, i.e. four- and eight-connectedness, and can model overlap and generalized groupings such as object clusters or partitions. In our work we give a formal background of these methods, and investigate some applications of these new notions of connectivity, where emphasis is given on attribute connected filters, and mask-based connectivity. We show how the attribute filters can be efficiently applied to grayscale images using Max-trees. We apply these methods on numerous examples, such as our galaxy and a volumetric rendering, and our results confirm the usability of these connected filters in their respective solution domain.

**Index Terms**—Mathematical morphology, second-generation connectivity, connectivity class, clustering, partitioning, attribute filtering, dual input max-tree, mask-based connectivity, attribute space.

## 1 Introduction

Connectivity has always been an important notion in the field of image processing. This is even more true for mathematical morphology, because of its topological nature. Connected operators are filtering tools that act by merging elementary regions. As these operators cannot create new contours nor modify their position, they form an interesting set of tools for filtering tasks where the contour information has to be preserved. A very simple connected operator is opening by reconstruction, where the operation is characterized by the reconstruction of a marker image by successive geodesic dilations [12]. This set in motion a research effort into the formal nature of connected filters [15, 11]. Besides opening by reconstruction, area opening [20], attribute-tree filtering [14], and more recently mask-based connectivity [8], fuzzy connectivity [6] and Attribute-space connected filters [22] have been proposed.

A (second-generation) connected operator is an operator that acts on the flat zones of an image, rather than on the level of individual pixels. By connected regions we mean maximal connected regions which is constant in the grayscale case, and foreground or background connected regions (grains) in the binary case. An important new notion of operators is second-generation connected operators. These operators can either cluster components, by merging connected regions, or partition the component into sub-components. A possible generalization which can combine both typologies is mask-based connectivity [8]. With mask-based based connectivity, all previous dependencies on the choice of the preselected operator is eliminated and the ways the image domain can be connected are extended. This allows connectivity to even be based on completely different images, which has, for example, shown its practical use in rubble detection using areal photos after a natural disaster [9].

Another important subcategory of connected filters is attribute-based operators. These operators accept or reject connected components based on corresponding attribute data of the flat zone. Many attribute functions have been proposed, such as the elongation, moments, and compactness [18]. An extension to compute these filters on grayscale images, is threshold decomposition; this can be generalized to max-trees which allows efficient computation of attribute filters [14]. A disadvantage of these filters is that, since they only function on complete components, they cannot deal with overlap. A possible extension to deal with this problem is to use second generation connectivity to partition the components. This can be further improved by using attribute-space based connectivity [22].

This paper is organized as follows. First a theoretical background is given, where the notions of second generation connectivity, and attribute filters are further clarified. Second, the theory is extended to mask-based connectivity, tree-based filtering, and attribute-space-based connectivity. Finally, we give an evaluation of the presented methods, and their solution domain.

## 2 Background

In this section we recall some of the notation and terminology used in mathematical morphology. We primarily restrict ourselves to connectivity on binary images. For a more comprehensive discussion about these concepts the reader may refer to [15]. In mathematical morphology it is common to denote $E$ as the universal set, i.e. the image domain, $X$ as the set of foreground pixels (in the discrete case), and $E \backslash X$ as the background. The power set of $E$, the set of all subsets of $E$, is denoted as $\mathcal{P}(E)$. We use $A \in \mathcal{P}(E)$ and $A \subseteq E$ interchangeably, depending on context.

An operator $\psi$ is said to be a mapping $\psi : \mathcal{P}(E) \to \mathcal{P}(E)$, meaning it operates on subsets of $E$. The operator $\psi$ is said to be:

1. *increasing*: if $X \subseteq Y$ implies that $\psi(X) \subseteq \psi(Y)$.
2. *extensive*: if $X \subseteq \psi(X)$.
3. *anti-extensive*: if $\psi(X) \subseteq X$.
4. *Idempotent*: if $\psi(\psi(X)) = \psi(X)$

Examples of operators are the dilation, and the erosion. When an operator is both increasing and idempotent it is called a *filter*. An (algebraic) *opening* is a filter which is anti-extensive. Respectively, a (algebraic) *closing* is a filter which is extensive. An example of an algebraic opening (resp. closing) is an erosion followed by a dilation (resp. a dilation followed by an erosion).

The elementary regions (pixels) of $E$ are connected by means of a connectivity class $\mathcal{C}$ :

**Definition 1.** *Let E be an arbitrary space. We call a connected class $\mathcal{C}$ a family in $\mathcal{P}(E)$ such that*

$$\emptyset \in \mathcal{C} \text{ and for all } x \in E \to \{x\} \in \mathcal{C} \tag{1}$$

$$\text{for any family } \{A_i\} \subseteq \mathcal{C}, \cap A_i \neq \emptyset \text{ implies } \cup A_i \subseteq \mathcal{C} \tag{2}$$

Alternatively, we say that $\mathcal{C}$ defines a connectivity on $E$. The empty set, and all the singletons $\{x\}$ are connected. Furthermore the union of a family of connected sets is connected, if they have a non-empty

• *Herman Schubert, E-mail: h.robert.schubert@gmail.com.*
• *Jeroen Lanting, E-mail: jeroen.lanting@gmail.com.*

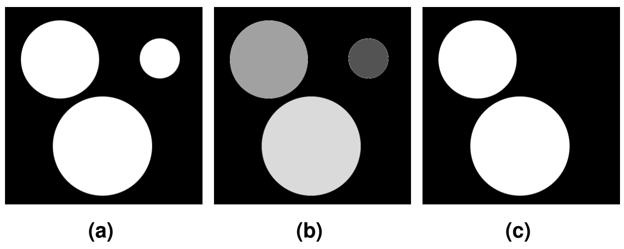


**Fig. 1:** A picture showing 3 foreground objects of different sizes (1a), A gray scale representation of the result of the area filter (1b), and the resulting area opening after thresholding (1c).

intersection. We can explain the behavior of $\mathcal{C}$ by an example. Let the universal set $E$ be the set of all integers, i.e. $E = \mathbb{Z}$ , and let $\mathcal{C}$ define an adjacency relation on $E$ so that every direct neighbor is connected. Then $\{1,2\} \in \mathcal{C}$, and $\{2,3\} \in \mathcal{C}$, and thus rule 2 implies that $\{1,2,3\} \in \mathcal{C}$. However $\{1,3\} \notin \mathcal{C}$, as the numbers 1 and 3 are not direct neighbors in $E$. When $E = \mathbb{Z}^2$ common examples of connectivity classes are the four- and eight- adjacency relationships.

Every set $X \in E$ can be written as a union of connected sets that are pairwise disjoint and of maximal extent, so that $A \subseteq X$, $A \in \mathcal{C}$ of maximal extent implies that no set $B \subseteq X$, $B \in \mathcal{C}$ exists where $A \subset B$. These sets are called connected components, or grains, commonly denoted as $A \Subset \mathcal{C}$. Visually they correspond to the collection of all pixels of a connected region. These connected components can be retrieved by means of a connectivity opening:

$$\Gamma_x(X) = \bigcup \{A_i \in \mathcal{C} | x \in A_i, A_i \subset X\} \tag{3}$$

It retrieves the connected component corresponding to the pixel x. The operator $\Gamma_x$ is anti-extensive, increasing, and idempotent and thus is an algebraic opening. In conclusion, the family of connected openings $\{\Gamma_x | x \in E\}$ uniquely characterize the connectivity class $\mathcal{C}$, and proving certain properties of the connectivity opening [10], proves that the related family is a valid connectivity class.

### 2.1 Connected filters

Connected filters as introduced by Salembier and Serra [15] are defined as a class of filters based on partitions of $E$.

**Definition 2.** *We call a family $P = \{a_i | a_i \subseteq E\}$ a partition of $E$ when the following holds:*

$$a_i \cap a_j = \emptyset \text{ for all } i \neq j$$
$$a_u \neq \emptyset$$
$$\cup a_i = E$$

Thus a partition of $E$ is a set of nonempty subsets of $E$ such that every element $x \in E$ occurs in exactly one subset.

**Definition 3.** *We call a partition $\{a_i\}$ coarser than a partition $\{b_i\}$, or in reverse $\{b_i\}$ finer than $\{a_i\}$, if for every element of the partition $\{b_i\}$ there is an element of $\{a_i\}$ such that $b_i \subseteq a_j$.*

We define $\mathbf{P}(X)$ a partition of $E$ containing the grains (or connected components) of an image $X$ together with the background set, $E \backslash X$. For a gray-scale image $f$, the partition $\mathbf{P}(f)$ consists of the connected flatzones (connected zones of constant gray level) of the image. We now have that [22]

**Definition 4.** *Let $\gamma$ be a morphological filter, and $\mathbf{P}(X)$ a partition of $E$, then the filter $\gamma$ is called a connected filter if, for any image $X$, partition $\mathbf{P}(\gamma(X))$ is coarser than partition $\mathbf{P}(X)$.*

Due to this coarseness restriction, connected filters have a connectivity preserving property. As such connected filters can only reassign values to whole connected components of an image, and by doing so possibly merge components, but a connected filter cannot split a single component in two. This means that no new edges can appear by applying a connected filter, but on the contrary, edges can disappear, by setting the neighboring components to the same value. A well known connected filter is the opening-by-reconstruction [26].

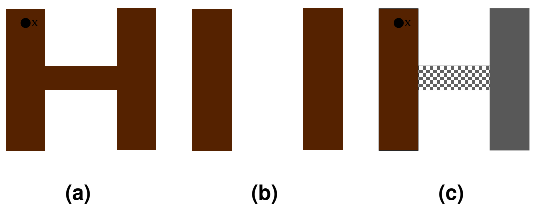


**Fig. 2:** In (2a) the connected component selected by $\Gamma_x(X)$ is shown. In (2b) the elongated bridge is removed by $\psi(X)$. In (2c) the selected component of $\Gamma_x^{\psi}(X)$ only corresponds to the left rectangle, as it so no longer connected with the bridge.

### 2.2 Attribute connected filters

In its currently known form introduced by Breen and Jones in 1996 [3] (though for instance in the form of the area opening already seen before [21]) attribute filters form an important class of the connected filters. Attribute filters work by by computing a single attribute value for each connected component. After, for every component $C$, the attribute is computed and a binary trivial opening is performed [18]:

**Definition 5.** *Let $\Lambda : \mathcal{P}(E) \to \{false, true\}$ be increasing, i.e. $C \subseteq D$ implies $\Lambda(C) \implies \Lambda(D)$. The binary trivial opening is given by:*

$$\Gamma_\Lambda(C) = \begin{cases} C & \text{if } \Lambda(C), \\ \emptyset & \text{otherwise} \end{cases} \tag{4}$$

$\Lambda(C)$ is often of the form of a simple threshold rule:

$$\Lambda(C) = \mu(C) \geq \lambda \tag{5}$$

With $\mu(C)$ an increasing scalar function and $\lambda$ a threshold. The binary attribute opening is now defined as follows [22]:

**Definition 6.** *Let $\Lambda$ be an increasing criterion, and $\Gamma_x(X)$ as defined in eq. (3). The attribute opening $\Gamma^\Lambda$ on a set $X$ is then given by:*

$$\Gamma^\Lambda(X) = \bigcup_{x \in X} \Gamma_\Lambda(\Gamma_x(X)), \tag{6}$$

As $\Gamma_x(X)$ provides us the connected component containing $x$, the binary attribute opening thus takes the union of the trivial openings for every grain of $X$.

**Example 1.** A commonly used attribute is the area. By filtering on the grains using the area threshold one obtains the area opening [21]. In fig. 1 we see the result of an area opening with a certain threshold on three disks of different sizes. As one can easily see this filter is indeed a connected filter as no new edges are created and partition $\mathbf{P}(\gamma(X))$ is indeed coarser than partition $\mathbf{P}(X)$, because for the remaining foreground objects we have equality, and for the removed grain we now have that it forms a subset of the background.

If the criterion evaluated in Definition 6 is not increasing, i.e. when the computed attribute is not dependent itself on the scale of the regions, then the transformation also becomes not increasing. Even if the increasingness property is not fulfilled, the filter remains idempotent and anti-extensive. For this reason, the transformation based on a non-increasing criterion is not an opening, but a thinning. Examples of attribute thinning attributes are the shape factor, orientation, or homogeneity criteria. Attribute thinning has an ambiguous grayscale extension, and multiple rules exist to deal with its non-extensiveness, as explained in section 4.

### 2.3 Second generation connectivity

From a connectivity class $\mathcal{C}$ it is possible to create a child class, which enriches the connectivity class by defining a new connectivity opening. This is referred to as second-generation connectivity [8, 10, 15, 16].

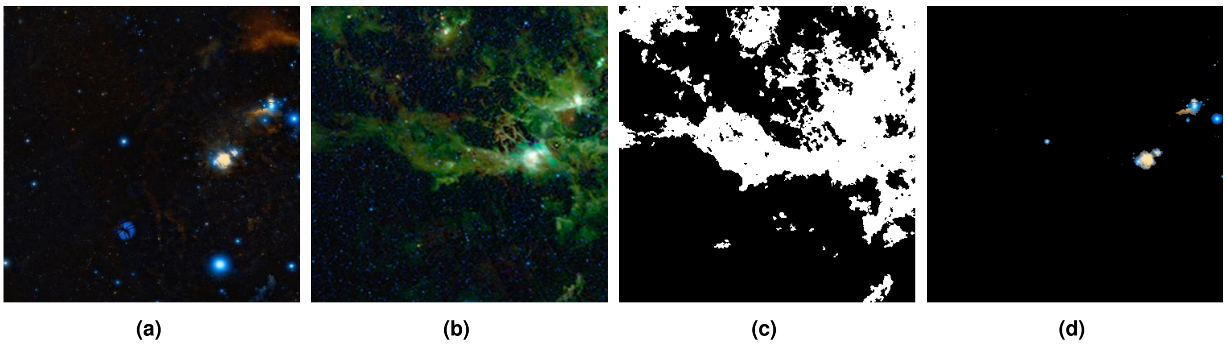


**Fig. 3:** Two pictures of our galaxy [17], using the visible spectrum (3a), and the near-infrared spectrum (3b). The binary connectivity mask (3c) created by thresholding fig. 3b, and the resulting clustered structures (3d) by using $\Gamma_x^M(X)$ and masking it with the original color image.

A connectivity operator commonly either clustering- or contraction-based. The clustering-based connectivity operator enriches $\mathcal{C}$ by merging connected regions together, i.e. by their relative distances. The contraction-based operator further divides the connected components into sub-components.

### 2.3.1 Clustering-based connectivity

The clustering-based connectivity is based on an extensive, and increasing operator $\psi$ such as the Minkowski dilation, or a closing, by using connected structural elements.

**Definition 7.** *Let $\{\Gamma_x | x \in E\}$ be the connectivity openings associated with E. If $\psi$ is a strong clustering on $\mathcal{P}(E)$, then the connectivity opening of $\mathcal{C}^\psi$ is given by:*

$$\Gamma_x^\psi(X) = \begin{cases} \Gamma_x(\psi(X)) \cap X & \text{if } x \in X \\ \emptyset & \text{otherwise} \end{cases} \tag{7}$$

The connectivity opening $\Gamma_x^\psi(X)$ first extends the image by $\psi$, it selects the connected component of the extended image, and masks it with the original image. The result is that the operator returns a new merged connected component without changing the position or contour of the connected regions.

### 2.3.2 Contraction-based connectivity

The contraction-based connective operator subdivides a class into multiple subcomponents, by using an anti-extensive, increasing operator, such as an opening, but not an erosion.

**Definition 8.** *Let $\{\Gamma_x | x \in E\}$ be the connectivity openings associated with E. If $\psi$ is a contraction on $\mathcal{P}(E)$, then the connectivity opening of $\mathcal{C}^\psi$ is given by:*

$$\Gamma_x^\psi(X) = \begin{cases} \Gamma_x(\psi(X)) & \text{if } x \in \psi(X) \\ \{x\} & \text{if } x \in X \backslash \psi(X) \\ \emptyset & \text{otherwise} \end{cases} \tag{8}$$

The connectivity operator $\Gamma_x^\psi(X)$ typically disconnect smaller objects, and preserves the connectivity of larger features. The disconnection of smaller objects corresponds to the second case of Definition 8, where previously connected pixels, are regarded as singleton. As an example of the connectivity operator $\Gamma_x^\psi(X)$ we consider an opening, where an erosion is followed by a dilation. We depict this in fig. 2 where an elongated bridge is removed, and thus the two previously connected rectangles, are now regarded as separate connected components. Furthermore, the pixels of the bridge are now considered disconnected.

## 3 Mask-based connectivity

The previous second-generation connectivity filters introduced in section 2.3.1 and section 2.3.2 have problems because of their superfluous constraints caused by their structural operators. As example, we may wish to use erosion as our structural operator, but this violates the idempotent property of the algebraic opening of the connectivity opening [8]. Additionally we face problems when combining both an opening and closing, as the resulting operator is neither extensive nor anti-extensive. Ouzounis and Wilkinson [8] propose a solution by using a mask instead of a connectivity operator. The desired features of the connectivity can be achieved by processing operations on X, and then use it as a mask $M = \psi(X)$. We can then use the following connectivity opening:

**Definition 9.** *Let $\{\Gamma_x | x \in E\}$ be the connectivity openings associated with E. If $M \subset E$, then the connectivity opening of $\mathcal{C}^M$ is given by:*

$$\Gamma_x^M(X) = \begin{cases} \Gamma_x(M) \cap X & \text{if } x \in X \cap M \\ \{x\} & \text{if } x \in X \backslash M \\ \emptyset & \text{otherwise} \end{cases} \tag{9}$$

The connectivity opening $\Gamma_x^M(X)$ is very similar to the connectivity opening defined in Definition 8. Connected components are retrieved by performing the connectivity opening on $M$, instead of $\psi(X)$, and intersecting it with the original image. The mask $M$ does not have to be a resulting operation on $X$. An example is shown in fig. 3, where two pictures are taken of our galaxy using different spectral responses. We wish to cluster the galaxies in a nearby region on the right, based on their near-infrared spectrum. A threshold of the second picture is used as a mask for the connectivity opening $\Gamma_x^M(X)$. The resulting connected cluster is shown in fig. 3d. Note that the mask does not have to be binarized, and a generalization to grayscale images exists, as explained in section 4.

The attribute opening defined in Definition 6 can be combined with the mask-based connectivity to create a new opening:

**Definition 10.** *Let $\Gamma_x^M(X)$ be related to the child connectivity class $\mathcal{C}^M$ and $\Lambda$ a trivial opening on E. Then the combined connectivity opening is given by:*

$$\Gamma_M^\Lambda(X) = \bigcup_{x \in X} \Gamma_\Lambda(\Gamma_x^M(X)) \tag{10}$$

The connectivity opening $\Gamma_M^\Lambda(X)$ provides attribute filtering on the connected component of the marker $x$ imposed by the mask $M$. This has useful purposes when we consider clustering connected regions, as we can reject or accept the connected component based on the combined metrics of the individual connected regions.

## 4 Tree-based computation

The previous defined second-generation connectivity, and attribute-based connectivity can also be extended to gray-scale images. This can be done through threshold superposition [5], where a gray-scale image $f$ is decomposed into binary images, based on thresholding $f$ at all levels $h \in [0, N-1]$, where $N$ is the number of graylevels of the image. Formally this can be defined as:

$$T_h(f) = \{x \in E | f(x) \geq h\} \tag{11}$$

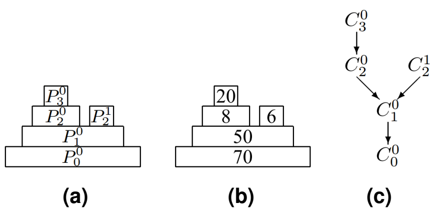


**Fig. 4:** Figure taken from [18]. The peak components of the threshold decomposition(4a), the calculated attribute values corresponding to the peak components (4b), and the Max-tree of the input signal (4c).

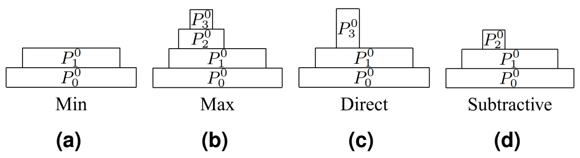


**Fig. 5:** Figure taken from [18]. The result of different filtering rules on the Max-tree (4c), with the threshold $\lambda = 10$.

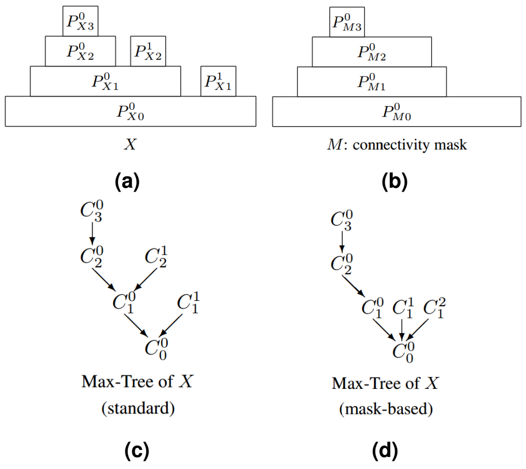


**Fig. 6:** Figure taken from Ouzounis [7]. The peak component of the image with standard connectivity (6a), the peak components of the mask (6b). The Max-tree of the image (6c), and the combined Max-tree (6d) of $X$ imposed by $M$.

In eq. (11) all the pixels corresponding to values higher than threshold $h$ are selected. It is decreasing with respect to $h$ so that when $k > h$ it follows that $T_k(f) \subseteq T_h(f)$. This is also referred to as hierarchical nesting [5]. Attribute openings can be evaluated by Definition 6, at every threshold-level $h$, and combining the resulting connected components. However, using this direct approach leads to an $O(N^2)$ implementation, which is not desired. A better method is to represent the data in a different data-structure, commonly referred to as Max-tree [14], which allows an efficient implementation in $O(N)$.

## 4.1 Tree structure

A grayscale image can be represented in a rooted tree structure. This is composed by the root node, which contains $T_0(f)$, i.e. all the pixels in the image, and thus represents the entire image domain $E$. The levels in the depth of the tree represent the graylevels of the image and at each level, the number of nodes corresponds to the number of connected components present in the binary image obtained by thresholding the current graylevel. By increasing the value of the graylevel $h$, the thresholded image will show separated connected components, represented by nodes at the level in the tree corresponding to the threshold. Those nodes are then linked to their parent nodes at the closest inferior level in the tree. In the image, each parent node corresponds to a connected component which is a superset of the connected component represented by the children nodes in the tree. The procedure is iterated until the threshold reaches he maximum graylevel of the image, which defines the leaves of the tree (the absolute maxima of the image).

In fig. 4 we show an example of a Max-tree of a 1-dimensional signal. A tree node is denoted as $C_h^p$, and a peak components as $P_h^p$, where $h$ corresponds to the grayvalue of the treenode, and $p$ is a position index of the connected component at grayvalue $h$. In fig. 4a the peak components are shown at each threshold level, and the associated attribute values are shown in fig. 4b. In fig. 4c we show the Max-tree of the resulting threshold decomposition. Note that the arrows of the tree are reversed, where descendants point towards the root instead of the other way around. This representation is common, as in practice the tree is computed starting from the peak components.

## 4.2 Tree-based attribute filtering

Once the tree is defined, the criterion associated with the transformation is evaluated at each node, i.e. the attribute is checked against a reference value $\lambda$. Subsequently, the tree is pruned by removing the nodes that do not satisfy the criterion according to a filtering rule. There are two typologies of filtering rules: *pruning* strategies, which remove or preserve a node together with its descendants, and *nonpruning* strategies, where if a node is removed, its children are linked to the parent of the removed node. Below we briefly discuss a few strategies. The *Min*, and the *Max* decision rules [14] are pruning strategies, while the *Direct* and the *Subtractive* rules [18] are non-pruning strategies.

1. *Max*: A node that does not satisfy the criterion is suppressed when all its descendants also do not not satisfy the criterion.
2. *Min*: A node is suppressed with its descendants, when it does not satisfy the criterion, even when its descendants do.
3. *Direct*: A node is suppressed when it does not satisfy the criterion, but its descendants are leaved intact.
4. *Subtractive*: If a node does not satisfy the criterion, then it is removed and all its descendants are lowered by its gray level.

When the tree is processed and pruned by attribute filtering, it is common to transform the pruned tree back to an image. This is done by assigning to each pixel the gray-level correspondent to the highest level of the tree having a node whose correspondent connected component in the image encloses the pixel. Here, the difference between the subtractive and the direct rule comes in to play, as the descendants nodes of the Max-tree have a decreased gray-value when it does not satisfy the criterion. In particular, the subtractive rule proved to be specifically useful when associated to attributes for describing the shapes of objects [19]. In fig. 5 an example of the effect of different filtering rules is shown. In fig. 7 tree filtering has been applied on a 3d volume image. We show it is possible to extract the pig from the 3d volume, segmenting it from the background and its containing coins. This is done using the non-compactness attribute, which measures the degree to which a shape is compact, by using the ratio of the surface area and its volume.

### 4.2.1 Extension to second-generation connectivity

Attribute filters applied to Max-trees can be extended to second-generation connectivity. In particular we can combine it with mask-based connectivity as defined in Definition 6. We can compute the attributes of the connected components imposed by the mask. In this case the mask is now also a grayscale image; its extension to grayscale is given by threshold decomposition. To clarify, the $T_h(f)$ is imposed by $T_h(m)$ for each threshold level $h$, where $m$ is the grayscale mask-image. This can be implemented by a dual-input Max-tree algorithm. It operates like the conventional Max-tree, only it requires two input images; the original image $f$ and the connectivity mask $m$. Both Max-trees are computed in parallel, where the Max-tree of the mask is used to partition or cluster the peaks, and its corresponding attributes, of the original image [8]. An example of a dual-input computation is

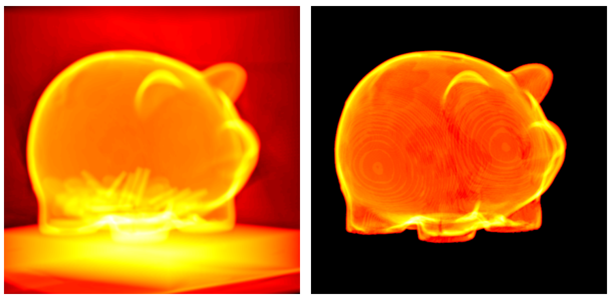


**Fig. 7:** Volume rendering of a density scan taken from [1]. The original model without filtering (7a), and our tree-based filtered result (7b) using the non-compactness attribute where $\lambda = 1.6$

shown in fig. 6. The first two diagrams illustrate the nesting of peak components in the original image $f$ and the connectivity mask $m$. In the resulting tree $C_2^0$ and $C_2^1$ are merged into a single node, where the attributes of the components are combined, and the node $C_1^1$ is split up to a number of singletons (i.e., $C_1^1$ and $C_1^2$).

### 4.3 Computation

Many algorithms exist which allow computation of the Max-tree, and a good overview and comparison of algorithms is given by Carlinet and Géraud [4]. There does not seem to be a clear winner which works well with all cases. The performance of the algorithms are highly dependent of the number of gray levels, additionally the memory footprint highly differ based on the used approach. We note that the high-memory footprint methods are consistently faster [4], however they might not perform well in embedded environments, or with very large images. When a low number of gray values is required (i.e., low quantization), there are two clear winners [2] (low memory footprint) and [14] (high memory footprint). When a high amount of quantization is required ($>$ 18 bits) a good choice is a modified version of [2] by Carlinet and Géraud [4] (high memory footprint), or the flooding-based approach by Wilkinson [24], which has a low-memory footprint.

## 5 Attribute-space connectivity

As the attribute connected filters of section 2.2 are restricted to a simple connected class, they cannot model generalized groupings. By using a similar substitution as with the mask based connectivity we can easily define a second order attribute connected filter:

$$\Gamma_{\psi}^{\Lambda}(X) = \bigcup_{x \in X} \Gamma_{\Lambda}(\Gamma_x^{\psi}(X)), \tag{12}$$

with $\Gamma_x^{\psi}(X)$ as defined in Definition 7. The second order attribute connected filter allows us to perform an attribute filtering (e.g. area opening) on a clustering section 2.3. For the contraction-based (or partitioning) case, however, it is shown by Wilkinson in [22] that unless the criterion is true for any singleton $x$, the attribute opening is equivalent to performing the standard attribute opening on $\psi(X)$. As such, any structural information of the connected components of $X \setminus \psi(X)$ is lost and the filtering results in severe edge deformation.

Due to these limitations Wilkinson [22] transforms an image into a higher dimensional space, which he coined as the attribute-space. In order to achieve this an injective function $\Omega$ is used together with an increasing inverse mapping $\Omega^{-1}$.

**Definition 11.** *An attribute space transform pair $(\Omega, \Omega^{-1})$ from $E \leftrightarrow E \times A$, is a pair of operators such that:*

- *$\Omega : \mathcal{P}(E) \to \mathcal{P}(E \times A)$ is an injective mapping from the image domain $E$ into $E \times A$, where $A$ is some space space encoding the local properties or attributes of pixels in any image,*
- *$\Omega(\emptyset) = \emptyset$,*
- *$\Omega(\{x\}) \in \mathcal{C}_{E\times A}\ \forall\, x \in E$, with $\mathcal{C}_{E\times A}$ the connectivity class used in $E \times A$,*
- *$\Omega^{-1} : \mathcal{P}(E \times A) \to \mathcal{P}(E)$ is a mapping such that $(x,a) \in E \times A$ is projected to $x$,*
- *$\Omega^{-1}(\Omega(X)) = X\ \forall\, X \in \mathcal{P}(E)$,*
- *$\Omega^{-1}$ is increasing (i.e. $Y_1 \subseteq Y_2 \implies \Omega^{-1}(Y_1) \subseteq \Omega^{-1}(Y_2)$).*

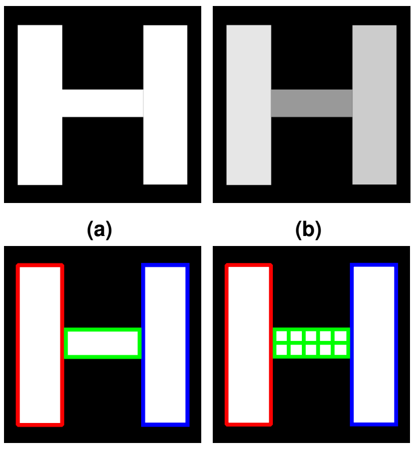


**Fig. 8:** An input image containing a single connected component (8a), the attribute values of granulometry-based attribute spaces with vertical structuring elements (8b), and the determined connectivity in attribute space (8c). Backprojection gives us 3 distinct connected components. For regular second generation filtering (8d) the bridge is split up in its individual pixels (represented as the small squares)

The preceding allows us to define a connected filter as defined by Definition 4 working in attribute-space just as we would for an attribute filter working on a regular image. Projecting the result of that filter back on $E$ gives us:

**Definition 12.** *an attribute-space connected filter $\Psi^A : \mathcal{P}(E) \to \mathcal{P}(E)$ more formally defined as $\Psi^A(X) = \Omega^{-1}(\Psi(\Omega(X)))$, with $X \in \mathcal{P}(E)$ and $\Psi : E \times A \to E \times A$ a connected filter.*

Wilkinson also shows that $\Psi^A$ has the following properties [22]:

- Due to increasingness of $\Omega^{-1}$ it holds that if $\Psi$ is (anti-) extensive, so is $\Psi^A$ as well.
- Increasingness of $\Psi$ does not imply increasingness of $\Psi^A$.
- Idempotence of $\Psi$ implies idempotence of $\Psi^A$ only if $\Psi(\Omega(X)) = \Omega(\Psi^A(X))$. This obviously holds if the pair $(\Omega, \Omega^{-1})$ is bijective, but the preceding is more general.

From these properties it follows that attribute-space connected filter $\Psi^A$ is not a filter following the classical notion of a filter as defined by Serra [15] as in the general case it is neither increasing nor idempotent.

It is important to note that $\Omega$ transforms an image on a per-pixel basis. Therefore it allows connected regions of $E$ to be disconnected in the attribute-space.

### 5.1 Granulometry-based attribute spaces

As an example of an attribute space we can use the local width around a pixel $\{x\}$, which is then assigned to that pixel. The attribute is thus the value $r$ for which, given a family of structuring elements $\{B_r\}$ with size $r$ (known as a granulometry), is the largest value where it holds that $x \in X \circ B_r$, with $X \circ Y$ a structural opening of image $X$ with the structuring element $Y$ [26]. As we deal with discrete images in this paper, the sizes of our structuring elements will also only be discrete values ($\mathbb{Z}^{0+}$). This allows us to define the transformation pair $(\Omega, \Omega^{-1})$:

$$\Omega_w(X) : \mathcal{P}(E) \to \mathcal{P}(E, \mathbb{Z}^{0+}) = \{(x, \max[r \in \mathbb{Z}^{0+} | x \in X \circ B_r]) | x \in X\}$$

$$\Omega_w^{-1}(Y) : \mathcal{P}(E, \mathbb{Z}^{0+}) \to \mathcal{P}(E) = \{x \in E | (x,y) \in Y\}$$

**Example 2.** Let our granulometry $\{B_r\}$ be the set of vertical lines with length $r$. In our attribute space our attribute for point $\{x\}$ will thus be the length of the longest vertical line that goes through $\{x\}$ while still fitting inside of $\Gamma_x(X)$. As fig. 8 shows us, following Definition 8 using a contracting second generation filtering would result in an undesirable result, as for the centerpiece we would have for every pixel $\{x\}$ that $x \in X \backslash \psi(X)$. Attribute space connectivity, on the other hand, allows us to retrieve three distinct components.

## 6 Discussion

We provided an overview of several connectivity methods, and some of their combinations. Our preliminary results suggest that the tested methods are usable in their solution domain. Where classical second generation connectivity only allowed us to perform certain dilations, closings, and openings [7], mask-based second generation connectivity allows us to use any arbitrary mask (some more sensible than others). Specifically, it allows us to use any operator on the image to generate our mask, or even combine the connectivity information from different light spectra. An example of which was shown in fig. 3. Mask-based connectivity might, however, cause oversegmentation [25], resulting in singletons, and in quite some cases the loss of edge preservation is undesirable. A possible future direction to solve this is to find the implicit boundaries of segmented objects, instead of treating the boundary pixels as singletons [25]. We think this is possible by using a Voronoi-based approach, in contrast with a energy minimization scheme to find the boundary separable path.

The dual-input max tree, an extension of the regular max-tree, not only provides us an efficient platform for attribute filtering, but it also extends to second generation connectivity using mask-based connectivity. It has been shown that it can be used to effectively filter shapes from volumes and images [18], as can be seen in fig. 7. Many algorithms exist which allow computation of the Max-tree. No clear winner exists, because of the dependency of the number of gray numbers, and the memory footprint varies based on the used approach (high-memory footprint methods are consistently faster). When a low number of gray values is required there are two clear winners: [2] (low memory footprint) and [14] (high memory footprint). When a large number of graylevels are required a good choice is a modified version of [2] by Carlinet and Géraud [4] (high memory footprint), or the flooding-based approach by Wilkinson [24], which has a low-memory footprint.

Attribute-space connectivity solves some of the problems belonging to attribute filtering with second generation partitioning connectivities. As it is able to handle overlapping regions, it is in some papers compared to hyper-connectivity [13]. It is shown by Wilkinson [23] that any hyperconnectivity is an attribute-space connectivity, but that the reverse is not necessarily true. The biggest downside of attribute-space connectivity is that, in its current form, it is not applicable to grayscale images. We feel that the tree based approach of section 4.2 might be suitable for this. As the mapping might not be an increasing one, careful considerations need to be taken concerning the used (non-) pruning strategies, methods which have not yet been applied to n-tuples in general.

## Acknowledgements

The authors wish to thank the anonymous reviewers for reviewing this paper, and M. Wilkinson for his guidance and support.

## References

[1] M. Bauer. Piggy bank dataset, 2006.

[2] C. Berger, T. Géraud, R. Levillain, N. Widynski, A. Baillard, and E. Bertin. Effective component tree computation with application to pattern recognition in astronomical imaging. In *IEEE Int. Conf. Image Processing, San Antonio, TX*, 2007.

[3] E. J. Breen and R. Jones. Attribute openings, thinnings, and granulometries. *Computer Vision and Image Understanding*, 64(3):377–389, 1996.

[4] E. Carlinet and T. Géraud. A comparison of many max-tree computation algorithms. In *Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 73–85. Springer, 2013.

[5] P. Maragos and R. D. Ziff. Threshold superposition in morphological image analysis systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(5):498–504, 1990.

[6] O. Nempont, J. Atif, E. Angelini, and I. Bloch. Fuzzy attribute openings based on a new fuzzy connectivity class. application to structural recognition in images. In *IPMU*, volume 8, pages 652–659.

[7] G. K. Ouzounis. *Generalized connected morphological operators for robust shape extraction*. University Library Groningen][Host], 2009.

[8] G. K. Ouzounis and M. H. F. Wilkinson. Mask-based second-generation connectivity and attribute filters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):990–1004, 2007.

[9] G. K. Ouzounis, P. Soille, and M. Pesaresi. Rubble detection from vhr aerial imagery data using differential morphological profiles. In *Proc. 34th Intl Symp. Remote Sensing of the Environment*, 2011.

[10] C. Ronse. Set-theoretical algebraic approaches to connectivity in continuous or digital spaces. *Journal of Mathematical Imaging and Vision*, 8 (1):41–58, 1998.

[11] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *Image Processing, IEEE transactions on*, 4(8): 1153–1160, 1995. ISSN 1057-7149. doi: 10.1109/83.403422.

[12] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *Image Processing, IEEE transactions on*, 4(8): 1153–1160, 1995.

[13] P. Salembier and M. H. F. Wilkinson. Connected operators. *Signal Processing Magazine, IEEE*, 26(6):136–157, 2009.

[14] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *Image Processing, IEEE Transactions on*, 7(4):555–570, 1998.

[15] J. Serra. Connectivity on complete lattices. *Journal of Mathematical Imaging and Vision*, 9(3):231–251, 1998.

[16] J. Serra. Connections for sets and functions. *Fundamenta Informaticae*, 41(1/2):147–186, 2000.

[17] R. S. Stuart Lowe, Chris North (Cardiff University). Chromoscope, 2014.

[18] E. R. Urbach and M. H. F. Wilkinson. Shape-only granulometries and grey-scale shape filters. In *Proc. Int. Symp. Math. Morphology (ISMM)*, volume 2002, pages 305–314, 2002.

[19] E. R. Urbach, J. B. T. M. Roerdink, and M. H. F. Wilkinson. Connected shape-size pattern spectra for rotation and scale-invariant classification of gray-scale images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(2):272–285, 2007.

[20] L. Vincent. Grayscale area openings and closings, their efficient implementation and applications. In *First Workshop on Mathematical Morphology and its Applications to Signal Processing*, pages 22–27, 1993.

[21] L. Vincent. Morphological area openings and closings for grey-scale images. In *Shape in Picture*, pages 197–208. Springer, 1994.

[22] M. H. F. Wilkinson. Attribute-space connectivity and connected filters. *Image and Vision Computing*, 25(4):426–435, 2007.

[23] M. H. F. Wilkinson. Hyperconnectivity, attribute-space connectivity and path openings: Theoretical relationships. In *Mathematical Morphology and Its Application to Signal and Image Processing*, pages 47–58. Springer, 2009.

[24] M. H. F. Wilkinson. A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 1021–1024. IEEE, 2011.

[25] M. H. F. Wilkinson and J. Oosterbroek. Mask-edge connectivity: Theory, computation, and application to historical document analysis. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 1334–1337. IEEE, 2012.

[26] R. E. Woods and R. C. Gonzales. *Digital Image Processing*. PEARSON, 2008. ISBN 013505267X.

# Comparing colour morphological operators

Klaas L. Winter and Stephan Groenewold

**Abstract**— Morphological operators are nowadays used extensively in image processing. Using morphological operators, complex tasks can be achieved using simple operators and combinations of operators. Examples of ways in which morphological operators can be used include: noise removal, skeletonisation [7], filtering of image features [13], and texture classification [5]. Unfortunately, although the basic morphological operators have a standard definition for grey-scale images, they do not have a standard definition for colour images. Numerous papers have been written, suggesting morphological operators suitable for use with colour images. However, none of the suggested methods is used as a standard definition, and all of the methods have their own advantages and disadvantages. In this paper a number of existing colour morphological operators are compared based on their performance in several tasks. Two of these tasks are general image processing related tasks, noise removal and edge detection. The last task looks at the bias and colour preserving properties of the different morphological operators. We show that component wise application of the basic greyscale morphological operators yields the best results for the first two tasks, but also point out that the results of this method may not be desired in all situations.

**Index Terms**—Colour morphology, multivariate data ordering, vector ordering, colour images.

## 1 Introduction

Morphological operators are operators used for manipulating images, first introduced by Serra [10]. There exist two basic morphological operators, which act as building blocks for a whole range of morphological operators. The basic morphological operators are erosion and dilation. Both these operators work by for every pixel of an image checking what the minimum or maximum pixel value is in a neighbourhood of pixels surrounding the original pixel. This neighbourhood is often called a mask. Erosion is defined by taking the minimum value, while dilation is defined by taking the maximum value. Erosions or dilations on their own are somewhat limited. However, when these operators are combined they can be used to do various image operations, including: noise removal, skeletonisation, filtering of image features, and texture classification.

Unfortunately, although both erosion and dilation have standard definitions for grey-scale images, no such standard definitions exist for colour images. The reason that no standard definitions exists is the fact that it is hard to define what the minimum or maximum is of a range of colour values. Numerous operators have been proposed that can be used on colour images, each with their own advantages and disadvantages. Because no operator exists that has no disadvantages, none of the proposed operators is used as standard definition.

Because of the large quantity of available colour morphological operators, it can be hard to decide which morphological operator to use in which use case in order to get the best results. To find out which operator to use to get the desired results when working with morphology, this paper will compare several existing colour morphological operators in various situations. The way these morphological operators work is explained in Section 2. The morphological operators are used to perform noise removal and edge detection in Section 3. Furthermore, we study the bias of the different operators and their vector preserving properties in Section 4. An overview of the results and a conclusion is given in Section 5.

## 2 Compared methods

Since the number of existing colour morphological operators is quite high, we have not compared all of them in this paper. In total, ten methods are looked at in this paper. We have chosen these ten methods as they are quite varied with respect to their bias and color preserving properties. Many more exist however, such as Fuzzy-Pareto ordering [8]. Unfortunately due to time limitations, we had to leave these to a future research. To help understand how these methods work, a brief explanation is provided for each method.

### 2.1 No relationship between vector components

By far the easiest way of performing morphological operations on vector data is by treating every vector component separately. Basically, this means that the same morphological operators can be used on every vector component as those used for grey-scale images. Unfortunately, this technique is only feasible when no relationship exists between the different vector components. In the case of a colour image, where the vector components are red, green, and blue, or RGB, a relationship does exist. When applied to colour images, the filtered image can contain colours that did not occur in the original image, also known as false colours. At places in an image where there is little colour variance this effect is not that noticeable. However, at colour edges false colours can easily appear.

One way to remedy the problem of false colours is to apply the mentioned technique on the image in HSV space. HSV space is a colour space where the first channel, the hue, represents the colour, the second channel, the saturation, determines how grey a colour is, and the third channel, the value, determines how dark a colour is. Although this can introduce colours that do not exist in the image, it can not introduce hues that do not exist in the image. The downside of this method is that it is biased towards whatever hue has the highest value.

### 2.2 Ordering vectors

When a morphological operator is not allowed to introduce false colours or non-existing vectors into an image, there needs to be a way to sort the vectors. Unless the different components of the vector data already have some sort of ordering, sorting vectors is hard to do unbiasedly. For example, in a colour image it could be the case that red is deemed more important than blue and that blue is deemed more important than green. In this case it is easy to sort the colours, as some sort of lexicographical ordering can be used. However, usually there is no clear ordering in the colours of an image. Red is usually deemed just as important as blue.

When no clear ordering exists within the components of an image, an ordering is needed that is as unbiased as possible. There are quite a lot of existing methods that perform some kind of ordering. We will look at some existing methods of vector ordering.

#### 2.2.1 Lexicographical ordering

Lexicographical ordering is by far the easiest ordering method. This ordering is extensively discussed, among others papers, in [3]. The

• *Klaas L. Winter is a master computing science student at the University of Groningen, E-mail: K.L.Winter@student.rug.nl.*
• *Stephan Groenewold is a master computing science student at the University of Groningen, E-mail: S.Groenewold.1@student.rug.nl.*

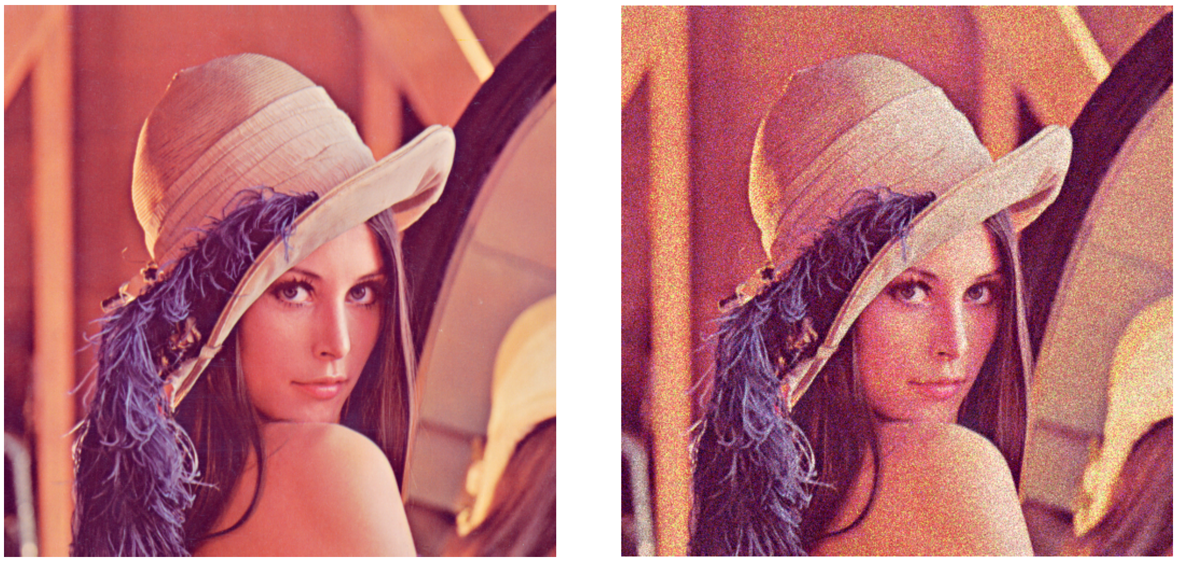


Fig. 1. The original Lena image and the Lena image corrupted with zero-mean Gaussian noise with variance $5 \times 10^{-3}$.

way lexicographical ordering would work on a colour image, is by first sorting the colours on the red colour component. If two colours have the same red colour component, they are sorted on the green colour component. If two colours have both the same red and green colour component, they are sorted on the blue colour component. Although this method will always give a complete ordering without any ambiguity, it is highly biased towards the red colour component.

### 2.2.2 $\alpha$-Trimmed lexicographical extrema

E. Aptoula et al. [1] propose to use a less strict form of lexicographical ordering. To achieve this, the method first compares two vectors based on the first component of the vectors. The $\alpha\%$ most important vectors are then used to compare based on the second component. This process continues until all components have been handled. When using colour images, the authors propose to first convert the image to HSV and first compare based on the value, second on the saturation, and last on the hue. The method can be less biased compared to the original lexicographical ordering, because vectors that would otherwise be ignored because of a low first component now still have a chance to be chosen as having the highest value.

### 2.2.3 Bit wise ordering

J. Chanussot et al. [4] propose to use a bit wise ordering. The idea is to take the bit interpretation of the different components and interleave them. In a colour image this would mean that the first bit would be the first red bit, the second bit would be the first green bit, etcetera. The resulting integer would be used to order the colours on. Although this method is also a complete ordering, it is still quite biased towards the red colour component. Say that a colour with only a red component of 1000 is compared to a green colour with only a green component of 1111. In this case, the green colour is almost twice as bright as the red colour, however, the red colour has a higher value.

### 2.2.4 Distance ordering

To order vectors unbiased towards certain colours or values, it is possible to use a distance function to determine the most important vector. This basic ordering, and extensions to it, are discussed by Gonzalez et al. [6]. The distance ordering method looks at the distance between a vector and the largest or smallest vector as determined by component wise ordering, depending on whether the maximum or minimum is desired. This way, the result will contain vectors that are as close as possible to the extreme component wise vector. The disadvantage of this method is that the ordering is not a complete ordering, but a preordering. A preordering is an ordering that can give two different vectors the same importance. This means that such an ordering can lead to ambiguous results. When, in the case of distance ordering, two different vectors have the same distance, the vector that was seen first is chosen. This effect can lead to noisy results in areas with ambiguity.

### 2.2.5 Vector length ordering

A slight variation on the distance ordering method is vector length ordering. Vector length ordering calculates the sum of components for every vector. The most important vector is the one with the highest sum. Just like distance ordering this method is unbiased. However, this method has the same disadvantage as the distance ordering: the ordering is a preordering.

### 2.2.6 Using colour properties

When working with colour images, it does not always make sense to look at the RGB components of the image. Intuitively, humans do not look at images in terms of red, green, and blue components but in terms of hue, saturation, and brightness. By converting the image to the HSV colour space, we can define the ordering of colours in a more intuitive way.

Intuitively it makes sense to order different colours based on their brightness, with brighter colours being classified as having a higher value than darker colours. The problem that remains is what to do with the saturation and the hue.

M. I. Vardavoulia et al. [12] propose to first order on value, or brightness, and when two values are the same, the colour with the lowest saturation is chosen. To avoid the problem of ordering hues, it is proposed to always have a pixel keep its hue. This can be advantageous in the case where colours should not move, for example, when performing illumination equalisation. However, when it is expected that colour areas should be able to expand or contract, this method fails.

The same is proposed by G. Louverdis et al. [9], except that when both value and saturation are equal, the colour with the highest hue is chosen as being the most extreme. This method does not have the problem with colour areas not being able to expand or contract. Unfortunately, this method is biased and makes certain colours more important than other colours.

## 3 Results

To compare the performance of the different morphological operators, we have tested their performance when used for various tasks. In these experiments we quantitatively compare the results of each of the ordering methods we have implemented.

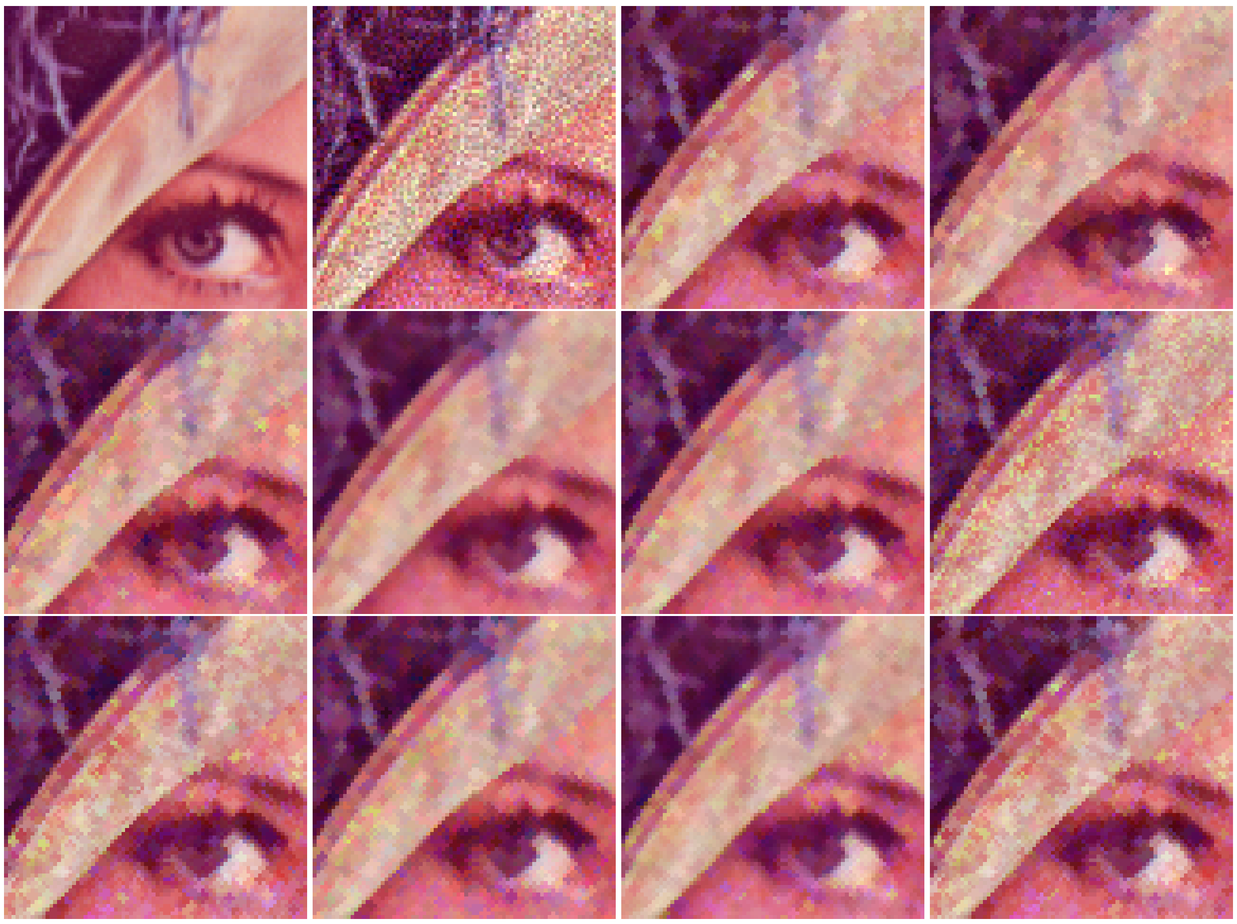


Fig. 2. Cut outs of the same area for each of the filtered results. (a) original, (b) unfiltered noise image, (c) $\alpha$-lexicographic ordering(40%), (d) $\alpha$-lexicographic ordering(70%), (e) bit wise ordering, (f) component wise, (g) distance ordering, (h) lexicographic HSV with hue preservation, (i) lexicographic HSV, (j) vector length ordering, (k) HSV component wise, (l) lexicographic RGB.

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | l |

### 3.1 Noise removal

Our first experiment involved comparing the noise removal capabilities of the different morphological operators. For this experiment, we used the famous Lena image. To this image we added zero-mean Gaussian noise with variance $5 \times 10^{-3}$. The original image and the image with added noise can be seen in Figure 1. To compare the results of the different techniques, a measure is needed that indicates how well noise has been removed, as compared to the original image. We have chosen to use the normalised mean squared error (NMSE) for this. The NMSE and variants are often used when looking at noise removal performance. Examples of usages are the paper by Aptoula et al. [2] and the paper by van de Gronde et al. [11]. This quantity is calculated as follows:

$$NMSE = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\|\mathbf{f}(i,j)-\mathbf{f}'(i,j)\|^2}{\sum_{i=1}^{N}\sum_{j=1}^{M}\|\mathbf{f}(i,j)\|^2} \tag{1}$$

In this formula, N and M represent the dimensions of the image. The functions $\mathbf{f}(i,j)$ and $\mathbf{f}'(i,j)$ represent the color vector at pixel location $i,j$ in the original and the filtered image respectively.

To remove the noise in the Lena image (Fig. 1), we used a smoothing filter defined as the pixelwise average of an opening followed by a closing and a closing followed by an opening, which is also known as the OCCO filter. Given the erosion and dilation operators are defined as $\ominus$ and $\oplus$ respectively, the opening $\circ$ and closing $\bullet$ operations can be defined as:

$$f \circ b = (f \ominus b) \oplus b \tag{2}$$

$$f \bullet b = (f \oplus b) \ominus b \tag{3}$$

where f is an input image, and $b$ is a structuring element. Using this, the OCCO filter can be defined as:

$$OCCO(f,b) = \frac{1}{2}((f \bullet b) \circ b) + \frac{1}{2}((f \circ b) \bullet b) \tag{4}$$

This method however clearly introduces false colours, as it takes the average of the two results. This however is not necessarily an issue when performing noise removal. When an image is contaminated with noise, the original colours have already been lost. As such, colour preservation becomes less important. As will be seen in the results, techniques that do not guarantee colour preservation have an advantage when filtering noise.

The structuring element we used for this filter is a $3 \times 3$ cross. We tested various structuring elements and while the results differed in the normalised mean squared error, the relative performance did not change significantly. An advantage of this small structuring element is that it does not blur the image to a large degree. The results of performing the OCCO filter with each of the methods we implemented can be seen in Table 1. A piece of each of the resulting images can be seen in Figure 2.

Interestingly, the best results were achieved using the most simple method, namely component wise ordering. As mentioned before, this can be explained by the fact that component wise ordering is not limited by the colours found close to a pixel, as it does not guarantee preservation of colours. Because of this, it has a larger range of possible output colours. The other methods are not only limited by the fact that they guarantee vector preservation, but also by the fact that most methods have a bias towards a certain colour. This is a bad property when performing smoothing, as one channel will potentially receive

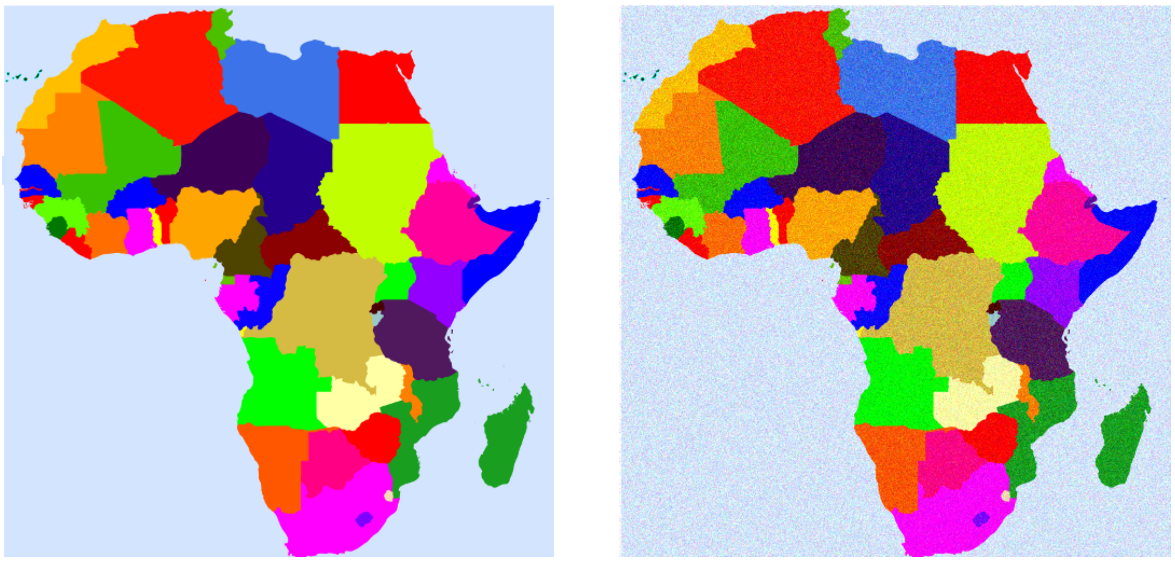


Fig. 3. The original Africa image and the Africa image corrupted with zero-mean Gaussian noise with variance $5 \times 10^{-3}$.

Table 1. NMSE values for the filtered Lena images

| Method | NMSE $\times$ 100 |
|---|---|
| $\alpha$-trim 40% | 1.17 |
| $\alpha$-trim 70% | 0.86 |
| Bit wise | 1.09 |
| Component wise | 0.57 |
| Distance | 0.88 |
| Lexicographical HSV, hue preserved | 1.99 |
| Lexicographical HSV | 1.42 |
| Maximum length | 1.06 |
| HSV component wise | 1.17 |
| Lexicographical RGB | 1.42 |

Table 2. Edge detection error measures

| Method | Used threshold | NMSE |
|---|---|---|
| $\alpha$-trim 10% | 0.6 | 0.6051 |
| $\alpha$-trim 70% | 0.6 | 0.6103 |
| Bit wise | 0.6 | 0.3243 |
| Component wise | 0.8 | 0.1952 |
| Distance | 0.6 | 0.4068 |
| Lexicographical HSV, hue preserved | 0.6 | 0.6716 |
| Lexicographical HSV | 0.6 | 0.6444 |
| Maximum length | 0.6 | 0.5146 |
| HSV component wise | 1.1 | 0.5596 |
| Lexicographical RGB | 0.6 | 0.6051 |

more smoothing than others. The results of the $\alpha$-trimmed ordering method best approach the results of component wise ordering, given the right alpha is chosen. The distance ordering takes the third place, also giving acceptable results.

Methods that perform quite badly on this experiment are the HSV based methods. The hue preserving method struggles, as it is unable to remove or suppress the new hues introduced by the noise. The not hue preserving variant does not suffer from this issue and performs slightly better. However, it considers brighter colours more important, which will be the noise in some cases. As such, both of these methods have colours in their results that were introduced by the noise and were not present in the original image. This effect can clearly be seen in Figure 2. By filtering the HSV components independently, the best results of the HSV methods was achieved, as it has the largest freedom in output. As such it does suppress the new colours introduced by the noise.

### 3.2 Edge detection

Our second experiment looks at edge detection. Morphological operators are good candidates for edge detection, as they are not extremely sensitive to noise, yet very simple in design. The image that is used to detect edges on is a geographical map of Africa, as seen in the left image of Figure 3. In this map every country has an individual colour. When a grey-scale version of this image is edge detected, quite a high number of edges are not detected. The reason for the undetected edges is that two different colours may look very similar in grey-scale. Performing edge-detection in a colour image directly should solve this problem. To make the edge detection more challenging, Gaussian noise with a mean of zero and a variance of $5 \times 10^{-3}$ has been applied to the original Africa image. When no noise is used, most of the methods produce almost identical results, making it hard to compare the different methods. The noisy version of the Africa image can be seen in the right image of Figure 3.

To perform the edge detection the following equation is used:

$$(f \oplus b) - (f \ominus b) \qquad b = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{5}$$

Here, $f$ equals the noisy Africa image, and $b$ equals the used mask. The reason for using a square mask is that a smaller mask makes the edge detection unnecessarily hard, while a larger mask makes the edges too thick. Afterwards, the result is thresholded based on the sum of the individual red, green, and blue components. The threshold is chosen in such a way that the best result is achieved.

To be able to see how well each method performs, a reference set of edges is used to compare the results to. To obtain the reference edges a component wise operator was used on a noiseless Africa map. The threshold used to obtain the black on white edges is 0.3. Figure 4(a) shows the used reference image. To compare the results to the reference edges, the normalised mean squared error, or NMSE, is used in the same way as was done in Section 3.1.

It must be noted that due to the presence of the minus operator in Equation 5, the equation is not colour preserving. This is not a big issue because the final result does not contain any colours, only white on black edges. Table 2 shows a list of methods, their used threshold, and the normalised mean squared error achieved. Some of the resulting edges can be seen in Figure 4.

When looking at the NMSE it is clear that the component wise method yields the smallest NMSE. The generated images seen in Figure 4 confirm this. The component wise method found almost all of the edges also detected in the reference image. The second best results were achieved by the bitwise method, which recognised the general

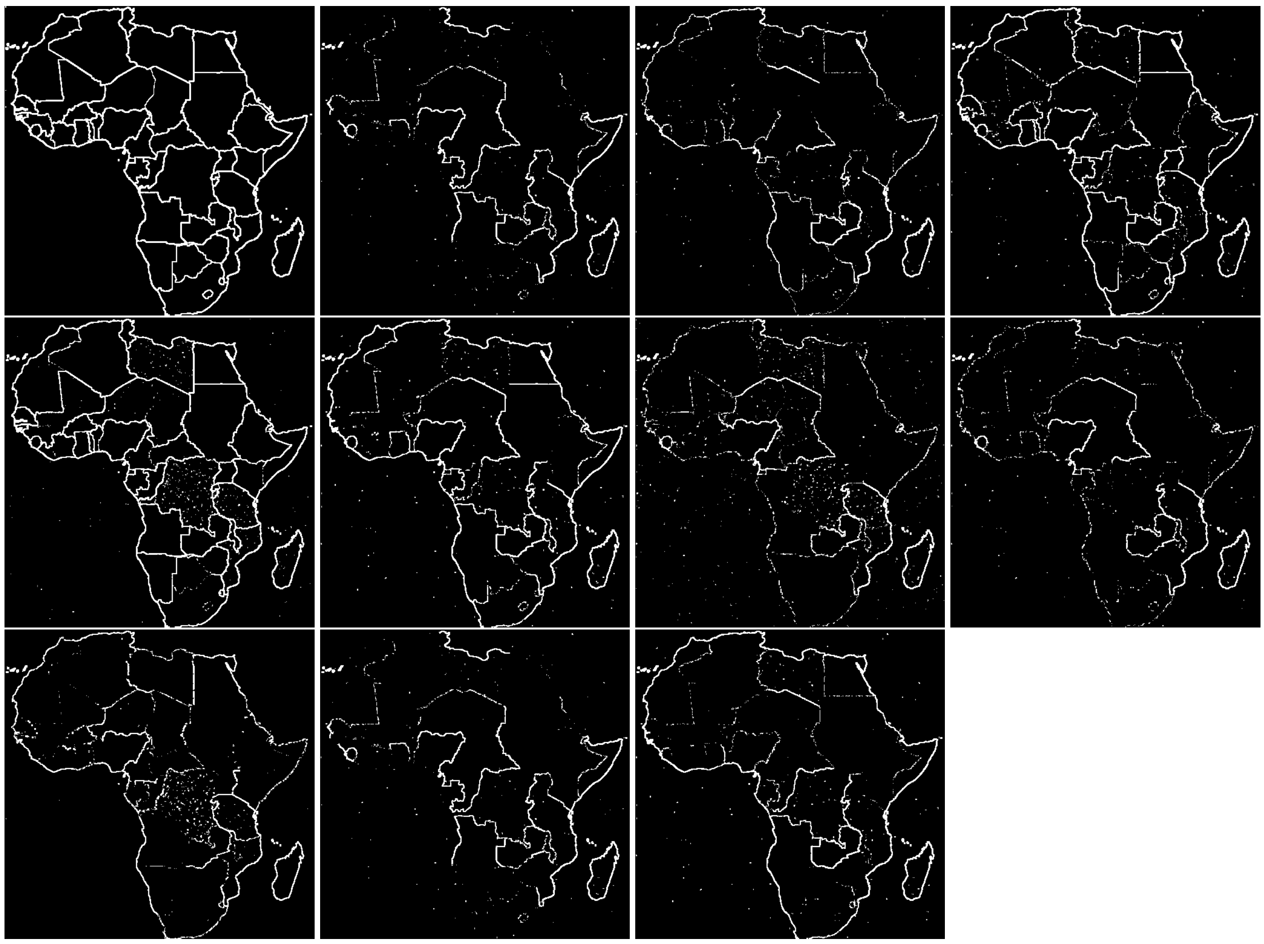


Fig. 4. Detected edges in the Africa image. (a) reference edges, (b) $\alpha$-lexicographic(10%), (c) $\alpha$-lexicographic(70%), (d) bit wise, (e) component wise, (f) distance, (g) hue preserving lexicographical HSV, (h) lexicographical HSV, (i) HSV component wise, (j) lexicographic, (k) maximum length.

| | | | |
|---|---|---|---|
| a | b | c | d |
| e | f | g | h |
| i | j | k | |

shape of Africa and several countries. The rest of the methods missed most of the present edges. Some methods, such as $\alpha$-trim and lexicographic, did not even recognise the general outline of Africa.

One of the influences that seems to determine how well a method performs is its bias. The four best performing methods, component wise, bit wise, distance, and maximum length are all methods that are non-biased or in the bit wise case, close to non-biased. All the other methods, with the exception of hue preserving lexicographical HSV, are biased and perform worse. The best performing biased method is HSV component wise, which is not surprising since it has more freedom in the colours it can output. The hue preserving lexicographical HSV is the worst performing method. This result does not support the correlation between bias and performance. The reason for this exception is that the Africa image is characterised by hue differences, not by brightness differences. Since the hue is not modified by the hue preserving lexicographical HSV, the method can only find edges that occur at a difference in brightness.

All in all, the component wise method performs best when detecting edges. Even though component wise is not vector preserving, it is no problem to use component wise for edge detection. As mentioned before, Equation 5 is not vector preserving no matter the used method. Also, since we are only interested in an end result consisting of white edges on a black background, the introduction of false colours into our image is not an issue. Of course, the Africa image is an image with very clear edges. The results might have been different with a more complex image, such as the Lena image used in Section 3.1.

## 4 Colour preservation & bias

Both previous experiments found the component wise ordering method to be the best method by a reasonable margin, even though this method does not guarantee colour preservation. For both of the test cases we concluded that colour preservation was not an important property. For noise removal, it is not important to keep the colours of the original image, as these colours have been corrupted by noise and are already false colours regardless. In the edge detection case, the colours are not even visible in the end result and as such are not important either. On top of that, the operations performed in both situations are not color preserving by definition. However, colour preservation may be important in other cases. In this section we take a closer look at the effects of the different ordering techniques by looking at a very simple image, and we show that false colours and an inappropriate bias can have undesired effects.

We will take a look at the effect of performing morphological operations on an image consisting of a few basic shapes with constant colours. The image used is shown in Figure 5(a). Using this image, false colours will be easy to see, as colours that are far apart in the RGB space are next to each other. To determine the performance of the methods regarding the introduction of false colours, we perform an erosion on this image. For this erosion we use a $5 \times 5$ structuring element.

The results of applying this erosion operation to the shapes image are also shown in Figure 5. The results of using component wise ordering, bit wise ordering, and maximum length ordering are shown here. While we have performed this experiment with other ordering methods, the results are similar to the results of these three methods.

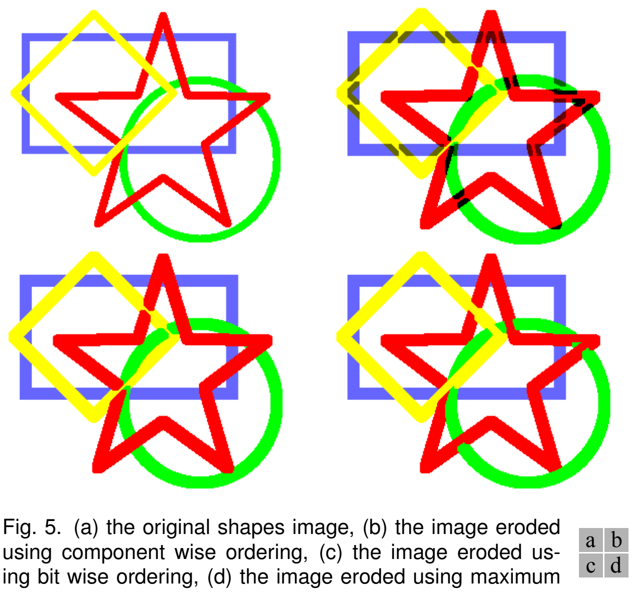


Fig. 5. (a) the original shapes image, (b) the image eroded using component wise ordering, (c) the image eroded using bit wise ordering, (d) the image eroded using maximum length ordering.

What affects the results the most is whether or not the method preserves colours and what, if any, colour it is biased towards.

What is most notable about these results is the result of the erosion performed using component wise ordering. The introduction of false colours is very apparent in this example. The blending of the different colours results in dark patches in areas where different colours are next to each other. It is not biased towards a specific colour and as such introduces these false colours in every location where different colours meet.

The second result is acquired by using bitwise ordering. While this technique does not introduce false colours like the component wise ordering technique, it is biased. It prefers red over green and green over blue. This preference can clearly be seen in the result. Areas next to the red colour are turned red, even when a different colour is nearby.

The final result, constructed by performing the erosion using maximum length ordering, looks similar to the result gotten by using bitwise ordering. It also does not introduce false colours. It does however not have a bias towards any colour. As a result, no particular colour is chosen in the areas where colours meet. All the colours are represented in these areas.

Clearly these techniques all deal with these cases differently. It is hard if not impossible to say which technique performs best. The reason for this is that it depends on the application what the expected result is. In the end it is up to the user of a morphological operator to determine which operator best matches his or her expectations.

## 5 Conclusion

In this paper, we have shown comparisons of several colour morphological operators on several problems. We have compared the different methods based on their noise removal and edge detection capabilities, and have studied their bias and vector preserving properties. Most of the compared methods have been made vector preserving, in order to avoid false colours. However, both in the area of noise removal and the area of edge detection, we have shown that component wise vector ordering has the best performance. It is not terribly surprising that the component wise method has such a good performance, because both problems are not biased towards a certain colour and in both cases false colours are not an issue.

In general, when a morphological operation is not affected by false colours and does not need a colour bias, it is probably best to use the component wise method. When the goal is to eventually produce a modified colour image, it is usually desired to not have the image contain false colours. However, this can be hard to achieve because often subtracting, addition, and other operators are also used, such as in the OCCO filter, which are not vector preserving.

When false colours really are an issue it is obviously best to use a vector preserving method. However, vector preserving methods are usually either biased or they give ambiguous results. When the bias produced by these methods is also an issue, the best strategy is to define a custom method with a bias that fits the problem.

In many cases colour biasing is needed. Say, for example, that red and blue spheres need to be removed from a green background. In cases like this there is no one method that will always work. $\alpha$-trim is slightly modifiable and might work in some cases, but this would still be a limited approach. As such, we can say that in specific situations like this one, the best approach is to modify one of the methods to suit the problem description. An example of this would be modifying lexicographical ordering in such a way that colours are ordered based on the problem at hand.

Only a selection of the available morphological colour operators have been compared in this paper. Other interesting methods exist, such as Fuzzy-Pareto ordering, and future work might show how these methods compare to the methods analysed in this paper.

As a conclusion, the comparisons done in this paper suggest that in general cases it is best to use component wise vector ordering. Only when false colours are really an issue, a vector preserving method would be better. In non-general cases the best approach is to write your own morphological operator.

## References

[1] E. Aptoula and S. Lefèvre. $\alpha$-trimmed lexicographical extrema for pseudo-morphological image analysis. *Visual Communication & Image Representation*, 19, Oct. 2007.

[2] E. Aptoula and S. Lefèvre. A comparative study on multivariate mathematical morphology. *Pattern Recognition*, 40, Feb. 2007.

[3] E. Aptoula and S. Lefèvre. On lexicographical ordering in multivariate mathematical morphology. *Pattern Recogn. Lett.*, 29(2):109–118, jan 2008.

[4] J. Chanussot and P. Lambert. Total ordering based on space filling curves for multivalued morphology. In *Proceedings of the international symposium on mathematical morphology*, volume 4, pages 51–58, 1998.

[5] Y. Chen and E. R. Dougherty. Gray-scale morphological granulometric texture classification. *Optical Engineering*, 33(8):2713–2722, 1994.

[6] P. Gonzalez, V. Cabezas, M. Mora, F. Cordova, and J. Vidal. Morphological color images processing using distance-based and lexicographic order operators. In *Chilean Computer Science Society (SCCC), 2010 XXIX International Conference of the*, pages 258–264, Nov 2010.

[7] B.-K. Jang and R. Chin. Analysis of thinning algorithms using mathematical morphology. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(6):541–551, Jun 1990.

[8] M. Köppen, C. Nowack, and G. Roesel. Pareto-morphology for color image processing. In B. K. Ersboll and P. Johansen, editors, *The 11th Scandinavian Conference on Image Analysis*, volume 1, pages 195–202, Kangerlussuaq, Greenland, 1999.

[9] G. Louverdis, M. I. Vardavoulia, I. Andreadis, and P. Tsalides. A new approach to morphological color image processing. *Pattern Recognition*, 35, July 2001.

[10] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

[11] J. J. van de Gronde and J. B. T. M. Roerdink. Group-invariant colour morphology based on frames. *IEEE Transactions on Image Processing*, 23(3):1276–1288, Mar. 2014.

[12] M. I. Vardavoulia, I. Andreadis, and P. Tsalides. Vector ordering and morphological operations for colour image processing: Fundamentals and applications. *Pattern Analysis & Applications*, 2002, June 2001.

[13] K. Zhang, S.-C. Chen, D. Whitman, M.-L. Shyu, J. Yan, and C. Zhang. A progressive morphological filter for removing nonground measurements from airborne lidar data. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(4):872–882, April 2003.

# Decreasing the amount of computational resources needed for molecular simulations of macromolecules in water

Aloys Akkerman and Robbert-Jan Pijpker

**Abstract**—The key of decreasing the amount of computational resources needed for molecular simulations of macromolecules in water is by decreasing the degrees of freedom of the computational box and decreasing the amount of necessary solvent. Molecular simulations use a computational box for periodic boundary conditions. Most simulations are done in a box with a high degree of freedom, i.e. a rhombic dodecahedron. Decreasing this degree of freedom by transforming this computational box into a triclinic box results in a decrease of computational resources.
The solvent for a molecular simulation scales with the volume of the computational box. By using a near densest lattice packing, this volume is decreased to a near-minimal volume necessary for the simulation, resulting in a decrease of solvent in the simulation.
In this paper a method is presented for transforming any computational box into a triclinic box and two different methods to obtain the near-minimal volume of the box. The first method focuses mainly on finding a near-densest lattice packing of a shape $M$ while the second method will discuss how this can be optimized into a faster method and how this method can be used for ensembles.

**Index Terms**—Molecular simulation, computational geometry, periodic boundary conditions, minimal volume, lattice packing, computational box

## 1 Introduction

A molecular simulation is the simulation of the physical movement of atoms and molecules. The simulation of a macromolecule, i.e. a biomolecule, is based on the interactions of all the atoms of which the molecule consists. To mimic an infinite system of these macromolecules in water, a computational box with periodic boundary conditions is often used. Thus, the macromolecule is placed in a computational box after which this box is surrounded in a space-filling way by replica boxes. Since the simulation has to be as realistic as possible, the molecules in these different boxes are not allowed to interact with each other. The box is filled with a layer of solvent, usually water, to make sure that this does not happen. Since this solvent is used for separating multiple molecules and does not really influence the simulation, it is desired to minimize the volume of the box. This way, the amount of redundant solvent is decreased resulting in less solvent-solvent interactions. This means that by reducing the amount of solvent of the simulateion, less computational resources are needed for the simulation.

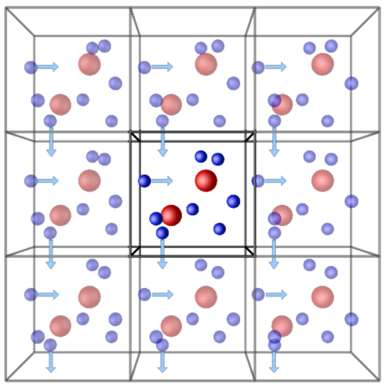


Fig. 1. Example of Periodic boundary conditions [6].

- *Aloys Akkerman is a MSc. student at the University of Groningen, Email: A.Akkerman.3@student.rug.nl*
- *Robbert-Jan Pijpker is a MSc. student at the University of Groningen, Email: R.Pijpker@student.rug.nl*

## 2 Computational boxes

As shown by Fejes Tóth [8], in a three dimensional space there are five different types of box shapes. Figure 2 shows these five box shapes, where "PCT" stands for "primitive cell type".

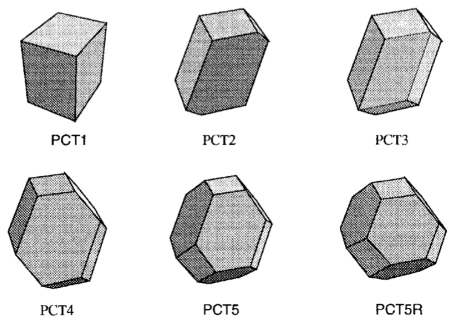


Fig. 2. The triclinic box (PCT1), hexagonal prism (PCT2), two types of dodecahedrons (PCT3, PCT4), the truncated octahedron (PCT5), and a regular instance of truncated octahedron (PCT5R) [2].

Historically, it was believed that the only way to obtain a box with near-minimal-volume was by constructing a tight fitting box. For this reason, complex-shaped boxes were introduced (PCT3 to 5). However, this idea was limited to the fact that a molecule should be unfragmented. H. Bekker [2] showed that a molecule can also be fragmented inside a box by using PBC and that by doing so, every box can be transformed into any other box. This way it is possible to obtain the exact same simulation with a triclinic box as with an octahedron. However, since a neighbourhood search takes the shape of the box into account it is much more desirable to use a triclinic box. Bekker describes in his paper how each box can be transformed into any other box and how a simulation can be set up in a simple box, instead of using a complex one.

### 2.1 Defining box shapes

First, let us describe the size and shape of a box. Since there are multiple ways to define the size and shape of a computational box, we will describe two of them here. The first way is to divide a space into multiple primitive cells, also known as Voronoi regions, by using a lattice and a metric. A lattice, $L$, in 3D space is the set of points:

$$L(P,Q,R) \equiv n_1P + n_2Q + n_3R, \quad (1)$$
$$\text{with } n_1, n_2, n_3 \in \mathbb{Z} \quad (2)$$

where $P$, $Q$ and $R$ are the lattice vectors. A point $p_1$ in one lattice corresponds to a point $p_2$ in a different lattice if their positions are related by:

$$p_1 + \text{lattice vector} = p_2 \quad (3)$$

The squared distance in Euclidean space is given by

$$d^2(p_1, p_2) = (p_1 - p_2)^T m(p_1 - p_2) \quad (4)$$

where $m$ is the metric, defined as a positive definite matrix. Using this metric and the lattice vectors $P$, $Q$, $R$, the shape of the primitive cells is one of the computational boxes $PCT1$ to $PCT5$.

The second way to define box shapes, is by defining the edge vectors of the box. Since a box consists out of parallel edges, a PCT5 box can be described by defining its six edge vectors $b,c,d,e,f,g$, as shown in Figure 3.

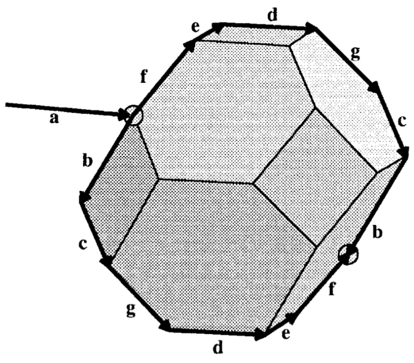


Fig. 3. An instance of a truncated octahedron (PCT5) defined by six edge vectors $b,c,d,e,f,g$[2].

Each of these vectors consists out of three components ($x$, $y$ and $z$) and since $|c,e,g| = 0$, $|b,d,g| = 0$, $|c,d,f| = 0$ and $|b,e,f| = 0$, this leads to a degree of freedom of $6*3-4 = 14$. From these six edge vectors, PCT5 can be degenerated into PCT4 by setting the length of $g$ to 0. In the same way, PCT3 can be obtained by setting the length of $f$ to 0. By making $c,d,e$ to be linearly independent from each other ($|c,d,e| = 0$, PCT2 can be obtained and PCT1 can then be obtained by setting the length of $e$ to 0. The PCT1 has only 3 edge vectors remaining, resulting in $3*3 = 9$ degrees of freedom. This entire process of degenerating box types is shown in Figure 4.

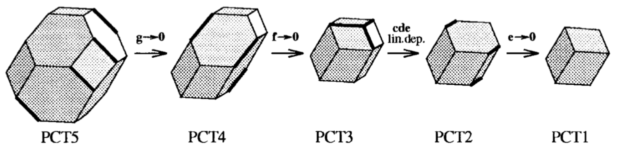


Fig. 4. Degenerating a PCT5 to a PCT1 [2].

## 2.2 Constructing a box

Now let us combine the above described ways to describe the size and shape of a computational box as follows:

$$P = (g + d + e + f) \quad (5)$$
$$Q = (g + b + e) \quad (6)$$
$$R = (f - c + e) \quad (7)$$

This can easily be verified by using Equation 3 in combination with Figure 5

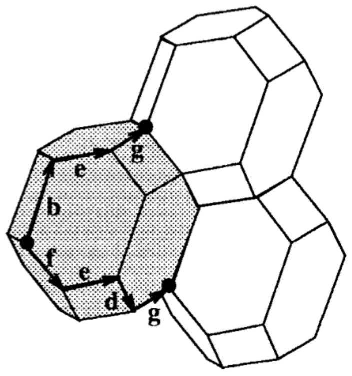


Fig. 5. Example of the lattice vectors of a PCT5 [2].

Now that we have found the three lattice vectors, we can construct a triclinic box from this, by using these vectors as edge vectors of the box. Depending on the simulation, it might sometimes be better to use a rectangular box, also known as PCT1R. Since a PCT1 box does not have to be rectangular, another transformation is sometimes needed. This transformation is done by using a Gram-Schmidt process [7]:

$$U = K \quad (8)$$
$$V = L - (L \cdot \hat{K})\hat{K} \quad (9)$$
$$W = (M \cdot K \hat{\times} L)\, K \hat{\times} L \quad (10)$$

with $\hat{K} \equiv \mathbf{K}/K$ and $K \hat{\times} L \equiv K \times L / |K \times L|$. An example of this transformation is shown in Figure 6.

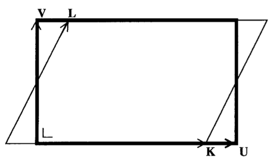


Fig. 6. Transformation of a PCT1 into a PCT1R by using a Gram-Schmidt process [2].

## 2.3 Translating particles

We have shown how each box can be transformed into every other box, specifically the PCT1 and PCT1R boxes. The next step is to make sure that the simulation in the transformed box, PC' is exactly identical to the simulation in the box before the transformation, PC. Two simulations in different boxes in the infinite system, PC and PC', are identical when:

- PC and PC' define the same lattice;
- the particles in PC and PC' are at corresponding positions.

Consider a particle $p$ in PC. This particle might be outside of PC', however as shown before, this particle can be translated into PC' by a unique shift over a lattice vector. This way, all the particles can be translated from PC into PC', causing the molecular simulation of PC' to be identical to PC.

Now, the brute-force way to transform all the particles from one box to another would be to generate all linear combinations of $P$, $Q$ and $R$. This would mean that the implementation consists of a triple loop, making it fairly inefficient. Bekker proposes a more efficient algorithm, that first tries most probable shifts, which is no shift at all, followed by first ordered shifts (i.e. $-1 \leq n_1, n_2, n_3 \leq 1$). After this the second ordered shifts are tried, and so on.

## 3 Near Densest Lattice Packing

In the previous section, we have shown how a computational box can be constructed and used for molecular simulations. Now consider a molecule $m$ as shown in Figure 7a. Often, molecular simulations use a certain cut-off radius $r_{co}$ to truncate short-range interactions between molecules. However, since the infinite space of a PBC consists out of multiple replicas of the same molecule, this interaction is not taken into account for a simulation using PBC. This means that the molecule $m$ should be dilated with a size that is at least $\frac{1}{2}r_{co}$. This dilation will result in the shape $M$ shown in Figure 7b. Now that we have found this shape, we can construct a computational box $B$ around it and we can tessellate the infinite molecular simulation system by this box (Figure 7c,d,e).

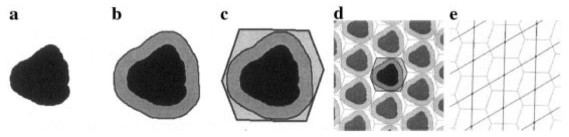


Fig. 7. Dilating a molecule $m$ and constructing a box around it [3].

Important now is the fact that it is desirable to have the volume of $B$ as small as possible to reduce irrelevant computations. However, instead of creating a tight fitting box as is normally done, Bekker et al. [3] proposed a method to find a near densest lattice packing (NDLP). This is because they argue that a tight fitting box does not necessarily result in a box with near minimal redundant solvent, as can be seen in Figure 8. From this near densest lattice packing a general computational box can be derived, which can be used for the molecular simulation.

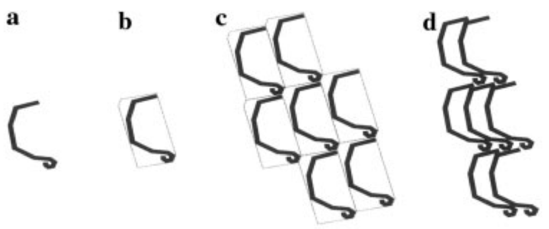


Fig. 8. Example showing that the smallest enclosing PBC box does not always result in a densest infinite molecular simulation system [3].

### 3.1 Finding the NDLP

First of all, in order to find the NDLP of a shape $M$, we have to construct the contact body of $M$. The contact body $N$ has the property that for all points $p$ on the boundary of $N$, the shapes $M$ and $M_p$ (which is the dilated molecule $M$ at position $p$) touch without overlapping. An example of this is shown in Figure 9. For the creation of $N$ the Minkowski sum is used:

$$N \equiv M \oplus -M \tag{11}$$

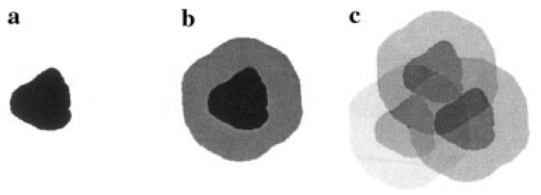


Fig. 9. (a) a body $M$, (b) the contact body $N$ of $M$, (c) Three copies of $M$, all on the boundary of $N$ [3].

The idea behind NDLP is to place $M$ and three translates $M_a$, $M_b$ and $M_c$ of $M$ such that they do not have any overlap and that the volume of the triclinic cell with edge vectors $a$, $b$ and $c$ is minimal. Since this means that we would have to try all combinations of $a$, $b$ and $c$, Bekker et al. suggested to only look at the combinations of $a$, $b$ and $c$ for which holds that every body in the set $\{M, M_a, M_b, M_c\}$ is touched by all the other bodies, i.e. an all-contact situation of $\{M, M_a, M_b, M_c\}$. This reduces the nine dimensionality problem to a three dimensionality problem. Is has been shown, however, that there might also be solutions that lead to a minimal volume that do not fulfil the all-contact situation [4], but since trying to find these situations require a lot more CPU time, it is desirable only consider the all-contact situations.

From all the combinations of $a$, $b$ and $c$, the one resulting in a minimal volume is the one with minimal $|det(a,b,c)|$. It may, however, be possible that for a non-convex shape $M$, a lattice point $d$ exists for which $M$ and $M_d$ have some overlap. This means that, even though $|det(a,b,c)|$ might be minimal for this point, it does not result in a proper simulation and should thus be discarded. The final minimal volume is thus determined by whether there is no point $d$ for which $M$ and $M_d$ have some overlap and for which $|det(a,b,c)|$ is minimal.

```
Data: Point set of a molecule $m$
Result: Near-densest lattice packing for $m$
Construct $M$ by dilating $m$ with $\frac{1}{2}r_{co}$;
Construct $N$ from $M$, by $N \leftarrow M \oplus -M$;
$old\_det\_abc \leftarrow \infty$;
for each point a on boundary of N do
    for each point b on boundary of N and $N_a$ do
        for each point c on boundary of N, $N_a$, $N_b$ do
            if $|det(a,b,c)| < old\_det\_abc$ AND NOT $point\_inside\_N$ then
                store(a, b, c);
            end
        end
    end
end
put $m$ in box with edge vectors $a$, $b$, $c$; fill voids with solvent;
```
**Algorithm 1:** Generating near-densest lattice packing, Bekker [3].

### 3.2 Implementation

For the implementation of the NDLP-algorithm, we have to consider $m$ and $M$ to be point sets while $N$ is a polyhedron. $M$ can be created from first applying the $\alpha$-hull algorithm [5] with $\alpha = \frac{1}{2}r_{co}$ to $m$. This will return the boundary points $m_{bp}$ of $m$. From $m_{bp}$ $M$ can be constructed by taking the Minkowski sum of $m_{bd}$ and $ball$, i.e. $M \equiv m_{bp} \oplus ball$. Here, $ball$ is a set points of points that cover the boundary of a sphere with radius $\frac{1}{2}r_{co}$. From $M$ the contact body $N$ can then be found as described in Equation 11. We will now use again the $\alpha$-hull algorithm with $\alpha$ as the diameter of a grid-cell to obtain a set of triangles that define the boundary of $N$. Now $N_a$ can be constructed from $N$ by

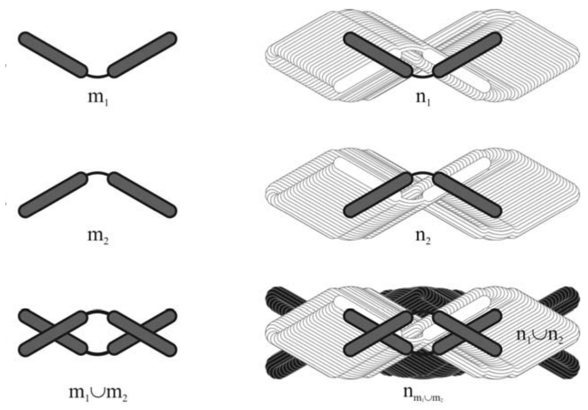


Fig. 10. Two configurations of $m$ ($m_1$ and $m_2$) and their union $m_1 \cup m_2$ resulting in a contact body of $n_{m_1 \cup m_2}$ (in black) and $n_1 \cup n_2$ (in gray) [9].

adding the vector $a$ to every point of $N$. The intersection curve of $N$ and $N_a$ is defined as $N \cap N_a$, from this we can then calculated the intersection of $(N \cap N_a) \cap N_b$, etcetera. A pseudocode implementation of this can is given in Algorithm 1.

## 4 Revised NDLP

As stated before it is desirable to find an optimal unit cell to reduce the computational costs. Generally we define optimal as minimal in total volume, assuming a prescribed minimal distance between any two periodic images is given. By using the NDLP method [3], almost all unnecessary solvent can be removed. Unnecessary meaning that the solvent is further away than half the prescribed minimal distance from any of the periodic images. The volume of the simulation system is greatly reduced using NDLP, resulting in a significant speed-up factor.

However, the NDLP method has some drawbacks. Creating it is slow and memory demanding, especially for simulations of large molecules. The time required to calculate the simulation cell is significant. Another drawback is the way the NDLP method determines a minimal volume cell unit. This cell unit is based on a single structure, and a change in that structure during the simulation is likely to violate the minimal required distance between any two periodic images. This problem is not specific to NDLP, but since NDLP results in a very tight fitting unit cell it is more profound.

A basic method to solve this problem is to estimate a combined structure based on the original structure which is changed in various ways, we call this an ensemble.

### 4.1 Contact body for an ensemble

Consider a molecule $m$ that has $p$ atoms in $k$ different configurations and let $r_i$ be the position of an atom $i$. Since the position of $i$ can change for each different configuration, a configuration $m_j$ can be defined as:

$$m_j \equiv \cup_{i=1}^{j} r_{ij} \tag{12}$$

To compute the contact body of an ensemble of $k$ configurations, a naive approach is to take the union of all the configurations of $m_j$:

$$M \equiv \cup_{j=1}^{k} m_j \tag{13}$$

However, as can be seen in black in Figure 10, this will lead to a non-optimal packing.

A better approach is, instead of calculating a contact body for all the set of points combined, is to calculate the contact body $N_j$ for every $m_j$ by using equation 11 and combining all these contact bodies in a single one:

$$N \equiv \cup_{j=1}^{k} N_j \tag{14}$$

This results in the grey part of Figure 10, discarding the redundant solvent around the contact body of the ensemble.

### 4.2 Vector approximation of the contact surface

Wassenaar et al. [9] state that the surface of a contact body of a molecule $m$ can be defined as a set of vectors that are allowed for translation of $m_j$. When considering that the contact surface is symmetric and located at the origin, and ignoring holes and crevices for placing neighbours of the contact surface, Wassenaar et al. introduce a simple and efficient approach to approximate the contact surface.

Define $s$ as a vector originating from the origin. Let the length of $s$ be set, so that the tip of $s$ lies on the contact surface of N (figure 11).

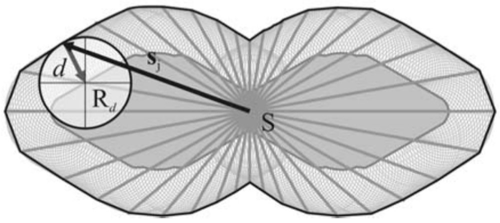


Fig. 11. The contact body $n_1 \cup n_2$, and the vector approximation to the surface of minimal distance. For each of a series of vectors sj starting from the origin, the end point is determined as the maximum point of intersection of the vector with a sphere placed at the surface of the contact body $n_1 \cup n_2$. [9].

Since the contact surface of an ensemble is the union of the contact surfaces for every configuration $m_j$, we can calculate for every vector $s$ in the set of vectors of the contact body, the maximum length for all $m_j$. This maximum length, will become the length of the vector for the final vector of the contact body. This way, the lengths of all vectors $s$ of the contact $N$ can be calculated without having to calculate every $N_j$, avoiding computations on a combined set of contact bodies.

### 4.3 Constructing an optimal lattice from the vector approximation of a contact surface

Since we now have a good representation of the contact surface, we search for three vectors forming the basis for a near-optimal lattice. Such a lattice should have minimal volume and no overlap. Normally an iteration over the triangulated surface would be performed. However, as stated earlier the triangulation is not calculated because of its computational impact.

A new method for determining the optimal lattice vector directly from the set of vertices describing the contact surface is presented by Wassenaar et al. [9]. In 3D, the aim is to find three independent vectors, which together form a basis for the optimal lattice. If we consider a 3D lattice consisting of three vectors **u**, **v** and **w**, we obtain as well 2D lattices created by vector pairs: [**u**, **v**], [**v**, **w**] and [**u**, **w**]. Thus to construct a 3D lattice we only need to search for a combination of vectors [**u**, **v**], and the third vector **w** is sought, under the condition that **w** forms valid 2D lattices with **u** and **v**. Finishing, it must be checked that non of the linear combination of the vectors result in a point existing inside one of the lattices. From all valid vector combinations the most optimal will be selected, this is the one with the lowest determinant.

### 4.4 Implementation

The original NDLP implementation [3] constructed the $\alpha$-hull to calculate the contact body. The $\alpha$-hull is quadratic in the number of points, thus this implementation was costly and therefore it is slow.

In order to speed-up the algorithm the dependency on the $\alpha$-hull must be removed. To achieve this a different approach to constructing an optimal lattice had to be taken, as explained in the previous sections. The new implementation consists of 3 main steps: (1) Generating a triangulated sphere. (2) Filtering points from the structure. (3) Scaling the vertices to the contact surface.

```
Data: Ensemble of structures S
Result: Near-densest lattice packing for S
Generate triangulate sphere approximation (1)
for each structure M in S do
    Filter surface points of structure M (2)
    for each vector I in S do
        for each surface point J of M do
            for each surface point K of M do
                $r_1 \leftarrow$ sphere centered at J-K
                $r_2 \leftarrow$ sphere centered at K-J
                if I has expit point P on $r_1$ or $r_2$ then
                    if length P > length I then
                        $i \leftarrow P$
                    end
                end
            end
        end
    end
end
```

**Algorithm 2:** Generating near-densest lattice packing, Wassenaar et al. [9].

If more than one shape is used, the second and third steps will be repeated for every shape. The largest constructed vectors, as explained in section 4.1, are stored. The algorithm will finish with searching for the three translation vectors that describe the best near-densest lattice packing. The algorithm is described by the pseudo code in Algorithm 2.

## 5 RESULTS

In sections 3 and 4, two methods were described to find a lattice with minimal volume for a molecule, namely, the original NDLP and the revised NDLP. We have already stated that the revised NDLP is a more efficient version of the original NDLP, however, to be able to properly show this, we will first discuss the performance of the original NDLP.

### 5.1 Original NDLP

The original NDLP was mainly designed to speedup simulations of macromolecules by reducing the volume of the computational box. In Table 1 it can be seen that by using the original NDLP the volume of the computational box is on average 50% of a rectangular box and 39% of a dodecahedron. This minimization of volume results in a simulation of 25,000 timesteps that is about two times faster on an AMD Athlon 600 Mhz processor. However, the drawback of using a triclinic NDLP box is that it takes a lot of time to create in comparison to rectangular box or dodecahedron. Even though this only has to be done once for a simulation, it is desirable to improve it, which is why the revised NDLP method was introduced.

### 5.2 Revised NDLP

The revised NDLP has about the same performance in simulation time as the original NDLP, which is why the performance tests of a revised NDLP mainly looks at the construction of the NDLP. Figure 12 shows the performance of constructing a revised NDLP against constructing a dodecahedron.

Originally constructing a NDLP box was much slower than the construction of a dodecahedron box. The original NDLP method took minutes, or even hours to construct the NDLP box. However, as can be seen in Figure 12c, the time that is needed for creating a revised NDLP is almost as fast as the construction of a dodecahedron. The maximum time taken here, on an Intel Core i7 2.80 Ghz processor, is about 36 seconds. Looking at the volume of the revised NDLP box (Figure 12a) we can see that the found lattice also results a volume that is a lot smaller than that of a dodecahedron.

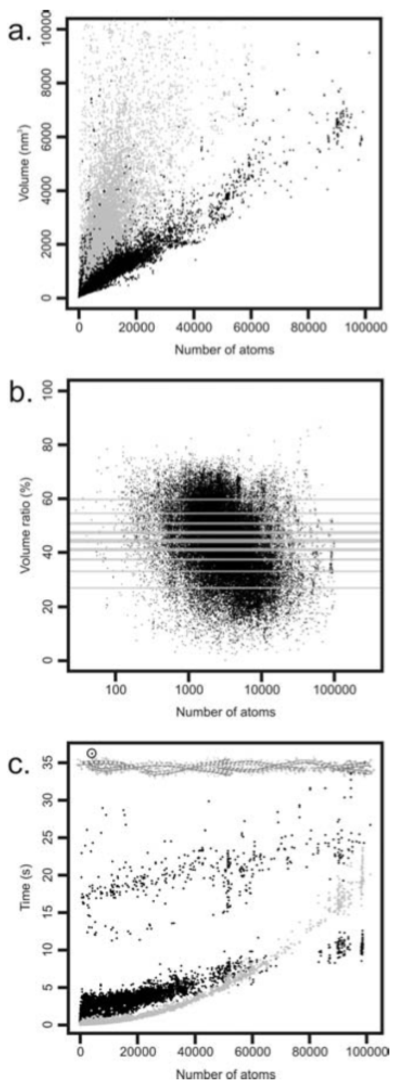


Fig. 12. Revised NDLP a) Volume against number of atoms b) Volume ratio against number of atoms c) Simulation time against number of atoms. In black the performance of the revised NDLP is shown, in gray the performance of a dodecahedron is shown [9].

## 6 DISCUSSION AND CONCLUSION

We have shown how every possible computational space-filling box can be transformed to any other space-filling box, with a preference for a triclinic box because of its simplicity to use. Using this information, we have stated that the computational time of a simulation is dependent on the amount of redundant solvent in the computational box and thus on the volume of the box. We have shown two different ways of obtaining a box that has a near-minimal volume, namely the original NDLP and the revised NDLP methods. The advantage of using an NDLP is that it decreases the time taken by the simulation with a speedup of about 2, however, the original NDLP takes minutes, or even hours, to be created. This is why the revised NDLP method is introduced. The revised NDLP constructs a surface body in a lot more efficient way, resulting in a construction of an NDLP box that takes only a couple of seconds.

The previously used mainstream methods of a molecular simulation usually use a dodecahedron as a computational box. We have shown here that we can find an NDLP in a very efficient way by using the revised NDLP method and that we can construct a simple space-filling computational box from this with a near minimal volume. This results

| Macro-Molecules | | | Rectangular Box | | Dodecahedron | | NDLP Triclinic Box | | Speedup | |
|---|---|---|---|---|---|---|---|---|---|---|
| Nr. | PDB Code | Nr. of Atoms | Volume [nm$^3$] | Simul. Time [min] | Volume [nm$^3$] | Simul. Time [min] | Volume [nm$^3$] | Simul. Time [min] | Factor 1 | Factor 2 |
| 1 | 1A32 | 1102 | 398.58 | 288 | 577.82 | 411 | 118.93 | 85 | 3.38 | 4.83 |
| 2 | 1A6S | 805 | 141.25 | 97 | 142.43 | 105 | 80.08 | 58 | 1.67 | 1.81 |
| 3 | 1ADR | 763 | 126.45 | 91 | 167.25 | 119 | 80.73 | 55 | 1.65 | 2.16 |
| 4 | 1AKI | 1321 | 188.46 | 134 | 233.54 | 168 | 93.99 | 67 | 2.00 | 2.50 |
| 5 | 1BW6 | 595 | 144.18 | 99 | 130.27 | 89 | 66.32 | 45 | 2.20 | 1.97 |
| 6 | 1HNR | 485 | 110.32 | 77 | 124.31 | 89 | 59.30 | 41 | 1.87 | 2.17 |
| 7 | 1HP8 | 686 | 133.17 | 94 | 177.10 | 129 | 77.57 | 53 | 1.77 | 2.43 |
| 8 | 1HQ1 | 982 | 201.62 | 143 | 218.77 | 158 | 103.71 | 72 | 1.98 | 2.19 |
| 9 | 1NER | 768 | 170.14 | 118 | 147.91 | 105 | 85.35 | 58 | 2.03 | 1.81 |
| 10 | 1OLG | 1808 | 297.63 | 210 | 468.93 | 337 | 203.44 | 145 | 1.44 | 2.32 |
| 11 | 1PRH | 11676 | 1031.30 | 759 | 1337.80 | 987 | 611.67 | 467 | 1.62 | 2.11 |
| 12 | 1STU | 668 | 130.79 | 91 | 190.32 | 136 | 73.41 | 50 | 1.82 | 2.72 |
| 13 | 1VCC | 833 | 141.12 | 100 | 152.69 | 108 | 69.77 | 49 | 2.04 | 2.20 |
| 14 | 1VII | 389 | 95.42 | 66 | 99.74 | 68 | 46.96 | 32 | 2.06 | 2.12 |
| 15 | 2BBY | 767 | 155.59 | 109 | 159.26 | 113 | 80.78 | 56 | 1.94 | 2.01 |
| 16 | 1D0G* | 1052 | 255.45 | 183 | 645.92 | 462 | 112.78 | 79 | 2.31 | 5.84 |
| 17 | 1D4V* | 3192 | 1184.47 | 859 | 1319.21 | 951 | 451.23 | 329 | 2.61 | 2.89 |
| 18 | 1AAB | 898 | 264.91 | 190 | 402.38 | 292 | 167.93 | 116 | 1.63 | 2.51 |
| 19 | 2ORC | 720 | 197.03 | 139 | 230.64 | 161 | 125.85 | 88 | 1.58 | 1.82 |
| | | | 5369 | 3945 | 6925 | 4988 | 2707 | 1945 | 1.98 | 2.54 |

Table 1. The results of creating three boxes, namely a rectangular box, a dodecahedron and a triclinic NDP for 19 randomly chosen macromolecules. The figure also shows the simulation time of 25,000 time steps and the average speedup. The number of atoms include both proteins and water. Speedup factor 1 and 2 is the speedup achieved comparing the NDLP triclinic box with respectively a rectangular box and a dodecahedron box. [3]

in a better performance running the simulation.

The NDLP method also has some concerns. It might, for example, happen that some conformational changes may occur that are not covered by the input ensemble, resulting in a problem with the distance criteria. Even though this is a concern that also applies to more conventional setups for molecular simulations, the problems might be more severe because of the use of a near-densest lattice packing.

Also, the way that each structure defines the contact surface of an ensemble, may result in some problems when using very flexible molecules. Namely, for flexible molecules the ensemble is ill-defined and will thus result in some problems. This effect can, however, be countered by using rotational constraints (Amadei et al. [1]) to remove the degree of freedom for rotations.

## References

[1] A. Amadei, G. Chillemi, M. A. Ceruso, A. Grottesi, and A. Di Nola. Molecular dynamics simulations with constrained roto-translational motions: Theoretical basis and statistical mechanical consistency. *J. Chem. Phys.*, 112.

[2] H. Bekker. Unification of box shapes in molecular simulations. *Journal of computational chemistry*, 18(15):1930–1942, 1997.

[3] H. Bekker, J. P. van den Berg, and T. A. Wassenaar. A method to obtain a near-minimal-volume molecular simulation of a macromolecule, using periodic boundary conditions and rotational constraints. *Journal of Computational Chemistry*, 25(8):1037–1046, 2004.

[4] U. Betke and M. Henk. Densest lattice packings of 3-polytopes. *Computational Geometry*, 16(3):157 – 186, 2000.

[5] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, Jan. 1994.

[6] http://isaacs.sourceforge.net/phys/pbc.html.

[7] D. Lay. *Linear Algebra and Its Applications*. Pearson Education, 2002.

[8] L. Tóth. *Regular figures*. International series of monographs in pure and applied mathematics. Macmillan, 1964.

[9] T. A. Wassenaar, S. de Vries, A. M. J. J. Bonvin, and H. Bekker. Squeeze-e: The optimal solution for molecular simulations with periodic boundary conditions. *Journal of Chemical Theory and Computation*, 8(10):3618–3627, 2012.

# Comparing Software Maintainability Predictors

Razvan Florea, Antonios Gkortzis, *University of Groningen*

**Abstract**— Since the introduction of the object-oriented paradigm there has been a great increase in the demand for more complex object-oriented software systems. Thus, it becomes necessary for organizations to find a cost-effective method to maintain these systems. Maintainability, is the quality attribute that indicates the ease with which a software system can be debugged (corrective maintenance), extended (adaptive maintenance), tested (preventive maintenance) and enhanced with respect to quality (perfective maintenance). Quantifying maintainability is necessary from the early phases of development, because it provides useful information to improve the design, the code and the overall software quality. In order to quantify maintainability, several maintainability prediction models have been introduced in the last twenty years. Our research will be focused on two recent maintainability prediction models, which have been characterized as the most accurate, by an independent study. The first model is based on Bayesian Networks, while the second model on Multivariate Adaptive Regression Splines (MARS). Both models are constructed using the metrics and the data sets proposed by Li and Henry. In this study, we compare the above-mentioned models, with respect to their prediction techniques, the calculated metrics and their prediction capabilities. The results show that there is no uniformly optimal solution, but the prediction accuracy of the models depends on the data sets.

**Index Terms**—Maintainability, Prediction, Software design, Software quality assurance, Object-oriented, Metrics.

## 1 Introduction

Maintainability is a very important software quality attribute because maintenance is the most coslty process in the Software Development Life-Cycle (SDLC) [5], [42]. Maintenance cost prediction [21] or maintenance project effort estimation [3] is a part of the software cost estimation [22], [28] and considered necessary information for the project planning [19]. Hence, an effective mechanism that can predict the maintainability during the software development phase is necessary. In recent years, there has been a significant increase in production and use of commercial, big and complex object-oriented (OO) systems implemented with OO programming languages. IEEE defines software maintainability as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [1].

The OO paradigm introduced new concepts in software development like polymorphism, inheritance, classes and encapsulation. The differences of OO programing from non-OO programming made non-OO maintainability prediction models not applicable for OO systems [6]. Likewise, non-OO software metrics like Function Points [4] are not capable of properly measuring the OO code. Thus, inevitably new OO metrics have been introduced, e.g. the metric suites of Chidamber and Kemerer (C&K) [11] and Li and Henry (L&H) [25] concerning OO maintainability. Previous studies have revealed a correlation between the OO metrics and maintainability [2], [8], [17], [29], [35], [40]. Several models with different approaches for predicting the maintainability of OO software systems have been introduced the last twenty years. Specifically, studies [17], [29], [44] and [25] shows that a Multiple Linear Regression model consisting of C&K, L&H and other OO metrics were able to predict software maintenance effort. In [12], [13] and [32], authors present polynomial and regression maintainability assessment models based on several metrics including the Halsteads metrics. A more complex method for predicting maintainability, based on fuzzy deformable types has been introduced in [20]. In this study, maintainability is rationalized as a combination of other quality attributes of a software system like understandability, analyzability and modifiability. Authors in [34] propose a "stochastic feature selection" model while authors in [38] present three software maintainability models based on the non-homogeneous Poisson process.

This research focuses in analyzing two prediction models based on Bayesian networks [40] and Multivariate adaptive Regression Splines (MARS) [43] which are the highest ranked methods among several others, according to the study quality assessment presented in [36]. The analysis is followed by a comparison between the results of the application of the above two models on the Li and Henry's data sets to the results of other pre-existing models used for predicting maintainability. This comparison aims at distinguishing the model that provides the most accurate prediction of the system's maintainability. The results suggests that even if the Bayesian network approach provides better results in the first data set and the MARS model better results on the second data set, no safe conclusions can be drawn since the accuracy of the prediction provided by the models depends on the characteristics of the data sets.

The rest of this paper is structured as follows. Section 2 describes three typical prediction techniques, followed by the two most recent proposed prediction models: based on a Bayesian network and on Multivariate Adaptive Regression Splines. In Section 3 we present the datasets and the metrics on which both models are based on. In Section 4 a comparison between the results of the commonly used models and the two recent models is performed. Finally Section 5 presents conclusions and discussion about future studies.

## 2 Models

In this section, we first make a short description of the following commonly used techniques: the Multivariate Linear Regression (MLR), the Regression Tree (RT) and Artificial Neural Network (ANN) all previously presented in the [40] and [43] study. Then, we focus on presenting the software maintainability models that this study aims to compare: the Bayesian Network and the Multivariate Adaptive Regression Splines.

### 2.1 Commonly used models

#### 2.1.1 Multivariate Linear Regression

Statistical methods, such as regression models, are the best tools for investigating any relationship between dependent and independent variables of small sample size. This model has been the most commonly used technique of the last decade for modeling the linear relationship between one or more independent variables and a dependent variable.

A Multivariate Linear Regression model with independent variable $Y$ and $k$ dependent variables $X_1, X_2, ..., X_k$ satisfies the next equation:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots \beta_k X_{ki} + u_i \quad (1)$$

where $Y_i$ is the $i$th observation on the dependent variable, $\beta_j (j = 1, 2...k)$ is a $k$-vector parameters of the model and $u_i$ are residuals; $k$ is

- *Antonios Gkortzis, E-mail: a.gkortzis@rug.nl.*
- *Stefan Razvan Florea, E-mail: razvan.florea91@gmail.com.*

the number of independent variables [23].

When a MLR model is built, the selection of variables should ensure that only the most important independent variables are included in the model. Two commonly used variable selection procedures are: the backward elimination and the forward elimination. The difference between these two procedures is the way they choose the predictor variables. Thus, while backward elimination eliminates the most insignificant predicator variables, forward elimination enters new ones with a more significant contribution. A more efficient method for variable selections is the stepwise method which at each step either deletes or adds a variable at the regression model.

### 2.1.2 Regression Tree

Regression Tree is a regression based model which can be considered a variant of decision trees that predicts values of continuous variables. In a Regression Tree model, initially, all the data from the analyzed sets are put together in one node. Then, recursively, the algorithm chooses rules for getting the best split to minimize the sum of the squared deviations from the average of the newly separated parts. Once a rule is selected it splits a node into two while the same process is applied to each child node. The recursion stops when no further gain can be achieved. Every branch ends with a terminal node, which is defined by a unique set of rules (all the rules that were selected until reaching this terminal node) [7].

Figure 1 presents an example of a Regression Tree which contains five sequential splitting rules represented as circles, that produce six terminal nodes represented as rectangles. Each decision rule emphasizes on how the conditions that the predictors variables must satisfy are split. These characteristics reveals that a Regression Tree is very similar to a decision tree.

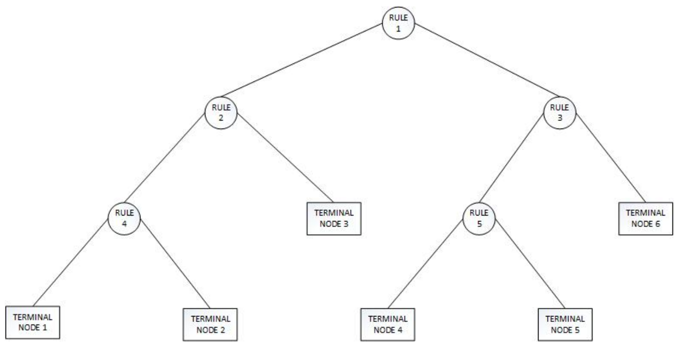


Fig. 1: A Regression Tree model example

### 2.1.3 Artificial Neural Network

An Artificial Neural Network is a computational model constructed in strong correlation with the way biological nervous systems (in particular the brain) process and make use of information. The neurons from a human brain are represented by nodes in an ANN, while the synapses becomes weights [15]. All these components are working in unison to solve specific problems, the best results being achieved for problems with a high number of variables, where the dataset is very large and the relationships between variables are difficult to understand. Typically, an ANN is defined by three types of parameters: the interconnection pattern between the different types of parameters, the adjusting process of weights and the activation function that converts neuron's weighted input to its output activation.

Figure 2 presents the simplest form af an ANN, which is fully connected and consists of three layers of nodes: Input Layer, Hidden Layer and Output Layer. For example, for an effort estimation task, inputs could be the database size, the complexity of the software or the lines of code. The hidden layer is made from neurons connected to the input layer and also to the output layer. The output layer, in this case, has only one output neuron, but depending on the output of the model it can have more. For this example, of effort estimation, an output neuron result is in terms of man-months, man-weeks or man-hours.

What has attracted the most interest in neural networks is the possibility of learning by training. The most used algorithm for training an ANN is back-propagation. It initializes the network with random values of weights and then it trains it by passing a set of input-output pairs. Thus, the algorithm looks for the minimum of the error function in weight space, minimizing the collective error between actual and expected outputs of the training set [37]. One of the most popular network activation functions using this algorithm is the sigmoid, which is a function $s_c : R \to (0,1)$ defined by the next formula:

$$s_c(x) = \frac{1}{1+e^{-cx}} \tag{2}$$

where the constant $c$ is a slope parameter and is chosen arbitrarily, but in accordance with the analyzed case.

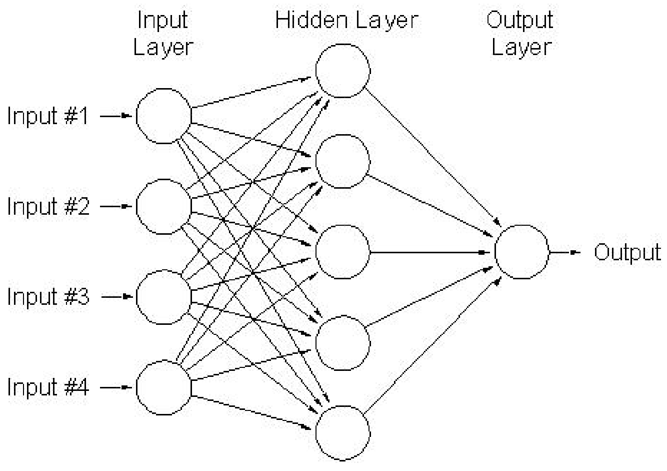


Fig. 2: An Artificial Neural Network example [39]

## 2.2 Models to be compared

In this section, we will describe the theoretical concepts about the software maintainability models that our paper is comparing: The Bayesian network and the multivariate adaptive regression splines. This information is sufficient for understanding the analysis from the next section.

### 2.2.1 Bayesian Network

A Bayesian network, Bayes network, Bayes(en) model or probabilistic directed acyclic graphical model is a probabilistic graphical model represented as a directed acyclic graph and organized in events. These events are represented by nodes interconnected with directed links that represents causal relationship between them.

Figure 3 presents an example of a Bayesian network in which the events are represented as ellipses, while the links are represented as arrows. In addition to the graph structure, for a directed model, at each node it is necessary to specify the Conditional Probability Distribution (CPD), which is the probability distribution of an event when a previous event is known to be a particular value. Because the variables from the example are discrete, the CPD can be represented as a table, which will contain all the probabilities of all the combinations of this node parents. In the example from Figure 3 it can be seen that the event "WetGrass" (W=true) can be caused by two previous events: "Rain" (R = true) or "Sprinkler" (S=true). These relationships has different strengths as shown in the table. For example, we see that Pr(W=true — S=true, R=true) = 0.9 (last row), and hence, since each row must sum to one, Pr(W=false — S=true, R=true) = 1 - 0.9 = 0.1. The first event, called "Cloudy", has no parents, what causes that its CPT specifies only the prior probabilities that this event to be either $true$ or $false$, which in our example are 0.5 and, respectively, 0.5.

In a Bayesian Network, the arrows between nodes are defined as conditional probabilities. For instance, for two events $X$ and $Y$ the probability that $Y$ to happen being influenced by $X$ is represented as $P(Y|X)$ [31]. For decades conditional probabilities of events of interest have been computed from known probabilities using Bayes' theorem, which for the events presented is:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{3}$$

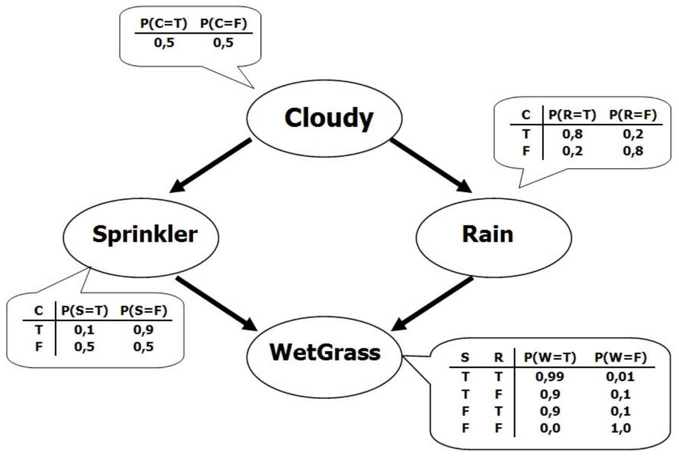


Fig. 3: A Bayesian Network example [30]

In a Bayesian network the chain rule provides a more compact representation of the joint probability $P(U) = P(A_1, A_2, \cdots, A_n)$ to make the probability calculations easier. If the joint probability table $P(U)$ is obtained, then the probabilities $P(A_i)$ can be calculated as well as the probabilities $P(A_i|e)$, where $e$ is the evidence [41]. By the chain rule of probability, the joint probability of all the nodes in the network from Figure 3 is:

$$P(C,S,R,W) = P(C) * P(S|C) * P(R|C,S) * P(W|C,S,R) \tag{4}$$

In the general case, where $\theta$ denotes a vector whose components are parameters that describe the probability distributions over the network, and $x$ denotes a piece of data, the joint probability distribution $P(\theta, x)$ over the network is defined as:

$$P(\theta, x) = P(x|\theta)P(\theta) \tag{5}$$

where $P(\theta)$ is called the prior probability distribution of the network and $P(x|\theta)$ is called the likelihood of the data and is defined as the probability of the evidence given the parameters $\theta$. Writing the Bayes' theorem for the same variables, we have:

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{6}$$

where $P(\theta|x)$ is called the posterior probability defined as the probability of the parameters $\theta$ given the evidence $x$.

From Equations 5 and 6 results the next relation between these concepts that we have introduced here:

$$\text{posterior distribution} = \text{likelihood} \times \text{prior distribution} \tag{7}$$

which is the principle of Bayesian update, where, using the likelihood of observed data the prior probability distribution is updated [40].

### 2.2.2 Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines (MARS) is a non-parametric regression technique introduced by Freidman [18] and used in network intrusion analysis [33], biomedical analysis [16] and many other applications. The first attempt to introduce MARS in software engineering has been by the authors in [9] and [10]. MARS is a technique suitable for modeling large and complex relationships by assigning a different regression equations to subsets separated from the training data.

The following MARS analysis is presented in study [43]. MARS use piece-wise linear regression splines to fit data in a Cartesian coordinate system in which $X$ is the independent variable while $Y$ is the dependent variable. In Figure 4 the notion of the concept of knots is presented, where the knots are the points that define a region of data in which a distinct equation is going to run. $x1$ & $x2$ are variables representing two knots that demarcate three regions where different linear relationships are identified. These variables are identified through a fast search procedure [40].

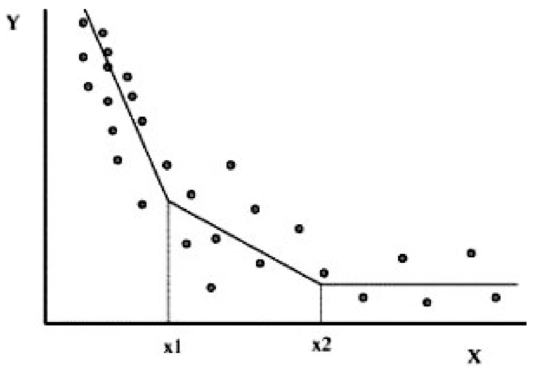


Fig. 4: Example knots in MARS

The following equation 8 represents a general MARS model:

$$\hat{y} = c_0 + \sum_{m=1}^{M} c_m \prod_{k=1}^{K_m} b_{km}(x_{u(k,m)}) \tag{8}$$

where $\hat{y}$ is the dependent variable, $c_0$ is a constant, $b_{km}(x_{u(k,m)})$ is the truncated power basis function with $u(k,m)$ being the index of the independent variable used in the $m$th term of the $k$th product. $K_m$ is a parameter that limits the order of interactions while the splines $b_{km}$ are defined in pairs:

$$b_{km}(x) = (x - t_{km})_+^q = \begin{cases} (x - t_{km})^q & \text{if } x > t_{km} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

and

$$b_{km+1}(x) = (x - t_{km})+^q = \begin{cases} (t_{km} - x)^q & \text{if } t_{km} > x \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

for $m$ an odd integer, where $t_{km}$ is known as the knot of the spline, $q \geq 0$ is the power to which the splines are raised in order to manipulate the degree of smoothness of the resultant regression models. The only exception is when $q = 1$ where linear splines are applied.

Every MARS model must be built in two phases in order to reach an optimum level: a forward stepwise selection process followed by a backward "pruning" process. In the first phase a repeatable process occurs in which at each step the process chooses from a set of splits the one that minimizes some "lack of criterion". How often this is repeated depends on a predetermined maximum number of basis functions. In the second phase, where the backward "pruning" process is performed, the "lack of fit" criterion defines how each basis function contributes to the descriptive abilities of the model, eliminating stepwise those that contributed the least. The following equation 11 defines the generalized cross-validation criterion (GCV) for $n$ observations in the data set, on which the used "lack of fit" measure is based:

$$GVC(M) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2 \Big/ \left[1 - \frac{C(M)}{n}\right]^2 \tag{11}$$

In this equation, $M$ is the number of non-constant terms in the model, and $C = (M)$, which is a complexity penalty function, in order to avoid over-fitting and to promote the parsimony of the model, is responsible for penalizing the model complexity. Its definition is as follows:

$$C(M) = M + cd \tag{12}$$

where $c$ represents a penalty factor for each basic function optimization predefined by the user and $d$ is the effective degrees of freedom (independent basic function in the model). The built model is now able to estimate the relative importance of a variable in terms of contribution to the fit of the model. The most relatively important variable is the one that its deletion most reduces the fit of the model.

## 3 Data sets

In this chapter we will present the OO metrics that are used in the datasets followed by a distribution and correlation analysis of these metrics on the following data sets: The User Interface Management System (UIMS) consisting of metric data from 39 classes and The Quality Evaluation System (QUES) consisting of metric data from 71 classes [25].

### 3.1 Studied Metrics

Both prediction models make use of a set of eleven OO metrics, five published by Chidamber and Kemerer [11], four by Li and Henry [25], the SIZE1 which is the lines of code and the CHANGE representing the number of lines changed in a class. The five C&K OO presented metrics are: Lack of Cohesion Of Methods (LCOM), Number Of Children (NOC), Depth of the Inheritance Tree (DIT), Weighted Method per Class (WMC) and Response For a Class (RFC). The four L&H OO presented metrics are: Data Abstraction Coupling (DAC), Message-Passing Couple (MPC), the number of properties (SIZE2) and the Number Of Methods (NOM). In both data sets CHANGE metric counts the number of lines changed per class in the last three years of the project. The above metrics were chosen because they cover important object-oriented concepts like inheritance, cohesion and coupling.

### 3.2 Characteristics of data sets

Table 1 presents the descriptive statistics of the above-mentioned eleven metrics derived from the UIMS and QUES data sets.

Table 1: Descriptives statistics [40]

| | Mean | Median | Std dev | Min | Max | Skewness | Kurtosis |
|---|---|---|---|---|---|---|---|
| | | | *UIMS dataset* | | | | |
| LCOM | 7.49 | 6 | 6.11 | 1 | 31 | 2.49 | 6.86 |
| NOC | 0.95 | 0 | 2.01 | 0 | 8 | 2.24 | 4.28 |
| DIT | 2.15 | 2 | 0.90 | 0 | 4 | -0.54 | 0.09 |
| MPC | 4.33 | 3 | 3.41 | 1 | 12 | 0.73 | -0.70 |
| RFC | 23.21 | 17 | 20.19 | 2 | 101 | 2.00 | 4.94 |
| DAC | 2.41 | 1 | 3.40 | 0 | 21 | 3.33 | 12.87 |
| WMC | 11.38 | 5 | 15.90 | 0 | 69 | 2.03 | 3.98 |
| NOM | 11.38 | 7 | 10.21 | 1 | 40 | 1.67 | 1.94 |
| SIZE1 | 106.44 | 74 | 114.65 | 4 | 439 | 1.71 | 2.04 |
| SIZE2 | 13.97 | 9 | 13.47 | 1 | 61 | 1.89 | 3.44 |
| CHANGE | 46.82 | 18 | 71.89 | 2 | 289 | 2.29 | 4.35 |
| | | | *QUES dataset* | | | | |
| LCOM | 9.18 | 5 | 7.31 | 3 | 33 | 1.35 | 1.10 |
| NOC | 0 | 0 | NA | 0 | 0 | NA | NA |
| DIT | 1.92 | 12 | 0.53 | 0 | 4 | -0.10 | 5.46 |
| MPC | 17.75 | 17 | 8.33 | 2 | 42 | 0.88 | 1.17 |
| RFC | 54.44 | 40 | 32.62 | 17 | 156 | 1.62 | 1.96 |
| DAC | 3.44 | 2 | 3.91 | 0 | 25 | 2.99 | 12.82 |
| WMC | 14.96 | 9 | 17.06 | 1 | 83 | 1.77 | 3.33 |
| NOM | 13.41 | 6 | 12.00 | 4 | 57 | 1.39 | 1.40 |
| SIZE1 | 275.58 | 211 | 171.60 | 115 | 1009 | 2.11 | 5.23 |
| SIZE2 | 18.03 | 10 | 15.21 | 4 | 82 | 1.71 | 3.42 |
| CHANGE | 64.53 | 52 | 43.13 | 6 | 217 | 1.36 | 2.17 |

Both studies [40] and [43] measure the Pearson's correlation coefficients between the CHANGE metric and each of the OO metrics presented in Table 1. The Pearson's correlation coefficients measure the degree of linear dependence between two variables. Table 2 shows that despite the fact that a significant correlation between CHANGE and most of the OO metrics exists, there is a difference between the correlations of the two data sets. Thus, both studies construct different maintainability prediction models since the regard the data sets as being heterogeneous.

Table 2: Correlations between CHANGE and OO metrics [40]

| | Pearson's correlation coefficient | |
|---|---|---|
| | UIMS data set | QUES data set |
| LCOM | 0.568 | 0.050 |
| NOC | 0.559 | NA |
| DIT | -0.433 | -0.090 |
| MPC | 0.454 | 0.461 |
| DAC | 0.629 | 0.083 |
| WMC | 0.646 | 0.425 |
| NOM | 0.635 | 0.142 |
| SIZE1 | 0.626 | 0.635 |
| SIZE2 | 0.666 | 0.149 |

### 3.3 Prediction accuracy measures

In this section, we introduce the criteria for evaluating the prediction accuracy of the models that we have presented earlier in this study.

During last decades several mathematical techniques were used to measure accuracy. The most popular, and also, the ones that we are going to use in this paper are the following: the Absolute Residual (Ab.Res.), the count of the number of predictions within $m$% of the actuals, pred($m$) and the Magnitude of Relative Error(MRE) [24].

Equation 13 shows that Ab.Res is the absolute value of residual given by the difference between the actual value and the predicted value.

$$\text{Ab.Res} = |\text{actual value} - \text{predicted value}| \tag{13}$$

MRE is a normalized measure of the above-mentioned absolute residual defined as in Equation 14.

$$\text{MRE} = \frac{|\text{actual value} - \text{predicted value}|}{\text{actual value}} \tag{14}$$

The MaxMRE measures the maximum relative discrepancy, which is the maximum error relative to the actual value in prediction. The mean magnitude relative error (MMRE) prediction accuracy statistic is the most widely used indicator in the recent years [24]. It is extensively used for assessing the performance of software effort estimation models as defined in Equation 15.

$$\text{MMRE} = \frac{1}{n}\sum_{i=1}^{n} \text{MRE}_i \tag{15}$$

Finally, another widely used prediction quality indicator is Pred($m$), which is a measure of what proportion of the predicted values have MRE less than or equal to a specified value. In this paper we will use 0.25 and 0.3 as reference values for this measure since these values have been used in the studies that introduced the presented prediction models [40], [43]. Pred is given by:

$$\text{Pred}(q) = \frac{k}{n} \tag{16}$$

where $k$ represents the number of the cases that are less or equal to $m$, and $n$ is the total number of cases in the dataset.

The comparison of the five models presented in this study is based upon the above three mathematical techniques (MaxMRE, MMRE, Pred). The values of these techniques are directly comparable since all derived from the distance between the predicted values and the observed values on the two data sets, regardless of the prediction model. Previous studies have set different thresholds for the above-mentioned measurements values in order for a model to be considered accurate. According to [14] MMRE should be $\leq 0.25$ and/or either Pred(0.25)$\geq 0.75$ or Pred(0.30)$\geq 0.70$ [26] and [27].

## 4 Model evaluation and comparison

In this section we analyze the results with respect to the above-mentioned criteria, making a comparison regarding the performance of each model that has been presented in Section 2.

Table 3: Prediction accuracy for the UIMS dataset

| Model | MaxMRE | MMRE | Pred(0.25) | Pred(0.3) |
|---|---|---|---|---|
| BN [43] | **7.039** | **0.972** | **0.446** | **0.469** |
| MARS [40] | 14.06 | 1.86 | 0.28 | 0.28 |
| MLR [40] | 18.88 | 2.70 | 0.15 | 0.21 |
| RT [40] | 24.57 | 4.95 | 0.10 | 0.10 |
| ANN [40] | 19.63 | 1.95 | 0.15 | 0.15 |

Table 4: Prediction accuracy for the QUES dataset

| Model | MaxMRE | MMRE | Pred(0.25) | Pred(0.3) |
|---|---|---|---|---|
| BN [43] | **1.592** | 0.452 | 0.391 | 0.430 |
| MARS [40] | 1.91 | **0.32** | **0.48** | **0.59** |
| MLR [40] | 2.03 | 0.42 | 0.37 | 0.41 |
| RT [40] | 4.82 | 0.58 | 0.41 | 0.45 |
| ANN [40] | 3.07 | 0.59 | 0.37 | 0.45 |

### 4.1 Results from UIMS dataset

Table 3 shows the prediction accuracy measures achieved by each of the maintainability prediction models for the UIMS dataset. As can be clearly observed, both the Bayesian Network and the MARS model provide more accurate predictions than the other three models. Nevertheless, the Bayesian Network appear to be significantly more accurate than the MARS model providing almost two times better results in every measurement. However, none of the models in Table 3 meet the criteria to be considered accurate, according to [14], [26] and [27].

### 4.2 Results from QUES dataset

Table 4 shows the prediction accuracy measures achieved by each of the maintainability prediction models for the QUES data set. In this data set the results are very close in almost every measurement. In contradiction to the UIMS data set, where the Bayesian Network was clearly preferable over the other models, here it is preferable only in cases that we are interested in a low maximum error (worst case scenario), relative to the actual value in prediction. In these cases the Bayesian Model seems to be the safest prediction model whereas it provides the lowest MaxMRE. In all other cases the MARS model provide the most accurate predictions. Similar to the UIMS data set none of the models can be considered accurate, however, all provide better prediction accuracy for the QUES data set.

## 5 Conclusions

In this paper we presented two recently introduced maintainability prediction models: the first one based on a Bayesian Network and the second based on Multivariate Adaptive Regression Splines, along with three other pre-existing models. Both models were built on the Li and Henry's data set. The results of both models were compared with those of the Multivariate Linear Regression prediction model, the Regression Tree prediction model and the Artificial Neural Network prediction model. Although none of the maintainability prediction models proposed so far satisfy the criteria of an accurate prediction model [14], [26], [27], the Bayesian Network model offers significantly more accurate predictions in the UIMS data set than other models followed by the MARS model. In the QUES data set the MARS model provide the most accurate prediction measure while the Bayesian network is reasonably competitive and tends to provide measures close to the other most accurate prediction models. The above findings show that there is no uniformly optimal solution, but the accuracy of the maintainability prediction models may vary depending on the data set.

Predicting the maintainability of a software system is an open research issue and demands further investigation. A limitation of the study is that all models are based on the metrics of two data sets derived from software written in Java. This provides an interesting direction for future work where models can be validated on several data sets from software systems written in different OO programming languages.

## Acknowledgements

We would like to thank Dr. Apostolis Ampatzoglou and the coordinators of the Student Colloquium course for their valuable suggestions and guidance that they offered. We would also like to thank our fellow students for reviewing our paper and providing us with feedback which allowed us to improve the quality of this paper.

## References

[1] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[2] Predicting maintenance performance using object-oriented design complexity metrics. 2003.

[3] Y. Ahn, J. Suh, S. Kim, and H. Kim. The software maintenance project effort estimation model based on function points. *Journal of Software Maintenance*, 15(2):71–85, Mar. 2003.

[4] A. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *Software Engineering, IEEE Transactions on*, SE-9(6):639–648, Nov 1983.

[5] P. Bhatt, W. K, G. Shroff, and A. K. Misra. Influencing factors in outsourced software maintenance. *SIGSOFT Softw. Eng. Notes*, 31(3):1–6, May 2006.

[6] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1981.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[8] L. Briand, C. Bunse, and J. Daly. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *Software Engineering, IEEE Transactions on*, 27(6):513–530, Jun 2001.

[9] L. Briand, W. Melo, and J. Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *Software Engineering, IEEE Transactions on*, 28(7):706–720, Jul 2002.

[10] L. C. Briand, B. Freimut, and F. Vollei. Using multiple adaptive regression splines to support decision making in code inspections. *Journal of Systems and Software*, 73(2):205 – 217, 2004. Applications of statistics in software engineering.

[11] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, Jun 1994.

[12] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, Aug 1994.

[13] D. Coleman, B. Lowther, and P. Oman. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29(1):3 – 16, 1995. Oregon Metric Workshop.

[14] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.

[15] V. Dave and K. Dutta. Neural network based models for software effort estimation: a review. *Artificial Intelligence Review*, pages 1–13, 2012.

[16] E. Deconinck, Q. Xu, R. Put, D. Coomans, D. Massart, and Y. V. Heyden. Prediction of gastro-intestinal absorption using multivariate adaptive regression splines. *Journal of Pharmaceutical and Biomedical Analysis*, 39(5):1021 – 1030, 2005.

[17] F. Fioravanti and P. Nesi. Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *Software Engineering, IEEE Transactions on*, 27(12):1062–1084, Dec 2001.

[18] J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 03 1991.

[19] K. Furulund and K. Molokken-Ostvold. Increasing software effort estimation accuracy using experience data, estimation models and checklists. In *Quality Software, 2007. QSIC '07. Seventh International Conference on*, pages 342–347, Oct 2007.

[20] M. Genero, J. Olivas, M. Piattini, and F. Romero. Using metrics to predict oo information systems maintainability. In K. Dittrich, A. Geppert, and M. Norrie, editors, *Advanced Information Systems Engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 388–401. Springer Berlin Heidelberg, 2001.

[21] J. C. Granja-Alvarez and M. J. Barranco-Garca. A method for estimating maintenance cost in a software project: A case study. *Journal of Software Maintenance: Research and Practice*, 9(3):161–175, 1997.

[22] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1):33–53, Jan 2007.

[23] Z. Kaiqi, Z. Xiaoyue, and L. Ying. Multivariate linear regression model based on fuzzy variable in econometrics. In *Artificial Intelligence and Computational Intelligence, 2009. AICI '09. International Conference on*, volume 4, pages 111–114, Nov 2009.

[24] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure [software estimation]. *Software, IEE Proceedings -*, 148(3):81–85, Jun 2001.

[25] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2):111 – 122, 1993. Object-Oriented Software.

[26] A. D. Lucia, E. Pompella, and S. Stefanucci. Assessing effort estimation models for corrective maintenance through empirical studies. *Information and Software Technology*, 47(1):3 – 15, 2005.

[27] S. G. MacDonell. Establishing relationships between specification size and software process effort in {CASE} environments. *Information and Software Technology*, 39(1):35 – 45, 1997.

[28] E. Mendes, N. Mosley, and S. Counsell. The application of case-based reasoning to early web project cost estimation. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, pages 393–398, 2002.

[29] S. Misra. Modeling design/coding factors that drive maintainability of software systems. *Software Quality Journal*, 13(3):297–320, 2005.

[30] K. Murphy. A brief introduction to graphical models and bayesian networks, 1998.

[31] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[32] P. Oman and J. Hagemeister. Construction and testing of polynomials predicting software maintainability. *J. Syst. Softw.*, 24(3):251–266, Mar. 1994.

[33] S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114 – 132, 2007. Network and Information Security: A Computational Intelligence Approach Network and Information Security: A Computational Intelligence Approach.

[34] N. Pizzi and W. Pedrycz. Predicting qualitative assessments using fuzzy aggregation. In *Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual meeting of the North American*, pages 267–272, June 2006.

[35] T.-S. Quah and M. M. T. Thwin. Application of neural networks for software quality prediction using object-oriented metrics. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 116–125, Sept 2003.

[36] M. Riaz, E. Mendes, and E. Tempero. A systematic review of software maintainability prediction and metrics. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 367–377, Oct 2009.

[37] R. Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, Berlin, New-York, 1996.

[38] K. Shibata, K. Rinsaka, T. Dohi, and H. Okamura. Quantifying software maintainability based on a fault-detection/correction model. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pages 35–42, Dec 2007.

[39] S. M. Tor Ivry. License plate number recognition using artificial neural network, 1998.

[40] C. van Koten and A. Gray. An application of bayesian network for predicting object-oriented software maintainability. *Information and Software Technology*, 48(1):59 – 67, 2006.

[41] C. Yonghui. Study of the bayesian networks. In *E-Health Networking, Digital Ecosystems and Technologies (EDT), 2010 International Conference on*, volume 1, pages 172–174, April 2010.

[42] M. V. Zelkowitz. Perspectives in software engineering. *ACM Comput. Surv.*, 10(2):197–216, June 1978.

[43] Y. Zhou and H. Leung. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software*, 80(8):1349 – 1361, 2007. The Impact of Barry Boehms Work on Software Engineering Education and Training.

[44] Y. Zhou and B. Xu. Predicting the maintainability of open source software using design metrics. *Wuhan University Journal of Natural Sciences*, 13(1):14–20, 2008.

# An overview of available cloud task scheduling algorithms

Fokko Driesprong, Mohamed El Sioufy

**Abstract**—Task scheduling plays a key role in cloud computing systems. Though several scheduling algorithms have been proposed, it is often the case that each approached the problem from its authors' perspective. The generality and lack of standard definition for cloud computing as well as various usage and deployment scenarios allowed researchers to add customized assumptions and constraints to their scheduling models and prefer specific requirements over the others. Moreover, being NP-Complete problem directed researchers to propose heuristic algorithms that finds a 'reasonably good' schedule rather than evaluate all possible schedules. In this paper we give an overview on the problem of task scheduling in the context of IaaS clouds. Moreover, we present a variety of cloud scheduling algorithms highlighting their main features, concerns and objectives. Through our work, we clarify possible reasons behind the diversity of available algorithms.

## 1 Introduction

Task scheduling is the process by which threads, processes or data flows, which represent user tasks, are given access to system resources such as processor time, memory and communications bandwidth [7]. In contrast to traditional computing paradigms, cloud computing has a dynamic infrastructure [19], where the same set of resources, physical servers which forms the cloud, could be virtualized in several possible ways. Using existing virtualization techniques, a single physical server could be sliced into multiple small partitions [20]. Similarly, a virtual super computer could be created by aggregating resources from multiple servers. The different nature of cloud computing prevents the employment of traditional scheduling algorithms [16] and makes the process of cloud scheduling more complicated.

Analogous to other cloud computing techniques, scheduling aims to promote relevant requirements of both cloud providers and consumers. However, in response to the generality and lack of standard definition for cloud computing as well as its various usage and deployment scenarios researchers are obliged to add customized assumptions and constraints to their scheduling models, and favour requirements that suits their considered cloud environments. Moreover, being NP-Complete problem directed researchers to propose heuristic algorithms that approximate the ideal schedule rather than evaluate all possible schedules, which is very costly in terms of computational time [6].

Scheduling is one of the major activities performed in all computing environments [4, 16]. It is considered of paramount importance in cloud computing because users are only charged for the resources they use on a pay per use basis, hence efficient utilization of such resources is a must for the sake of providers. This could be highly achieved by an efficient scheduling mechanism [16]. With the vast number of scheduling algorithms proposed in literature, readers might be confused from their diversity and different considerations. Specially, it is the case that sometimes the authors do not mention the cloud environment which their scheduling algorithm suits the most. In this paper we give an overview on the problem of task scheduling in the context of IaaS clouds. We present a selection of varying cloud scheduling algorithms and highlight their main features, concerns and objectives. Throughout the paper we clarify possible reasons behind the diversity of available scheduling algorithms.

In section 2 we give an overview on cloud computing as well as the processes oing. We end the section by providing a task scheduling model that is general enough to compensate the properties and considerations of the selected algorithms. In section 3 we describe each of our selected algorithms and highlighting their main features, concerns and objectives. Finally, in section 4, we conclude our work.

## 2 Overview

Cloud computing is a general term for anything that involves delivering hosted services over the Internet [1]. Cloud services can be broadly divided into three categories: Software as a Service (Saas), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [15]. Using the online office suite Google Docs, the Azure platform for hosting your applications or renting a Virtual Machine from Amazon EC2 are different cloud usage scenarios that corresponds to the mentioned services respectively.

Several definitions with different perceptions for cloud computing exists in literature [1, 12]. The main reason for that is because cloud computing is not a new technology, but rather a new model that brings together a set of existing technologies to develop and execute applications in a different way [24]. In this section we give an overview on cloud computing from a scheduling perspective, then we explain the processes of task and resource scheduling in details. We end this section by providing a task scheduling model that is general enough to compensate the selected algorithms.

- *Fokko Driesprong, MSc. Student, Software Engineering and Distributed Systems, University of Groningen, E-mail: f.t.driesprong@student.rug.nl.*
- *Mohamed El Sioufy, MSc. Student, Software Engineering and Distributed Systems, University of Groningen, E-mail: m.el.sioufy@student.rug.nl.*

### 2.1 Cloud Computing

Cloud Computing is a computational paradigm that enables provisioning of computational resources as services to end users over the Internet. Computational resources are often owned by third-party 'resource providers', lives in one or more data centers and are dynamically configured and provisioned by means of virtualization. In a Cloud Computing environment, the end user could practically request as much resources as needed, and is charged on a pay per use basis. Guarantee of Quality of Service is given through a Service Level Agreement (SLA), which formally defines the quality of the service or performance [19].

Since the popularization of Cloud Computing term in 2007, with IBM Blue Cloud [21], many organizations have invested in Cloud Computing, and to date more than 400 datacenters exists around the globe [11]. On the side of the consumers, Cloud Computing allows reduction or elimination of costs associated with the internal infrastructure, which is required for provisioning of their services. Moreover, as their computing requirements changes, the option of both scaling up or down is made available permanently and could be done instantly without associated costs [3, 18]. This opportunity of cost reduction and dynamic infrastructure makes Cloud Computing an attractive alternative for consumers, especially for business initiatives. A recent study stated that cloud adoption continued to rise in 2013, with 75 percent of those surveyed reporting the use of some sort of cloud platform, up from 67 percent last year [17]. Such growth on both sides is motivated by the ongoing consolidation as well as by the revenues that customers and operators are observing with Cloud Computing [12].

Cloud Providers have a stimulating return on investment from the infrastructure supporting the cloud. Since it allocates resources for several users concurrently, the model benefits from a form of statistical multiplexing [5]. This is guaranteed through several decades of research in areas, such as: distributed computing, grid computing, web technologies, service computing, and virtualization [12]. In order to attain maximum profit, cloud providers need to achieve a high level of resource utilization, which in turn gives higher availability to serve more users.

### 2.2 Task and Resource Scheduling

Scheduling of tasks is a critical issue in Cloud Computing, and has received lot of attention in recent years. Cloud computing has a dynamic infrastructure which enforces additional burden to the traditional process of scheduling, this corresponds to resource allocation [13].

In order to make a clear distinction between task scheduling and resource allocation we provide the following scenario. This also gives the user an intuition about how different requirements could affect the mechanics of these two processes. For the sake of simplicity, we consider RAM as a representation of computational resources. Consider a data center with two physical servers, each of which has 2GB of RAM. Using existing virtualization techniques these servers could be provisioned as virtual machines in several possible ways. Consider that at a point of time with the two servers being idle, 4 independent tasks has arrived, each of which has RAM requirements of 512MB.

In cloud computing, it is not the case that arriving tasks are directly scheduled to run on physical resources. First, physical resources are allocated as virtual machines, this corresponds to the process of resource allocation, then tasks are scheduled to run on these virtual machines, this corresponds to task scheduling. Consider the following three different resource and task scheduling schemes.

**Resource Under-utilization** In this case, four virtual machines are allocated each of which has 1 GB of RAM, as given in Table 1. Each task is mapped on one of the virtual machines. These virtual machines consume all the computing power of the datacenter, and are underutilized. Despite the fact that available physical resources could accommodate more tasks, any task that arrives while the introduced tasks are being processed has to wait.

| $ID_{Server}$ | $Ram_{Server}$ | $ID_{Vm}$ | $Ram_{Vm}$ |
|---|---|---|---|
| 1 | 1GB | 1 | 1GB |
| 1 | 1GB | 2 | 1GB |
| 2 | 1GB | 3 | 1GB |
| 2 | 1GB | 4 | 1GB |
| Net RAM Allocated: | | 4GB | |

Table 1. Resource Allocation and Task Scheduling scheme that leads to underutilization of the datacenters' physical resources.

**Energy Efficiency vs Reliability** A wiser decision is to allocate 4 virtual machines, each of which has the same RAM requirements of the arriving tasks, so that there is no underutilization of resources, this scenario is given in Table 2. However, a decision about which set of physical resources should be allocated still remains. In this particular situation either server 1 or server 2 could solely provide the required virtual machines leaving the other server idle. This could lead to more energy efficiency since half of the required energy is being consumed, assuming the idle server is shutdown. From a different perspective, placing the entire load to a single server increases its probability of failure. Thus using both servers in allocation could be a more reliable decision. Note that in both cases, a net of 1GB of RAM is still idle and can be used to serve newly arriving tasks.

| $ID_{Server}$ | $Ram_{Server}$ | $ID_{Vm}$ | $Ram_{Vm}$ |
|---|---|---|---|
| 1 | 512MB | 1 | 512MB |
| 1 | 512MB | 2 | 512MB |
| 1 (2) | 512MB | 3 | 512MB |
| 1 (2) | 512MB | 4 | 512MB |
| Net RAM Allocated: | | 2GB | |

Table 2. Resource Allocation and Task Scheduling scheme(s) that achieves higher level of resource utilization and availability.

**Maximizing Resource Utilization** In some cases arriving tasks could accept sharing resources. To take advantage of this, allocated virtual machines will not have explicit access to the actual physical resources but share them with other virtual machines. This will lead to achieving higher levels of resource utilization, as given in Table 3.

| $ID_{Server}$ | $Ram_{Server}$ | $ID_{Vm}$ | $Ram_{Vm}$ |
|---|---|---|---|
| 1 | 256MB | 1 | 512MB |
| 1 | 256MB | 2 | 512MB |
| 1 (2) | 256MB | 3 | 512MB |
| 1 (2) | 256MB | 4 | 512MB |
| Net RAM Allocated: | | 1GB | |

Table 3. Resource Allocation and Task Scheduling scheme that make use of arriving tasks requirements to achieve higher level of resource utilization .

As it could be seen, the process of cloud task scheduling is much more different from scheduling as known in traditional computing paradigms. This arises from the dynamic infrastructure of cloud computing. In a small scenario like the one provided, several decisions could have been taken, each of which has its own advantages, disadvantages and operability. For example, the third scheduling scheme can only be utilized if arriving tasks accepts sharing resources. In the second scenario, further decision needs to be taken that has to compromise between reliability and energy efficiency. Keep in mind that to keep these examples as simple, we did not consider other contradicting requirements. For example, in the second scheduling scheme, turning off a server instance to save energy, could affect the users' response time, since newly arriving tasks has to wait more time for the shutdown instance to run. In the same sense, the time taken for creation and running of virtual machines could be eliminated by doing that in advance i.e. prior to the arrival of tasks. However, this could lead to unsuitability of the pre-allocated virtual machines for the arriving tasks. This could be considered the case in the first scheme.

In the context of cloud computing, scheduling is generally concerned with mapping of user tasks to available virtual machines. These virtual machines are dynamically (re)allocated; this corresponds to the process of resource allocation. Task scheduling and resource allocation are two different issues, however, considering both simultaneously results in higher system performance [2, 9]. Figure 1 provides a visual aid to our description for resource allocation and task scheduling.

## 2.3 Problem Definition and Modeling

In this section we provide a comprehensive model that describes the problem of task scheduling. This model is based on the model provided in [22] however, some additions and modifications were essential to accommodate assumptions, constraints and considerations of the presented algorithms.

In this model we overlook the process of resource allocation. Despite its importance relative to task scheduling, we consider that dynamic resource (re)-allocation is being done in isolation, however, it is the case that some of the presented algorithms considers and manages this process internally. We believe this would elevate reasonable complexity from the model and will keep the user focused on task scheduling. Finally, we consider IaaS Clouds, where cloud resources in terms of processing power, communication bandwidth and storage are provided to the end users as services.

Services: Based on IaaS service model, cloud resources are encapsulated into services, these are provisioned to the end user over the Internet to fulfill his tasks. In our model these services corresponds to actual virtual machine instances, that

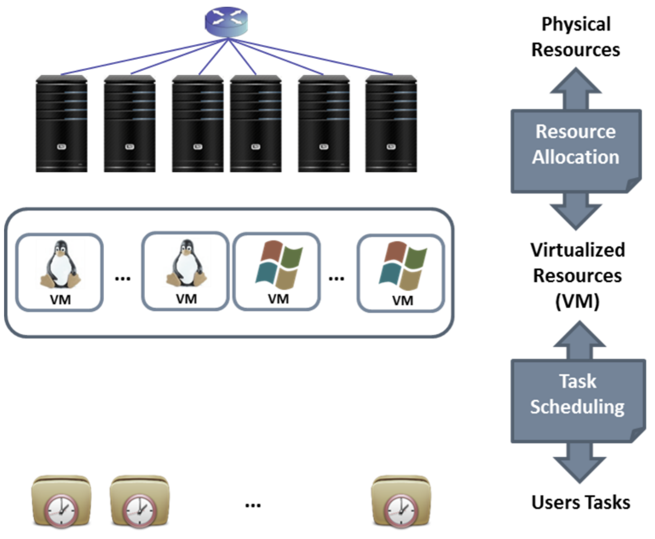


Figure 1. General Overview for Cloud Resource Allocation and Task Scheduling

are results of the process of dynamic resource (re)allocation.

Let $S = \{s_1, s_2, ...s_m\}$ be a service array, which specifies all the current allocated virtual machines. In the simplest form each service or VM instance is characterized by a service ID $S_{ID}$, computational ability $S_{Ability}$ and availability $S_{Availability}$. These are defined as following:

$S_{Id}$**:** a unique id that identifies a service.

$S_{Ability}$**:** the computational ability of the allocated virtual machine. This is expressed in terms of processing power, communication bandwidth and storage. $S_{Ability} = \{$processing power, communication bandwidth, storage$\}$.

$S_{Availability}$**:** specifies the time left until this service is ready. $S_{Availability} = t$, where t is the time remaining until the service will be ready to accept and start processing a task. When $t = 0$ implies the service is ready, $t \geq 0$ implies that the service is currently processing a task and will finish in $t$ seconds.

Tasks: Users submit their resource requirements as well as additional constraints through a variety of available cloud applications established by the cloud service providers. A task is the container that holds these requirements and could be mapped to any of the available services. Processing time of a task is service dependent. In the simplest form a task is specified by task ID $T_{Id}$, user type $T_{User}$, arrival time $T_{Arrival}$, task length $T_{Length}$, deadline $T_{Timeout}$, resource requirements $T_{Requirements}$, task dependencies $T_{Dependencies}$ and task state $T_{State}$.

$T_{Id}$**:** an unique id that identifies a task

$T_{User}$**:** an object that represents the task user. This also contains Quality of Service information.

$T_{Arrival}$**:** the time at which the task has been received in the system.

$T_{Length}$**:** expected computing workload of the task.

$T_{Timeout}$: maximum time the user is willing to wait, otherwise the task in considered as failed.

$T_{Requirements}$: task computational requirements. Corresponding to the aforementioned service attribute $S_{Ability}$, $T_{Requirements}$ = {processing power, communication bandwidth, storage, explicit?}, where explicit? is a flag that specifies if this task requires explicit usage of resources or not. Its obvious that a task could only be mapped to a service with either same or higher computational ability.

$T_{Dependencies}$: dependency array that specifies the task dependencies. $T_{Dependencies} = \{T1_{ID}, T2_{ID}, ...Tn_{ID}\}$. A task cannot be processed until all the tasks in its dependency array are processed.

$T_{State}$: the current task state. This includes idle, suspending, mapping, executing and completed.

## 3 Algorithms

In this section we provide a brief description to each of our selected algorithms. We considered the selection of a set of algorithms that reflects the diversity of concerns and objectives of the authors rather than the selection of a similar set of algorithms to compare between. Our focus is not to conclude which algorithm is the best, but to demonstrate that different scheduling algorithms suits different cloud environments and that different perceptions to the term of cloud computing often leads to different considerations.

We present each algorithm by describing it with respect to; Considered Cloud Environment, Motivation, Algorithm, Special Settings and Results. Each is described as following:

- Considered Cloud Environment: The cloud environment addressed by the authors.
- Motivation: The motivation of the authors, what they think is the current state and what should be done. This could elucidate their considerations and target objectives.
- Algorithm: The proposed scheduling algorithm, its mechanics, considerations and objectives.
- Special Settings: Any specific technology, or services integrated within the cloud environment.
- Results: Here we did not present quantitative numbers but focused on how did the authors assessed their scheduler and its performance in general.

### 3.1 Linear Scheduling Strategy for Resource Allocation in Cloud Environment [2]

**Considered Cloud Environment** The authors considered IaaS clouds, where the combination of Nimbus and Cumulus services are imported to a server node to establish the IaaS cloud environment.

**Motivation** The authors explicitly stated two powerful reasons for their direction towards resource utilization. (1) Cloud demand and cloud resource utilization are factors that most of the IT industries and other organization will demand the most in future. While, emerging cloud computing techniques focus on scalability and availability, resource utilization and management needs similar attention. The allocation of resources must be made efficiently that maximizes the system utilization and overall performance. (2) An important notifying advantage of infrastructure-as-a-service (IaaS) clouds is that it provides users on-demand access to resources. To provide on-demand access, cloud providers must either significantly overprovision their infrastructure such as paying a high price for operating resources with low utilization or to reject a large proportion of user requests in which case the access is no longer on-demand. At the same time, the important concept is that not all users require truly on demand access to resources of IaaS.

**Algorithm** The authors provided a 'best fit' scheduling algorithm which performs task and resource scheduling respectively. The main objective of the algorithm was to maximize the system throughput and resource utilization. The algorithm also focuses on eradication of starvation and dead lock conditions. The scheduling algorithm is carried out based on the prediction that the initial response to the request is made only after collecting the resource for a finite amount of time, say 1 day or 1 hour, but not allocating the resource as they arrive. The dynamic allocation could be carried out by the scheduler dynamically on request for additional resources. The authors emphasized the importance of considering both task and resource scheduling to result in maximum resource utilization and high performance.

**Special Settings** The authors considered importing the combination of Nimbus and Cumulus services to a service (master) node to establish the IaaS cloud environment and KVM/Xen virtualization along with their proposed least slack time rate first (LSTR) scheduling to manage resource allocation and task scheduling. A mentioned rational behind their consideration was: The cloud environment embedded with the nimbus and cumulus services will contribute more in making the responsibility of resource utilization in Cloud Computing.

**Results** Experimental analysis was made by forming a cloud with the service node to control all client requests that are collected between a pre-defined time intervals. The result analysis of their implementation specifies the amount of resource utilized.

### 3.2 A Task Scheduling Algorithm based on QoS-Driven in Cloud Computing [22]

**Considered Cloud Environment** Not explicitly mentioned, however, IaaS cloud could be inferred.

**Motivation** Quality of service is an inevitable issue that needs to be dealt with. In Cloud Computing it's a key issue how to dispatch efficiently and reasonably the tasks of users to different resources according to the Quality of Service requirements of both cloud providers and consumers. This belongs to task scheduling.

**Algorithm** The authors proposed a 'list scheduling' algorithm based on QoS-driven in Cloud Computing. The algorithm starts by computing a priority for each task, sort tasks accordingly and schedule the sorted tasks in order. Task priorities are computed based on user privilege, task urgency, task workload and task waiting time, where the latter allows task priorities to change dynamically and prevents starving to death conditions. For the actual scheduling, the algorithm evaluates the completion time of each task on all of the available services, and schedules tasks on services which lead to minimum completion time. The algorithm ensures that (1) the task with the highest priority is scheduled first (2) a task should be completed as soon as possible. In their model the authors considered only independent tasks.

The authors adopt what they call 'dynamic batching mode' where tasks are not mapped onto services as they arrive but are collected in a set that is examined for mapping. According to the collected information including the requirements of all the tasks and real time state of all the services, more reasonable scheduling strategy to deal with QoS is achieved.

**Special Settings** No special technologies or requirements were mentioned as mandatory for the algorithm to operate.

**Results** To evaluate their algorithm, the authors developed an extensive simulation platform based on CloudSim 2.1. In their evaluation, makespan, average latency and a load balancing factor were used for assessment. The algorithm was evaluated against a Min-Min algorithm provided in [8] and a Job Scheduling Algorithm based on Berger Model provided in [23]. The experimental results showed that the algorithm achieves good performance and load balancing by QoS driving both priority and completion time. Quantitative results were provided.

**Proposed Improvement** The proposed scheduling algorithm seems very costly in terms of processing, especially if the number of tasks and services is relatively large. Since the two main processes – calculating tasks priorities and evaluating completion time of each task on all available services - performed in the algorithm are independent; performing them simultaneously - in a distributed way - would result in a faster scheduling decision.

## 3.3 Policy-based scheduling of cloud services [14]

**Considered Cloud Environment** The authors were concerned with scheduling of cloud resources among multiple collaborating cloud users. This could correspond to scheduling of cloud resources used in development process for software as a service, where hardware/software resources are scheduled for multiple collaborating software engineering teams.

**Motivation** One role of Clouds is to influence the development process of software as a service. Currently, software engineering practices are far from ideal, and often the quality of the final product suffers. Complexity of software development process is still very high, and many processes are still manually executed. One of these processes is scheduling. Manual scheduling usually leads to non-optimal usage of resources and it is something that should be avoided.

**Algorithm** The authors proposed a domain independent policy-based scheduling mechanism for cloud services. The algorithm schedules tasks based on specified policies encapsulated within the tasks resource request. Examples of such policies can be that a task needs to run for three days consecutive days, or that a task has to be executed every two weeks. The algorithm aims to achieve optimal resource utilization with respect to a total usage of cloud resources in a predefined time interval. The provided scheduling algorithm takes into account dependencies between individual services, and can enforce common use of shared resources. In addition to optimization, the scheduler provides fair scheduling for multiple collaborating cloud users that have highly demanding requests for cloud resources. In a glance, the algorithm will try to find an optimal way to come up with a schedule that will utilize the available cloud resources and comply to the policies encapsulated within the requests.

Due to the complexity and time consumption of cloud scheduling in general, and thus the proposed algorithm, the authors proposed a dynamic relaxation scheme for the optimality requirement. This scheme is enforced as the number of requests exceeds a certain threshold. Following this scheme, requests are split into several groups, and the scheduler schedules each group in turn considering already scheduled groups. Such scheme follows a greedy approach which always makes the choice that looks best at that moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution [10]. In this scheme, great benefit is achieved by the consideration of shared resource access. Situations where a request in the currently processed group demands resources that could be shared with those in previous groups are always detected, and preference is given to timeslots that allow for maximum sharing of resources with already scheduled request.

**Special Settings** In their model, the authors used Apache Whirr at the heart of their deployment service. As well, ZooKeeper was used as a distributed configuration service.

**Results** The scheduler performance was evaluated and it has been shown that it scales well for a typical size of the resource allocation problems they consider. It was also ensured that the Scheduler can sustain a certain level of demand increase, and remain practical for higher number of requests.

**Other** The paper has a pragmatic approach and does not only focus on the scheduler itself, but rather provides a framework that could be implemented in their considered environments, for example, a software development company may have an own private cloud.

## 3.4 Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm [18]

**Considered Cloud Environment** IaaS Clouds.

**Motivation** (1) Operators of so-called Infrastructure-as-a-Service (IaaS) clouds, like Amazon EC2, let their customers allocate, access, and control a set of virtual machines which run inside their data centers and only charge them for the period of time the machines are allocated. Therefore, work flow management on cloud computing becomes more important, when many tasks are sent to cloud environment at the same time. (2) Researches on specification and scheduling of work flows have concentrated on temporal and causality constraints, which specify existence and order dependencies among tasks. However, another set of constraints that specify resource allocation is also equally important.

**Algorithm** The authors proposed an Improved Differential Evolution Algorithm (IDEA) that combines the Taguchi method and a Differential Evolution Algorithm in order to optimize task scheduling and resource allocation. The DEA had a powerful global exploration capability on macro-space and uses fewer control parameters. The systematic reasoning ability of the Taguchi method was used to exploit the better individuals on micro-space to be potential off spring. By this, the proposed IDEA is well enhanced and balanced on exploration and exploitation.

The authors imposed a multi-objective optimization based on proposed time and cost models. The cost model includes processing and receiving costs, while the time model incorporates receiving, processing and waiting time. The multi-objective optimization approach was applied to find the Pareto optimal of total cost and make span.[1]

In their model, the authors assumed no sharing of resource among tasks. That is, a task cannot be pre-empted once it acquires a resource, in the same sense, a resource can't perform on more than one task simultaneously. The authors considered dependent and independent tasks.

**Special Settings** To operate, this algorithm needs to be parametrized with the cost models of the particular service, which can be Amazon, Heroku and similar.

**Results** Performance of the proposed IDEA was assessed in two case studies, which included five-task five-resource and ten-task ten-resource problems and confirmed its high effectiveness and easy optimization.

## 4 Conclusion

Scheduling is one of the major activities performed in all computing environments [4, 16]. Cloud Computing has a dynamic infrastructure and thus additional burden is required to develop more efficient schedulers, this corresponds to resource allocation. Considering task scheduling and resource allocation simultaneously results in higher levels of resource utilization and improved system performance [2, 9].

Several definitions and different perceptions exists for cloud computing [1, 12], this allowed researchers to add customized assumptions and constraints when developing their scheduling models, and favour different requirements that suits their considered cloud environment. For example, a scheduling algorithm provisioned for a private cloud with limited infrastructure, should emphasize more on resource utilization. Analogous, being NP-Complete problem, directed researchers to seek near optimal solutions rather than evaluate all possible schedulers [6]. Some authors' proposed dynamic relaxation schemes for their optimizations that are imposed as the complexity of their original algorithms is expected to increase, i.e. as the number of tasks – resources being scheduled increase [14]. Similarly, others considered imposing multi-objective optimizations to find pareto-optimal schedulers [18].

In this paper, we formalized the problem of task scheduling in IaaS clouds. We gave the user an intuition on a selection of available scheduling algorithms. In our selection we considered algorithms that reflect the diversity of concerns and objectives of the authors and suits different cloud environments.

[1]Pareto-optimal solution is when all objectives are met within a certain extent. No specific objective is dominant, but all objectives are met as much as possible, even within the situation of conflicting objectives.

## Acknowledgement

The authors would like to thank dr. Alexander Lazovik and drs. Faris Nizamic for reviewing this paper as experts in the field and for their advice and guidance along the student colloquium.

## References

[1] Survey on cloud computing. *International Journal of Computer Trends and Technology*, 4(9), September 2013.

[2] S Abirami and Ramanathan Shalini. Linear scheduling strategy for resource allocation in cloud environment. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, 2, 2012.

[3] Amazon. Elastic compute cloud, 2013. `http://aws.amazon.com/ec2/`.

[4] Shalmali Ambike, Dipti Bhansali, Jaee Kshirsagar, and Juhi Bansiwal. An optimistic differentiated job scheduling system for cloud computing.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[6] S Arora and L Hochbaum. *Approximation Algorithms for NP-Hard Problems*. Course Technology, 1996.

[7] R.H. Arpaci-Dusseau and A.C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. 2012.

[8] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Lasislau L. Bölöni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 61(6):810–837, June 2001.

[9] Z. Chenhong, Z. Shanshan, L. Qingfeng, X. Jian, and H. Jicheng. Independent tasks scheduling based on genetic algorithm in cloud computing. In *5th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2009*, September 2009.

[10] T.H. Cormen. *Introduction to Algorithms*. Mit Press, 2009.

[11] DataCenterData. Datacenterdata, 2014. `https://www.datacenterdata.com/datacenter`.

[12] Glauco Gonçalves, Patrícia Endo, and Thiago Damasceno. Resource allocation in clouds: Concepts, tools and research challenges. pages 197–240, 2008.

[13] Tarun Goyal and Aankanksha Agrawal. Host scheduling algorithm using genetic algorithm in cloud computing environment. *International Journal of Research in Engineering and Technology (IJRET)*, 1(1):7–12, 2013.

[14] Faris Nizamic, Viktoriya Degeler, Rix Groenboom, and Alexander Lazovik. Policy-based scheduling of cloud services. *Scalable Computing: Practice and Experience*, 13(3), 2012.

[15] Buyya Rajkumar, Broberg James, and Goscinski Andrzej. *Cloud Computing: Principles and Paradigms (Wiley Series on Parallel and Distributed Computing)*. Wiley, 2011.

[16] Pinal Salot. A survey of various scheduling algorithm in cloud computing environment.

[17] Michael Skok. 2013 future of cloud computing 3rd annual survey results, 2013. `http://www.mjskok.com/resource/`

`2013-future-cloud-computing-3rd-annual\`
`-survey-results`.

[18] Jinn-Tsong Tsai, Jia-Cen Fang, and Jyh-Horng Chou. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. *Comput. Oper. Res.*, 40(12):3045–3055, December 2013.

[19] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[20] VMware. Vmware virtualization and cloud management solutions: Managing it in the cloud era. Technical report, 2011.

[21] M.A. Vouk. Cloud computing: Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31–40, June 2008.

[22] Xiaonian Wu, Mengqing Deng, Runlian Zhang, Bing Zeng, and Shengyuan Zhou. A task scheduling algorithm based on qos-driven in cloud computing. *Procedia Computer Science*, 17(0):1162 – 1169, 2013. First International Conference on Information Technology and Quantitative Management.

[23] Baomin Xu, Chunyan Zhao, Enzhao Hu, and Bin Hu. Job scheduling algorithm based on berger model in cloud environment. *Advances in Engineering Software*, 42(7):419 – 425, 2011.

[24] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, April 2010.

university of groningen

faculty of mathematics and natural sciences

computing science