Proceedings of the 10th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering

Christoph Meinel, Hasso Plattner, Jürgen Döllner,
Mathias Weske, Andreas Polze, Robert Hirschfeld,
Felix Naumann, Holger Giese, Patrick Baudisch,
Tobias Friedrich, Emmanuel Müller (Eds.)

Technische Berichte Nr. 111

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 111

Christoph Meinel | Hasso Plattner | Jürgen Döllner | Mathias Weske |
Andreas Polze | Robert Hirschfeld | Felix Naumann | Holger Giese |
Patrick Baudisch | Tobias Friedrich | Emmanuel Müller (Eds.)

Proceedings of the 10th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2017
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URN urn:nbn:de:kobv:517-opus4-100260
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-100260

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-390-9

Contents

Multi-objective Optimization for Biochip Design

Mirela Alistar

Human Computer Interaction group
Hasso-Plattner-Institut
Mirela.Alistar@hpi.uni-potsdam.de

Biochips are arrays of electrodes on a printed circuit board that can transport and process droplets, such as performing an in-vitro diagnosis on a droplet of human blood. Low-cost biochips hold the promise of putting the abilities to perform such tests into the hands of millions of users, thereby democratizing parts of health care. Unfortunately, designing such systems, including biochips, for a specific biochemical process ('bio-protocol') requires substantial expertise, especially since it requires optimizing for multiple objectives, such as cost and execution time. To allow non-experts to design custom biochips, we have implemented an algorithm that automates the design process, making our system accessible for a wider audience.

Our algorithm obtained a continuous solution space that allows selection among trade-off alternatives for biochip designs. When compared to the related work, our algorithm performed up to 125 % better in terms of execution time, and up to 76 % better in terms of cost.

As future work, we will extend our system to include human editing.

1 Overview

Biochips are electronic devices that can perform the tasks traditionally performed by a human wet lab technician with a pipette [18], such as an in-vitro diagnosis that identifies microbes in human blood.

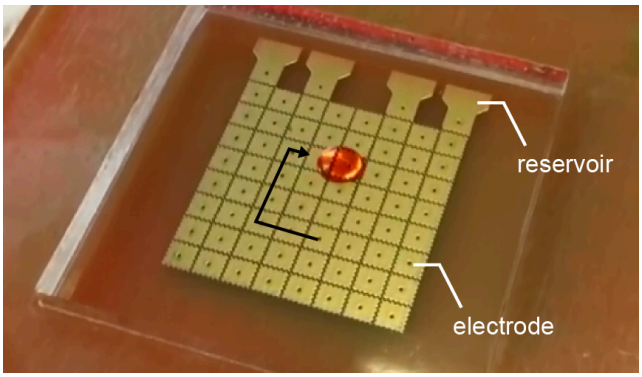


Figure 1: A biochip with four reservoirs and 8x8 electrodes holding a single droplet. The biochip can move the droplet by applying a voltage to a neighboring electrode.

As shown in Figure 1, biochips consist of arrays of electrodes, with each electrode capable of holding a droplet. To move a droplet, biochips apply electrical voltage on an electrode adjacent to the cell containing the droplet. The voltage attracts the droplet to the electrode's surface and the droplet moves. This allows biochips to execute biochemical processes ('bio-protocols'), i.e., sequences of operations on droplets, including dispensing, splitting, merging, and mixing (Figure 2).

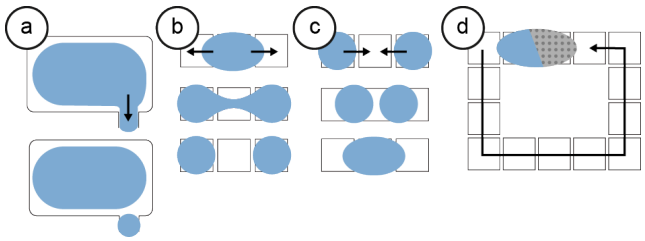


Figure 2: To execute biochemical processes or bio-protocols, biochips (a) dispense, (b) split, (c) merge, and (d) mix droplets.

The DIYbio [11] movement that aims at developing low-cost tools for, e.g., medical applications including low-cost biochips, holds the promise of giving people access to customized tests and procedures, thereby democratizing parts of health care. Unfortunately, designing such biochips for a specific biochemical process ('bio-protocol') requires substantial expertise in multiple disciplines [10, 12]. To better understand the struggles people have with biochip design, we invited 12 people without any prior knowledge on biochips to our lab and gave them the task to design biochips for bio-protocols of different levels of complexity (19 operations, 28 operations, and 71 operations respectively, see Figure 3 as an example). The bio-protocols had to execute within 20s, 45s, and 70s (e.g., moving a droplet by one electrode takes 0.01s). The reason for this is the fast degradation of samples [19]. Within this constraint, participants' objective was to produce the smallest possible biochip, i.e., a biochip with the minimum number of electrodes. Before starting, everyone watched a 40 min bio-chips lecture, and after that we informed them about the design constraints and how to compile the bio-protocol.

Participants reported that the most challenging aspect of the tasks was the trade-off between different resources: 'picking the right component as choices are interdependent' (p2), 'choosing the right chip size, small vs. fast' (p9), 'whether to choose one component over another while thinking of the cost of different solutions was super challenging' (p2), 'making trade-offs between different resources' (p5). Six participants suggested the use of a bio-protocol simulator to make the design process easier. While simulators exist that optimize for cost [4], no algorithm exists that optimizes for both dimensions cost and time simultaneously. Therefore, we propose a novel algorithm that takes both factors into consideration.

1.1 Multi-Objective Design Automation

We have implemented an algorithm that generates biochip layouts automatically based on a given bio-protocol, while considering cost and execution time. To illustrate how our algorithm works, we explain it at the example bio-protocol for an in-vitro diagnosis shown in Figure 3. The algorithm starts with a naïve biochip layout (Figure 4a), then iteratively generates multiple layout alternatives (Figure 4b/c). It evaluates each alternative in terms of cost and execution time and keeps the best solutions. It repeats this process for a specific number of generations, here 10,000. Figure 4d/e shows two versions of the final bio-chip generated by our algorithm: (d) is slightly more expensive, but performs faster, (e) is cheaper, but performs slightly slower, but within the time limit.

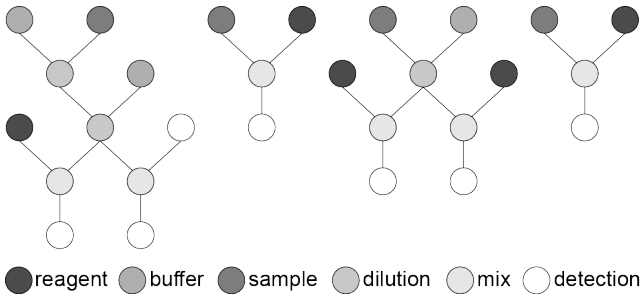


Figure 3: This example bio-protocol shows an in-vitro diagnosis that identifies microbes in human samples [15].

2 Related Work

Our work builds on previous work in biochips and in particular biochip optimization.

2.1 Biochip System

Droplet-based biochips were introduced in the late 2000s as a promising solution to a 'lab-on-a-chip' that can automate, miniaturize and integrate complex bio-protocols [18]. Since then, significant research efforts have been directed towards fabricating a cheap and reliable biochip. The most successful fabrication techniques so far are based on chromium electrodes on a glass substrate [8], gold electrodes on a printed circuit board [16] as in Figure 1, and silver electrodes printed on photographic paper [9].

2.2 Optimizing the Physical Layout

To optimize the layout of electrodes and reservoirs, which reduces the overall cost of the biochip, Alistar et al. introduced algorithms based on Simulated Annealing [2] and Tabu Search [4]. However, optimizing only for cost is not enough since timing, i.e., how fast the bio-protocol is executed on the bio-chip, is also crucial, especially since samples degrade. Developing an optimization algorithm for both factors is challenging since cost and time form a trade-off. For instance, when optimizing the number of reservoirs, there are two factors to consider: the number of reservoirs and the time it takes for droplets to be released. Having fewer reservoirs is cheaper, but since droplets can only be dispensed one at a time, it also takes longer.

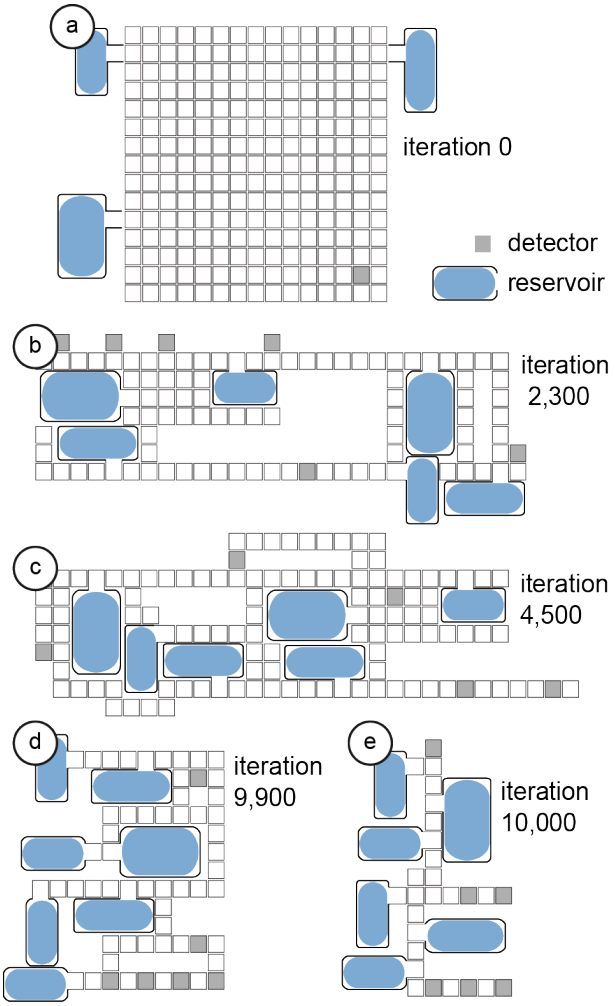


Figure 4: Biochip designs generated by our algorithm for the bio-protocol in Figure 3. (a) The algorithm starts from a naïve layout and (b), (c) evolves it iteratively to create (d), (e) the final layouts optimized in terms of cost and execution time.

The algorithm we propose optimizes for exactly this trade-off: it takes both cost and execution time into account, thereby allowing us to obtain the cheapest and fastest biochip.

2.3 Compiling a Bio-protocol

To determine the execution time of a bio-protocol, we need to translate it into a sequence of droplet movements on the biochip. Since this is an NP complete problem [13], there is no optimal solution. Initially, researchers proposed compilations for small sized bio-protocols using integer linear programming [13], but this approach does not scale. Generic methods to obtain the near-optimal solution include Parallel Recombinative Simulated Annealing [5], Genetic Algorithms [14], and Tabu Search [13]. Since they all employ different strategies to generate the final result, they are suitable for different use cases. However, all of these methods are slow and often have to run overnight. Recently, researchers developed an approach to run the compilation in real-time based on List Scheduling [1].

3 Proposed Algorithm

Algorithm 1: Optimization of the biochip design

Input: the *Bioprotocol*, deadline *Timeconstraint*, the cost of the components *Cost*, the number of solutions N, the number of *Iterations*, the mutation rate M_{rate}, the crossover rate C_{rate}

Output: the Pareto optimal biochip solutions *Children*

```
1  Solutions = GenerateInitialBiochip(Bioprotocol)
2  repeat
3      Children = GenerateLayouts(Solutions, M_rate, C_rate)
4      for each biochip B in Children do
5          CheckAndFixRoutability(B)
6          Compile(B, Bioprotocol, Timeconstraint)
7          CalculateCost(C, Cost)
8          if ) then ParetoOptimal(B, Solutions
9              CalculateCrowdingDistance(B, Solutions)
10             Update(Solutions, N)
11         end
12     end
13 until Iterations = ∅
```

Our algorithm is based on the Non-dominated Sorting Genetic Algorithm (NSGA II) [6]. Our algorithm takes as inputs the bio-protocol, the costs of the components, the

types of mutations and crossovers as described below, and the number of iterations. It outputs a number of different biochip designs that perform equally well but along different axes (e.g., one solution is cheaper but takes longer to execute vs. another solution is more expensive but faster).

We summarize all steps of our algorithm in Algorithm 1.

3.1 Generating new layouts through mutation/crossover

Our algorithm starts with an initial non-optimal solution provided as input. It then generates new solutions by performing mutations on the initial solution, such as removing or adding electrodes at random positions (see Figure 5). We use a large set of different mutation operations to increase the diversity of the biochip designs we generate. Since circular routes, such as the one shown in Figure 5d, can be used for both moving droplets and mixing, they enhance parallelism and thus save time [3]. We therefore favor them when generating new solutions.

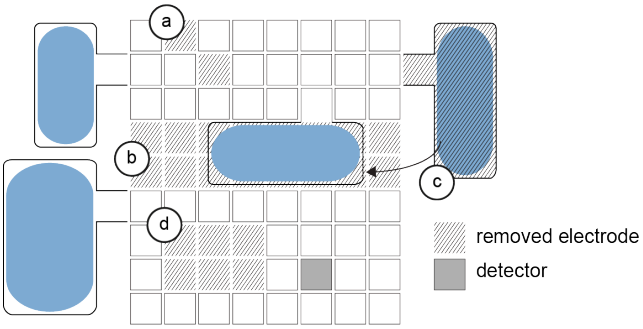


Figure 5: Here, our algorithm generates new layouts using mutations, such as (a) removing electrodes at random positions, (b) removing rows of electrodes, (c) re-positioning a reservoir, and (d) carving to favor certain mixing routes.

The initial naïve biochip solution is generated by computing the width of the bio-protocol graph (i.e., counting the maximum number of nodes on the largest level) and then placing the minimal number of electrodes needed to execute the operations in parallel. Our algorithm then adds border electrodes to enable faster routing. Figure 4a shows such a naïve biochip implementation for the bio-protocol from Figure 3.

3.2 Checking Routability

New solutions might have several disconnected parts (i.e., a droplet is not able to move from one side of the biochip to the other because of a gap, see Figure 5b). To

make sure that all electrodes are connected with each other, our algorithm checks the connectivity of the generated design (based on the k-vertex connectivity [7]) and if needed repairs the path by adding electrodes. The connecting electrodes are again added in a way that favors circular routes.

Checking if the generated design can actually execute the bio-protocol (e.g., if there is enough space to route the droplets and to place the mixing operations) is done in the next step together with the time estimation.

3.3 Evaluating the Execution Time

For each operation (e.g., mixing, dispensing, or splitting a droplet), our algorithm first places it on the biochip (taking up a certain number of electrodes), then determines a time at which it is executed, and finally computes a route to move the droplet there (we use the fast List-Scheduling [1] algorithm from the related work for this). Placing and scheduling the last operation determines the overall execution time of the bio-protocol. When our algorithm places the operations, it tries to avoid droplets passing each other in adjacent cells as this leads to them being merged (Figure 6). Our algorithm resolves this either by introducing extra electrodes for rerouting (Figure 6a) or by making one droplet wait until the other droplet has passed by (Figure 6b).

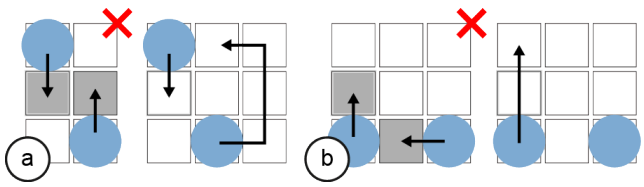


Figure 6: To avoid undesired droplet merge the droplets can be either (a) re-routed or (b) timed out.

In case not every operation can be placed (e.g., because there is not enough space, or droplets run into a deadlock), our algorithm dismisses that solution. For the remaining working solutions, our algorithm checks if they perform within the time constraints, and discards those that take too long.

3.4 Evaluating the cost of the solution

To evaluate the generated biochip design in terms of cost, we use the metric proposed by Alistar et al. [4] that considers both the cost for the electrodes/reservoirs and the costs for the fluids (Equation 1). The terms in the equation consist of the number of physical components N_{M_i} of type M_i (e.g., 5 electrodes, or 4 incubators that keep the droplets at a certain temperature), their cost $Cost_{M_i}$, and the number of reservoirs N_{R_i} times the cost of the fluid per µl R_i.

$$Cost_{\mathcal{A}} = \sum N_{M_i} \times Cost_{M_i} + \sum N_{R_i} \times Cost_{R_i}, \quad (1)$$

The first term of Equation 1 calculates the cost of the physical components and the second term calculates the cost of the input fluids. The physical components (e.g., electrodes, reservoirs and detectors) and their unit cost are provided by the designer in a library. The unit cost of the input fluids, used by the biochemical application, are specified in a fluidic library. The assumption is that *all* the reservoirs integrated in the cartridge are fully loaded. We ignore the cost of the controller platform because, regardless of its cost, the controller platform is acquired only once, thus having its cost amortized over time.

3.5 Next Generations

To increase the space of solutions, the working solutions that were generated from the initial input form the basis for the next round of mutations. Our algorithm uses only the best N solutions as input, thus it has to rank them first. The ranking is done on both scales time and cost using the Pareto dominance (e.g., one solution is cheaper but takes longer to execute vs. another solution that is more expensive but faster), and crowding distance (solutions with similar cost/time are ranked lower).

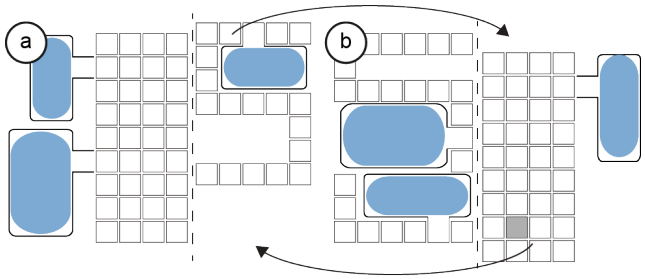


Figure 7: After generating the first round of solutions, we can use cross-overs between them for the next iteration.

Our algorithm then repeats steps 1–4, but now also uses cross-overs in step 1 as illustrated in Figure 7 in addition to the mutations. For the next ranking, all generated solutions (i.e., from the current and all previous generations) are considered. Our algorithm repeats the steps until the defined number of iterations is reached.

4 Performance

We evaluated the performance of our algorithm by comparing it to the best known solution as reported by Alistar et al. [4] using the four bio-protocols reported in [17, 20]. However, as pointed out in the related work section, Alistar et al.'s approach only

optimizes for cost and only outputs a single solution while our algorithm outputs a continuous solution space that allows users to select among different trade-offs. Figure 8 reports that solution space for one of the bio-protocols (the interpolating dilution bio-protocol from [17]) and shows how our algorithm outputs better solutions in terms of both cost and execution time. For a direct comparison, we select from our solution space, those solutions that are similar to the solution in the related work along one of the dimensions, i.e., either execution time (solution 1) or cost (solution 2). Solution 1, which has a similar execution time as the related work, is 74 % cheaper. Solution 2, most similar in cost, executes the bio-protocol 67 % faster.

Figure 8: Our algorithm outputs a continuous solution space that allows selection among trade-off alternatives.

We obtained similar results for the other three bio-protocols with an improvement over the related work from 20 % to 76 % in terms of cost and 27 % to 125 % in terms of execution time.

5 Discussion and Future Work

We presented an algorithm that automates the design process of biochips by optimizing for both cost and time constraints. By providing an automated way to generate biochips efficiently, we provide a step towards a future in which even non-technical users are able to create biochips for their personal applications, thereby democratizing parts of health care. Our algorithm obtained a continuous solution space that allows selection among trade-off alternatives. When compared to the related work,

our algorithm performed up to 125 % better in terms of execution time, and up to 76 % better in terms of cost.

As future work, we will further explore design strategies for more complex biochip systems. We plan to investigate strategies that combine both automation and human editing.

References

[1] M. Alistar, P. Pop, and J. Madsen. "Online synthesis for error recovery in digital microfluidic biochips with operation variability". In: *Proc. of the Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS*. 2012, pages 53–58.

[2] M. Alistar, P. Pop, and J. Madsen. "Application-specific fault-tolerant architecture synthesis for digital microfluidic biochips". In: *Proc. of the 18th Asia and South Pacific Design Automation Conference*. 2013, pages 794–800.

[3] M. Alistar, P. Pop, and J. Madsen. "Operation placement for application-specific digital microfluidic biochips". In: *Proc. of the Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS*. 2013, pages 1–6.

[4] M. Alistar, P. Pop, and J. Madsen. "Synthesis of Application-Specific Fault-Tolerant Digital Microfluidic Biochip Architectures". In: *IEEE T. on Computer-Aided Design of Integrated Circuits and Systems* 35.5 (2016), pages 764–777.

[5] K. Chakrabarty and F. Su. *Digital microfluidic biochips: synthesis, testing, and reconfiguration techniques*. CRC Press, 2006.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pages 182–197.

[7] S. Even. "An algorithm for determining whether the connectivity of a graph is at least *k*". In: *SIAM J. on Computing* 4.3 (1975), pages 393–396.

[8] R. Fobel, C. Fobel, and A. R. Wheeler. "DropBot: An open-source digital microfluidic control system with precise control of electrostatic driving force and instantaneous drop velocity measurement". In: *Applied Physics Letters* 102.19 (2013), page 193513.

[9] R. Fobel, A. E. Kirby, A. H. Ng, R. R. Farnood, and A. R. Wheeler. "Paper microfluidics goes digital". In: *Advanced materials* 26.18 (2014), pages 2838–2843.

[10] C. L. Kaiying and S. Lindtner. "Legitimacy, boundary objects & participation in transnational DIY biology". In: *Proceedings of the 14th Participatory Design Conference: Full papers-Volume 1*. ACM. 2016, pages 171–180.

[11] S. Kuznetsov, C. Doonan, N. Wilson, S. Mohan, S. E. Hudson, and E. Paulos. "DIYbio things: open source biology tools as platforms for hybrid knowledge production and scientific participation". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM. 2015, pages 4065–4068.

[12] S. Kuznetsov, A. S. Taylor, T. Regan, N. Villar, and E. Paulos. "At the seams: DIYbio and opportunities for HCI". In: *Proceedings of the Designing Interactive Systems Conference*. ACM. 2012, pages 258–267.

[13] P. Pop, M. Alistar, E. Stuart, and J. Madsen. *Fault-Tolerant Digital Microfluidic Biochips*. 2016.

[14] A. J. Ricketts, K. Irick, N. Vijaykrishnan, and M. J. Irwin. "Priority scheduling in digital microfluidics-based biochips". In: *Proc. of the Conference on Design, Automation and Test in Europe*. 2006, pages 329–334.

[15] J. Sambrook, E. F. Fritsch, T. Maniatis, et al. *Molecular cloning*. Volume 2. Cold spring harbor laboratory press New York, 1989.

[16] R. S. Sista, T. Wang, N. Wu, C. Graham, A. Eckhardt, T. Winger, V. Srinivasan, D. Bali, D. S. Millington, and V. K. Pamula. "Multiplex newborn screening for Pompe, Fabry, Hunter, Gaucher, and Hurler diseases using a digital microfluidic platform". In: *Clinica Chimica Acta* 424 (2013), pages 12–18.

[17] F. Su and K. Chakrabarty. "Benchmarks for digital microfluidic biochip design and synthesis". 2006.

[18] F. Su, K. Chakrabarty, and R. B. Fair. "Microfluidics-based biochips: technology issues, implementation platforms, and design-automation challenges". In: *IEEE Transactions on computer-aided design of integrated circuits and systems* 25.2 (2006), pages 211–223.

[19] J.-I. Yoshida. "Flash chemistry: flow microreactor synthesis based on high-resolution reaction time control". In: *The Chemical Record* 10.5 (2010), pages 332–341.

[20] Y. Zhao, T. Xu, and K. Chakrabarty. "Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-chip". In: *J. on Emerging Technologies in Computing Systems* 6.3 (2010). DOI: 10.1145/1777401.1777404.

One Working Day of the Berlin Police

Analysis of Data From the #*24hPolizei* Twitter Marathon

Aragats Amirkhanyan

Internet Technologies and Systems
Hasso-Plattner-Institut
Aragats.Amirkhanyan@hpi.de

Nowadays, emergency agencies actively use the potential of the social networks from the perspective of providing situational and public safety awareness. One of such agencies is the Berlin Police that is the subject of our interest in this report. They use its Twitter account to inform Berlin inhabitants about incidents that happen in the city. Also, they provide the Twitter marathons and campaigns to demonstrate how the police work and to provide public safety awareness for inhabitants. In this report, we analyze data from the Twitter account of the Berlin Police during the #*24hPolizei* marathon in May 2016. We provide the description of the methods for normalization, entity recognition, and tweets categorization. Our analysis results show the different views on public safety: distribution of incidents among districts in the city, distribution of different types of incidents, the intensity of the incidents during the day and others.

1 Introduction

Many law enforcement agencies and state departments actively use the potential of social networks from the perspective of providing situational awareness. They use social networks as a channel for informing people about threat-related events on a daily basis and, especially, during extreme events [4, 5, 7]. The Berlin Police Department is one of the law enforcement departments that actively uses social networks, particularly, its Twitter account (@*PolizeiBerlin_E*) [9], to post tweets (messages) about incidents in the city. And in this way, they provide situational awareness for Berlin inhabitants. Also, the police once per year run the #*24hPolizei*[1] marathon. During this marathon, which goes 24 hours, the police post in real-time tweets about incidents happening in the city. Sometimes after marathons, the police publish in Facebook the statistics of the most utilized words in tweets. Such analysis of data and statistics are limited and they do not fully discover the potential of data. Therefore, we are interested in the deep analysis of data from the #*24hPolizei* Twitter marathon. It can give us the possibility to obtained the detailed view, understand the safety situation in the city during the marathon, and as a result, provide public safety awareness for city inhabitants.

The remainder of the report is organized as follows: in Section 2, we provide the motivation and the brief overview of related work. Section 3 explains briefly

[1] https://de.wikipedia.org/wiki/24hPolizei (last accessed 2016-10-20).

our research design. Then in Section 4, we present our analysis and findings. We conclude the report and provide directions for future work in Section 5.

2 Motivation and Related Work

This research study is the part of our main research project, which is analysis and visualization of publicly available data to provide situational and public safety awareness [1, 2, 3]. Our focus is public safety awareness of Berlin, therefore, we gather relevant to this location publicly available data. One of such relevant sources of data is the Twitter account of the Berlin Police (*@PolizeiBerlin_E*). The police use it to post tweets about incidents that happen in the city. Analysis of data from this account is the next step of the research project and it is the challenge that we address in this report.

The data from the account of the Berlin Police have been partially analyzed in some papers [6, 8]. Mirbabaie and Ehnis et al. [8] analyzed the communication roles in public events. For that, they analyzed the Twitter data related to the 1st May 2014 event (Labour Day) in Germany. In their analysis, the account of the Berlin Police was one of five primary roles during the 1st May event and tweets from them were most re-tweeted very frequently. They showed the primary role of the Berlin Police in informing users about the happening event. Later in December 2014, Ehnis and Mirbabaie et al. [6] published another paper, in which they presented their analysis of the role of social media network participants in extreme events. In this paper, they again used data from the 1st May for analysis and they analyzed the network behavior and perception of the police role during the 1st of May event. The analysis showed the major role of the police during the 1st May event.

From the related papers of previous years, we have seen that these social network accounts of emergency agencies play the major role in informing users about incidents and, especially, during the extreme events. The data from the Twitter account of the Berlin Police have been partially analyzed in previous papers, but the analysis was very specific and limited by the concrete use case. We claim that more deep analysis of these data can bring new research findings and research results in the scope of analysis of public safety in the city. Therefore, in this report, we provide our research analysis of data from the account of the Berlin Police during the *#24hPolizei* marathon jointly with the research methods that we use for normalization and analysis.

3 Research Design

For analysis, we gathered data from the Twitter account of the Berlin Police (*@PolizeiBerlin_E*) during the last *#24hPolizei* marathon in May 2016. Data contain original and retweeted tweets. To fetch these data, we developed the TwitterCrawler tool based on Twitter Public API [12].

The key for analysis of data is the data normalization. One of the ways for normalization is the extraction of valuable named entities. Therefore, in the next two subsections, we provide the brief overview of the methods for the location entity recognition and incident-related keywords extraction that we use.

3.1 Normalization – Location

The Berlin Police post tweets without geo coordinates, but they quite often include location names or location hashtags into the text to show where the incident happened. Therefore, we aim to recognize and extract these location entities.

We use the Stanford Named-Entity Recognition (NER) tool [11] that supports the German language. This tool recognizes location entities in 66.91 % of data. We do not evaluate this result because the challenge of location entity recognition is out of the scope of this report and we use one of the existing solutions that supports the German language. Meanwhile, we do not expect to have 100 % of location recognized tweets, because a lot of them do not have location names. Many tweets are just announcements and updates of recent news.

We observed that there are some cases when the NER tool can not recognize the location entity in the text even if it is there. Therefore, we apply the second approach to increase the percentage of recognized locations. This approach is dictionary-based entity recognition (dictionary ER). We noticed that quite often tweets about incidents have the names of S-Bahn and U-Bahn stations. Therefore, we created the dictionary from the station names. The dictionary originally contains 315 station names, but we extend it up to 1222 names because the station names are not always written in the original form. If the station name has 2 or more words then the name can be written in the concatenated form without whitespaces or concatenated with dashes. It is just a one example, but we carefully apply known variations of reductions and concatenations to extend the dictionary and to cover all possible ways of writing station names. Additionally, we use the lowercase for the station names and tweets' texts.

The tweet has usually one location name, but if it has more we consider all of them. If the tweet has two or more location names and one of them is Berlin then we consider only others because Berlin reflects the very broad area. The dictionary ER recognizes location entities in 60.74 % of tweets, which is less than the result of the Stanford NER tool. But if we combine both methods then we obtain 83.55 %.

3.2 Normalization – Incident

We observed that the Berlin Police post tweets about different types of incidents (crimes) with appropriate hashtags: thefts (*#pickpocket*), traffic accidents (*#8geben*), break-ins (*#keinbruch*), public disorders, and some news. Therefore, we aim to categorize tweets into these four categories, which we call incident types. Here is the example of the incident-tweet (the tweet about the incident): Schon wieder **Taschendiebe** in **#Charlottenburg**. Wir helfen mehreren **Geschädigten**. #24hPolizei (English: Again **pickpockets** in **#Charlottenburg**. We help several **victims**. #24hPolizei).

Table 1: Example of Keywords of Incident Types

Theft	**Traffic Accident**
#pickpocket	#8geben
Stehlen (steal)	Unfall (accident)
Taschendieb (pickpocket)	VU (traffic accident)
Gestohlen (stolen)	Verkehrsunfall (traffic accident)
Dieb (thief)	Prallt(-en) (clashed)

Break-in	**Public Disorder**
#keinbruch	Alkohol (alcohol)
Einbruch (burglary)	Verprügelt (beaten)
Einbrecher (housebreaker)	Schrei(e) (scream)
Alarmanlage (alarm system)	Laut (loud)
-	Betrunken (drunk)

For incidents extraction we use dictionary-based ER. We observed the most utilized incident-related keywords and grouped them into four categories (incident types) and created the dictionary. In Table 1, you can find some keywords of categories: Theft, Traffic Accident, Break-in and Public Disorder. The original dictionary contains stemmed keywords in the lowercase. For stemming, we use the Snowball Stemmer library [10].

Every tweet should have one category. Therefore, firstly, we look at the hashtag, whether it is one of the three predefined (*#pickpocket*, *#8geben*, *#keinbruch*) because it can help easily to identify the category. Then we try to find the matches between words in the text and keywords in the dictionary. If the tweet has keywords of the several incident types, then we choose the type that is more specific (narrow). From Theft and Break-in, we choose Break-in. From Public Disorder and other, we choose other.

We have more keywords that can reflect incidents and threat-related information, but they do not definitely belong to one of the four categories. Meanwhile, they should be considered for public safety analysis. Such keywords we group into the "Other" category. We do not present them in the table, but they are more general than for other categories, for example, Krankenhaus (English: hospital), Hilfloser (English: helpless), Verdacht (English: suspicion) and so on.

Table 2: Statistics of Data

Tweets	Tweets with Location	Incident-tweets	Incidents with Injured
1070	894 (83.55 %)	562 (52.52 %)	34 (6.04 %)

4 Analysis – #24hPolizei Marathon

The Berlin Police once per year run the Twitter #*24hPolizei* marathon. During this marathon, which goes 24 hours, the police post tweets with the hashtag #*24hPolizei* about incidents happening in the city. Many of these tweets come from the calls to the police. The goal of this marathon is to show the police work and provide public safety awareness for inhabitants. Overall, there were 3 #*24hPolizei* marathons, but in this report we analyze only the last one in May 2016.

The last #*24hPolizei* marathon was on the 27–28th of May 2016 from 19:00 until 19:00 of the next day. Overall, we have 1070 tweets and 83.55 % of them have location names recognized by the Stanford NER tool and dictionary ER. Also, we apply the method for recognition and extraction of incident-related keywords from Table 1 and, as a result, 52.52 % of tweets are categorized into incident types. Additionally, we calculate how many tweets have keywords that reflect the injured. For German language, such words are Verletzung (English: injury) and verletzen (English: hurt). The percentage of incident-tweets with possible injured people equals 6.04 %. This statistics is presented in Table 2.

We calculate the frequency of the incident-related keywords in tweets. The top 10 utilized keywords are presented in the list below. In the list, you can find the German keyword, English translation, the count of tweets with this keyword, the percentage of tweets over the incident-tweets and the incident type to which the keyword belongs.

1. Laut (loud) 66 (11.74 %) – Public Disorder
2. Verdacht (suspicion) 36 (6.40 %) – Public Disorder
3. Verkehrsunfall (traffic accident) 35 (6.22 %) – Traffic Accident
4. Verletzt (injured) 33 (5.87 %) – Any type
5. #8geben 31 (5.51 %) – Traffic Accident
6. Schlag (beat) 27 (4.80 %) – Public Disorder
7. Unfall (accident) 22 (3.91 %) – Traffic Accident
8. Verlass (leave) 20 (3.55 %) – Any type
9. Betrunk (drunk) 19 (3.38 %) – Public Disorder
10. Larm (alarm) 18 (3.20 %) – Public Disorder

Table 3: Statistics of Incident Types

Public Disorder	Traffic Accident	Theft	Break-in	Other
274 (48.76 %)	96 (17.08 %)	52 (9.25 %)	19 (3.38 %)	121 (21.53 %)

In Table 3, we present the result of tweet categorization. In the overall data, the major part of categorized tweets belongs to the Public Disorder incident type. It has 48.76 % of categorized tweets. Then we have Traffic Accident and Theft. Break-in has very small percentage in the comparison to others. The Other type (tweets with not categorized keywords) is 21.53 % of data.

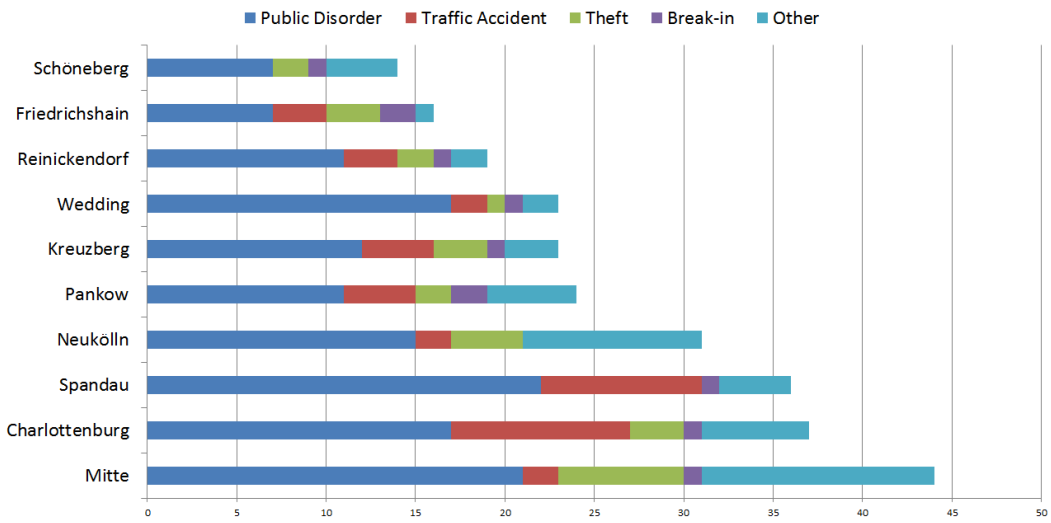


Figure 1: Distribution of the incident types in top locations.

In the list below, we present the top 10 location names utilized in tweets. All location names in the list are the names of the districts in Berlin.

1. Mitte 89 (9.95 %)
2. Charlottenburg 60 (6.71 %)
3. Spandau 59 (6.59 %)
4. Neukölln 52 (5.81 %)
5. Wedding 48 (5.36 %)
6. Kreuzberg 47 (5.25 %)
7. Pankow 38 (4.25 %)

8. Reinickendorf 37 (4.13 %)
9. Lichtenberg 34 (3.80 %)
10. Friedrichshain 33 (3.69 %)

You can see that, in the first place, we have Mitte 89 (9.95 %), which is the city center (Mitte reflects the city center if Mitte is only location name in the tweet). Then we have Charlottenburg 60 (6.71 %), which is the very lively district in the western part of the city. Next, we have Spandau, Neukölln, Wedding and Kreuzberg. Kreuzberg and Neukölln are known for its large percentage of immigrants and descendants of immigrants. Additionally, we need to mention that for the statistics we do not consider Berlin as a location name because Berlin is the too broad area.

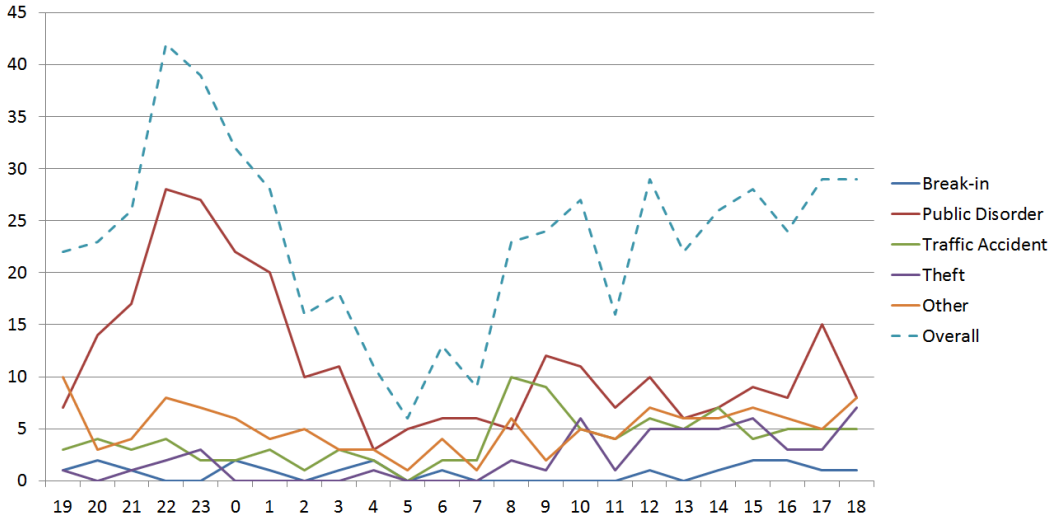


Figure 2: The #*24hPolizei* marathon: distribution of incidents over the day.

Figure 1 presents the bar chart of the top 10 locations where incidents happen. You can visually see the distribution of the incident types and the total number of incidents in locations. Firstly, we can see that Mitte is in the first place by the number of incidents and it dominates in the Theft type of incidents among other locations in the chart. Charlottenburg has the second place and Charlottenburg dominates in Traffic Accident over others locations with 10 incidents of that type. But it is comparable with Spandau that has 9 traffic accidents during the marathon. Spandau is in the third position with domination of the Public Disorder type of incidents and, comparable to Charlottenburg, the number of traffic accidents. We can notice that Break-in is the minor part of incidents. There are overall 52 incidents of that type (Table 3) and the dominating location in thefts is Mitte.

Since the marathon covers 24 hours, we have a chance to build the chart to find out the intensity of tweets by hours for different types of incidents. You can find this chart in Figure 2. We have data starting from 19:00, which is the start time of the

marathon. And immediately, we can see the dramatic increase of Public Disorder. This type of the incidents dominates over others during almost the entire marathon except for 8:00. We have the high intensity of incidents during the entire night and it goes down only closer to the morning after 3:00. During this marathon, we have the low number of break-ins and from 7:00 until 11:00 there are not Break-in tweets at all. Then we can see that on the next day with coming the evening, Public Disorder starts to increase again.

5 Conclusion and Future Work

Our analysis gives an initial overview of the safety situation in Berlin based on data from the *#24hPolizei* Twitter marathon organized by the Berlin Police. The result of this research study is the part of our research project (Section 2) and one of the steps towards achieving the main goal – providing situational and public safety awareness. We presented the research design and gave the description of the normalization process. This normalization gave us the possibility to analyze data from different perspectives. We provided top locations, distribution of incident types in data and distribution of incident types in concrete locations. Since during these marathon, the police continuously publish tweets about incidents during 24 hours, we also analyzed how the public safety situation changes in the city during the day. For that, we provided the chart of the distribution of incidents per hours during the day of the marathon. From that, we have seen the characteristic increase of incidents in the evening time.

As with every study, there are limitations. Our limitation based on the normalization task. Therefore, as future work, we plan to improve methods of location entity recognition. One of the possible solutions is usage of DBpedia as the third additional method for annotation and recognition of entities. Additionally, we plan to improve tweet categorization. For that, we would like to apply machine learning approaches for tweet classification. Also, we would like to integrate additional sources of relevant data.

References

[1] A. Amirkhanyan, F. Cheng, and C. Meinel. "Real-time clustering of massive geodata for online maps to improve visual analysis". In: *Innovations in Information Technology (IIT), 2015 11th International Conference on*. Nov. 2015, pages 308–313. DOI: 10.1109/INNOVATIONS.2015.7381559.

[2] A. Amirkhanyan and C. Meinel. "Visualization and analysis of public social geodata to provide situational awareness". In: *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*. Feb. 2016, pages 68–73. DOI: 10.1109/ICACI.2016.7449805.

[3] A. Amirkhanyan and C. Meinel. "Analysis of the Value of Public Geotagged Data from Twitter from the Perspective of Providing Situational Awareness". In: *Social Media: The Good, the Bad, and the Ugly: 15th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2016, Swansea, UK, September 13–15, 2016, Proceedings*. Edited by K. Y. Dwivedi, M. Mäntymäki, M. Ravishankar, M. Janssen, M. Clement, L. E. Slade, P. N. Rana, S. Al-Sharhan, and C. A. Simintiras. Cham: Springer, 2016, pages 545–556. ISBN: 978-3-319-45234-0. DOI: 10.1007/978-3-319-45234-0_48.

[4] C. Ehnis and D. Bunker. "The impact of disaster typology on social media use by emergency services agencies: the case of the Boston Marathon bombing". In: *24th Australasian Conference on Information Systems (ACIS)*. RMIT University, 2013, pages 1–12.

[5] C. Ehnis and D. Bunker. "Social media in disaster response: Queensland Police Service-public engagement during the 2011 floods". In: *ACIS 2012: Location, location, location: Proceedings of the 23rd Australasian Conference on Information Systems 2012*. ACIS, 2012, pages 1–10.

[6] C. Ehnis, M. Mirbabaie, D. Bunker, and S. Stieglitz. "The Role of Social Media Network Participants in Extreme Events". In: *Proceedings of the 25th Australasian Conference on Information Systems*. Publication status: Published. Auckland, New Zealand, 2014.

[7] T. Heverin and L. Zach. "Twitter for City Police Department Information Sharing". In: *Proceedings of the 73rd ASIS&T Annual Meeting on Navigating Streams in an Information Ecosystem - Volume 47*. ASIS&T '10. Pittsburgh, Pennsylvania: American Society for Information Science, 2010, 41:1–41:7.

[8] M. Mirbabaie, C. Ehnis, S. Stieglitz, and D. Bunker. "Communication Roles in Public Events". In: *Information Systems and Global Assemblages. (Re)Configuring Actors, Artefacts, Organizations: IFIP WG 8.2 Working Conference on Information Systems and Organizations, IS&O 2014, Auckland, New Zealand, December 11–12, 2014. Proceedings*. Edited by B. Doolin, E. Lamprou, N. Mitev, and L. McLeod. Berlin, Heidelberg: Springer, 2014, pages 207–218. ISBN: 978-3-662-45708-5. DOI: 10.1007/978-3-662-45708-5_13.

[9] A. Rickmann. *Storfiy: Der erste Twitter-Einsatz der Berliner Polizei*. URL: http://andreasrickmann.de/2014/03/23/storfiy-der-erste-twitter-einsatz-der-berliner-polizei (last accessed 2016-10-01).

[10] *Snowball Stemmer*. URL: http://snowball.tartarus.org/ (last accessed 2016-10-01).

[11] *Stanford Named Entity Recognizer (NER)*. URL: http://nlp.stanford.edu/software/CRF-NER.shtml (last accessed 2016-10-01).

[12] *Twitter Public API Documentation*. last visited on 27.07.2016. URL: https://dev.twitter.com/rest/reference/get/statuses/user_timeline (last accessed 2016-10-01).

Enhancing Decision Making for Business Processes

Ekaterina Bazhenova

Business Process Technology Group
Hasso-Plattner-Institut
Ekaterina.Bazhenova@hpi.de

Business process management is an acknowledged asset for running a company in an efficient way. A firm's value chain is directly affected by how well it designs and coordinates enterprise decision making. In recent years, a number of decision management frameworks have appeared in addition to existing business process management systems. Coupled with the recent release of the Decision Model and Notation aimed to be complementary to the Business Process Model and Notation by the OMG group, it is evident that stakeholders need integrated business process and decision management solutions. In order to highlight the existing gap, we introduced a concept of integrated business process and decision lifecycle and discussed the challenges related to the separation of concerns associated with each step. The lifecycle is entered in the design phase, in which business processes and decisions are identified and represented by corresponding models. Often in practice, decision logic is either explicitly encoded in process models through control flow structures, or it is implicitly contained in process execution logs. Our work proposes an approach of semi-automatic derivation of DMN decision models from process event logs with the help of decision tree classification. The approach is demonstrated by an example of a loan application in a bank.

1 Introduction

The value of business processes management (BPM) has been acknowledged as an essential asset to drive a company. One of the most important and challenging BPM aspects is decision making. Adding decision management perspective improves the process by focusing on both the way decisions are made and directing the process that must carry out the decisions. Following the "separation of concerns" paradigm, this allows to handle the compounding complexity of the business process models by externalizing the events, operational conditions and decisions in separate decision models, e.g. which is supported by the recent Decision Model and Notation (DMN) [8]. As companies are interested in running effective and competitive processes, they develop different decision support software (i.e., IBM Operational Decision Management, SAP Decision Service Management). Whether companies develop packaged decision management systems, adopt business rules, or apply advanced analytics to their business, a thorough decision understanding, modelling and execution is critical. The interest towards decision support in business processes is indicated by a number of decision ontologies and notations [7, 8], as well as decision service platforms [12]. However, compared to the extensive stream of BPM research, the decision perspective of business processes up to now has received by far less attention.

In our work we aim at overcoming the existing gap by identifing the challenges of decision modeling with respect to processes. To this end, we introduce in Section 2 the concept of a decision lifecycle at the enterprise with respect to business process perspective. Then, for each stage we establish the challenges coming emerging from theory and industrial experience. The lifecycle is entered in the design and analysis phase, when processes and decisions are modelled. Often in practice, decision logic is either explicitly encoded in process models through control flow structures, or it is implicitly contained in process execution logs. Section 3 proposes an approach of semi-automatic derivation of DMN decision models from process event logs with the help of decision tree classification. The approach is demonstrated by an example of a loan application in a bank. Extensions of the approach can be found in Section 4. The report reflects our work conducted during the year of 2016 and is reflected by the list of published and submitted papers in Section 5.

2 Challenges for the Integration of Business Process and Decision Management

Whereas business process management and decision management have developed over the last few decades into mature independent disciplines, little has been understood about how to effectively apply these concepts complementary to each other. Our work addresses a number of challenges required to reduce this gap. To this end, we presented the concept of an integrated BPDM-lifecycle which further served for scoping the challenges. It seems practical to view business process and decision management within the frames of a corresponding *Business Process and Decision Management lifecycle (BPDM-lifecycle)* as shown in Figure 1. The stages of the lifecycle in Figure 1 correspond to the stages of a standard business process lifecycle presented, e.g., by [11].

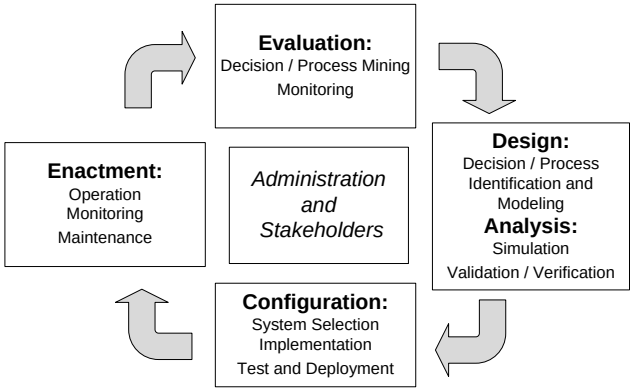


Figure 1: Integrated Business Process and Decision Management Lifecycle.

The phases of the BPMD-lifecycle are organized in a cyclical structure showing their logical dependencies. These dependencies do not imply a strict temporal ordering in which the phases need to be executed. Also, it is important to note that the evolution of process and decision management at real enterprises is not necessarily simultaneous.

In total, we identified 14 challenges relevant to each of the four stages, namely design and analysis, configuration, enactment, and evaluation of interconnected enterprise processes and decisions. For each challenge, we described an intrinsic problem, explained how related work has addressed the challenges, and identified the research gaps. The architecture and the challenges of integrated business process and decision management are demonstrated on a real-life example of the debt collection process, through the BPMN model reflecting the process of interaction between the company and the customer, and the DMN model representing the decision logic for handling claims. Our list of challenges is designed to provide a roadmap for future studies, whereby each challenge represents a research problem, the solution of which could help companies exploiting the full potential of integrated business process and decision management ultimately aimed at allowing enterprises to work optimally and flexibly.

3 Discovering Decision Models from Event Logs

The BPDM-lifecycle is entered in the *Design and Analysis* phase, in which business processes and decisions are identified, reviewed, validated and represented by corresponding models. To assist companies with successful automated decision management, knowledge about "as-is" decision making needs to be retrieved. This can be done by analysing process event logs and discovering decision rules from this information. Existing approaches to decision mining concentrate on the retrieval of control flow decisions but neglect data decisions and dependencies that are contained within the logged data. To overcome this gap, in [2] we extended an existing approach to derive control flow decisions from event logs [9] with additional identification of data decisions and dependencies between them. Furthermore, we proposed an algorithm for detecting dependencies between discovered control flow and data decisions. The output of this approach is a complete DMN decision model which explains the executed decisions, which can serve as a blueprint for further decision management.

3.1 Discovering Decision Models from Event Logs

For our work, we rely on notions of process model and execution as follows. A *process model* is a tuple $m = (N, C, \alpha)$, where $N = T \cup G$ is a finite non-empty set of control flow nodes, which comprises sets of activities T, and gateways G. $C \subseteq N \times N$ is the control flow relation, and function $\alpha : G \rightarrow \{xor, and\}$ assigns to each gateway a type in terms of a control flow construct. A process execution is a sequence of activity

instances $t_1 \ldots t_n$, with $n \in \mathbb{N}$ and each t_i is an instance of an activity in the set of activities T of m.

Let E be the set of event instances and A a finite set of attributes. Each attribute $a \in A$ is associated with the corresponding domain $V(a)$, which represents a set of either numeric or nominal values. Each event instance $e \in E$ has tuples (a, v), $a \in A$, $v \in V(a)$ assigned to it. A trace is a finite sequence of event instances $e \in E$ such that each event instance appears in the trace only once. An event log L is a multi-set of traces over E.

To represent the knowledge about decisions taken in business processes, we use the DMN standard, which distinguishes between two semantic levels: the *decision requirements* and the *decision logic*. The first one represents how *decisions* depend on each other and what *input data* is available for the decisions; these nodes are connected with each other through *information requirement edges*. A *decision requirement diagram* DRD is a tuple (D_{dm}, ID, IR) consisting of a finite non-empty set of decision nodes D_{dm}, a finite non-empty set of input data nodes ID, and a finite non-empty set of directed edges IR representing the information requirements such that $IR \subseteq D_{dm} \cup ID \times D_{dm}$, and $(D_{dm} \cup ID, IR)$ is a directed acyclic graph. A decision may additionally reference the decision logic level where its output is determined through an undirected association. One of the most widely used representation for decision logic is a decision table, which we utilize for the rest of the paper. *Decision table* $DT = (I; O; R)$ consists of a finite non-empty set I of inputs, a finite non-empty set O of inputs, and a list of rules R, where each rule is composed of the specific input and output entries of the table row.

In our work, we provided a formal framework enabling the extraction of complete decision models from event logs on the examples of Petri nets and DMN decision models. In particular, we extended an existing approach to deriving control flow decisions from event logs with additional identification of data decisions and dependencies between them. Furthermore, we proposed a modified approach to rebuilding decision trees to identify the dependencies between discovered decisions and overcame the problem of reusing attributes in a dependent decision. An assumption of our approach was that the decisions do not appear within loops, which we plan to investigate in future work. The extracted DMN decision model reflects the decisions detected in the event log of a process model, which could be served as an explanatory model used for compliance checks. Additionally, executing this model complementary to the process model supports the principle of separation of concerns by providing increased flexibility, as changes in the decision model can be executed without changing the process model.

3.2 Running Example

Our example process represents a loan application in a bank, as shown in Figure 2. Although we used a Petri net for the model representation, our approach can be applied to a wider class of notations, e.g., BPMN.

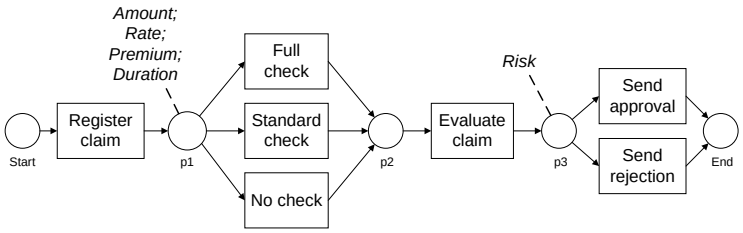


Figure 2: Process model of the loan application in a bank.

For analysing the example process, we created an event log with the help of the simulation system CPN Tools.[1] This tool uses coloured Petri nets for models' representation which allow tokens to have data values attached to them, as in our example process. We used the simulation parameters as presented in Table 1.

Table 1: Simulation parameters for generating the event log

Task/Attribute Name	Simulation Parameters
Trace ID	1 to 200 (incrementing)
Amount	discrete(2,99)
Premium	random boolean
Duration	discrete(2,30)
Rate	*Amount / Duration*
Risk	if $Amount \geq 50$ and $Duration > 15 : Risk = 4$
	if $Amount \geq 50$ and $Duration < 15$ and $Duration > 5 : Risk = 3$
	if $Amount \geq 50$ and $Duration < 5$: $Risk = 2$
	if $Amount < 50$ and $Duration > 20$: $Risk = 3$
	if $Amount < 50$ and $Duration < 20$ and $Duration > 10 : Risk = 2$
	if $Amount < 50$ and $Duration < 10 : Risk = 1$
$p1$	if $Amount \geq 50$ and $Premium = false$: *Full check*
	if $Amount < 50$ and $Premium = false$: *Standard check*
	if $Premium = true$: *No check*
$p3$	if $Risk \leq 2$: *Send approval*; if $Risk > 2$: *Send rejection*

Table 2 shows a fragment of the simulated event log for the process depicted in Figure 2. Whereas the knowledge about the process decisions can be empirically derived from the logged expert decisions depicted in Table 2 in the form of credit evaluation rules, the corresponding process model depicted in Figure 2 does not allow for decision knowledge to be obtained. Moreover, simply applying these rules for the development of credit scoring systems can lead to the unjust treatment of an individual applicant. An advantage of using such a model is that the separation of process and decision logic maximizes agility and reuse of decisions [4].

[1] http://cpntools.org/ (last accessed 2016-10-20).

Table 2: An excerpt of the event log for the example process

Event ID	Trace ID	Name	Other attributes
1	1	Register claim	Amount = 84 [EUR], Rate = 2.8 [%], Duration = 30 [Mths], Premium = false
2	1	Full check	—
3	2	Register claim	Amount = 80 [EUR], Rate = 4.4 [%], Duration = 18 [Mths], Premium = true
4	2	No check	—
5	1	Evaluate	Risk = 3
6	1	Send rejection	—

3.3 Application of the Approach on an Example Log

For evaluating our approach of the decision model discovery from the event log of a process model, we implemented it as a plug-in for the ProM framework 5.2[2] by extending the existing plug-in "Decision Point Analysis" for the discovery of control flow decision points [9] with our concepts. The ability of the tool to derive decision models from event logs is shown in a screencast[3] The input for the approach is an event log of a process model simulated as discussed previously, from which we mine the process model using one of the ProM process mining algorithms. As we have now both process model in the form of Petri net (Figure 2), and corresponding event log, we can start the discovery of decisions. The screencast reflects our step-by-step approach proposed for the discovery of decisions from event logs, which is described below.

1. Discovery of control flow decisions According to our approach, the program firstly identifies two control flow decisions: (1) $p1$ with decision alternatives *Full check, Standard check*, and *No check*; and (2) $p3$ with decision alternatives *Send approval* and *Send rejection*. A corresponding decision tree is constructed for $p1$ (omitted in this report, but can be found in our paper [2]).

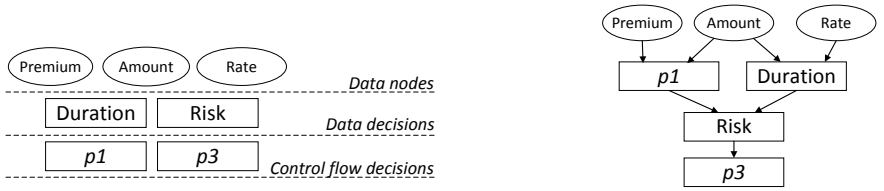
2. Discovery of data decisions Executing our algorithm further, the program finds: (1) A rule-based decision *Risk*; (2) A functional decision *Duration*.

The aggregate of the decisions discovered by the program is presented schematically in Figure 3a. Those are the elements which are used further for the construction of the decision requirements diagram: (1) Data nodes (*Premium, Amount, Rate*); (2) Data decisions (*Duration, Risk*); and (3) Control flow decisions ($p1$, $p3$). Additionally, the plug-in creates the decision table for each decision found.

3. Discovery of decision dependencies Further, the program mines the dependencies between the discovered decisions from Figure 3a, and it outputs the fully specified DMN decision model as depicted in Figure 3b. Thus, the program finds

[2] http://www.promtools.org/ (last accessed 2016-10-20).

[3] https://bpt.hpi.uni-potsdam.de/foswiki/pub/Public/WebHome/DMNanalysis.mp4 (last accessed 2016-10-20).



(a) Discovered decisions

(b) Discovered decision requirements diagram

Figure 3: The discovered decisions and the DMN model for the example process.

the trivial dependencies between the decisions *Duration* and *Risk*, as well as between *Risk* and $p3$, also, the non-trivial dependency between the decisions $p1$ and *Risk*. In case of circular dependencies, a random decision is kept.

The extracted decision model (Figure 3b) shows explicitly the decisions corresponding to the process from Figure 2, and thus, could serve for compliance checks by explaining the taken decisions. Also, the derived decision model can be executed complementary to the process model, thereby supporting the principle of separation of concerns.

4 Work in Progress

In this report we presented a short overview of our earlier work on challenges in modeling decisions complementary to business processes, and discovery of decision models from event logs. The extension of the presented work and corresponding planned activities are presented below:

1. Fuzzy Decision Mining Often decisions are not taken based on boolean logic but on manual and automatic assessment of inputs. This behavior can be represented more accurately by fuzzy rules than by traditional boolean logic. In fuzzy rules probabilities are used to map values to so called linguistic literals [10]. Linguistic literals are strings, that in general encode the semantic meaning behind a value range e.g. that a certain value is considered high or low. Since the meaning can be derived directly from the representation, it improves the interpretability for human users. Moreover, using literals across multiple rules allows a simple and consistent adaptation of all rules by adjusting the underlying mappings.

DMN offers extensive capabilities to model decisions and their dependencies, but it does not support fuzzy decisions yet. Furthermore, many corporations do not model their decisions explicitly, therefore decision model mining [2] has been introduced. While there are approaches to mine fuzzy decisions [5], there are no methods to mine Decision Requirement Diagram with a fuzzy rule base. Thus, our plan is to investigate an approach, which would combine the mining of fuzzy decisions with Decision Requirement Diagram to close this gap.

2. Optimal Acquisition of Input Data for Decision Taking in Processes Once processes and decisions are designed, they need to be executed. There exist real-world situations, when a set of information inputs is available, and acquisition of only a subset of it might be enough to make a decision. For example, in a process of assessment of a credit or an insurance application, depending on the case, there may be questions to an applicant which could be asked firstly in order to make a decision faster. With that, the existing literature on modeling and executing decisions complementary to business processes, including the Decision Model and Notation [8], do not provide guidelines on how to optimally acquire the inputs needed for decision making. Thus, prioritization of inputs acquisition for enabling efficient decision execution remains an open question in the context of processes.

To address the described problem, we investigate the question of acquisition of decision inputs utilizing DMN decision tables, acknowledged as one of the most powerful tools for representing decision logic [8]. In particular, we propose an approach for executing a decision model during instantiation of the associated process model aimed at (1) reducing the number of inputs to be acquired; (2) finding an optimal order of acquisition of decision inputs. We firstly propose to exclude pre-existing process instance data from the set of inputs to be acquired during decision execution. Next, we present an algorithm for constructing a decision tree which stores an optimal ordering of acquisition of inputs needed for decision execution. Thereby, as optimization criteria we use costs associated with inputs acquisition, and predictions about decision outcomes.

5 Publications 2016

Published:

- E. Bazhenova, S. Buelow, and M. Weske. "Discovering Decision Models from Event Logs". In: *Business Information Systems: 19th International Conference, BIS 2016, Leipzig, Germany, July, 6-8, 2016, Proceedings*. Edited by W. Abramowicz, R. Alt, and B. Franczyk. Cham: Springer International Publishing, 2016, pages 237–251. ISBN: 978-3-319-39426-8. DOI: 10.1007/978-3-319-39426-8_19.

- L. Janssens, E. Bazhenova, J. De Smedt, J. Vanthienen, and M. Denecker. "Consistent Integration of Decision (DMN) and Process (BPMN) Models". In: *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), Ljubljana, Slovenia, June 13-17, 2016.* 2016, pages 121–128.

Under review:

- E. Bazhenova and M. Weske. *Optimal Acquisition of Input Data for Decision Taking in Business Processes*. Submitted to the 32nd ACM Symposium on Applied Computing.

- E. Bazhenova, K. Batoulis, L. Janssens, J. Vanthienen, J. De Smedt, and M. Weske. *Research Challenges for the Integration of Business Process and Decision Management*. Submitted to the Information Systems Research Journal.

References

[1] E. Bazhenova, K. Batoulis, L. Janssens, J. Vanthienen, J. De Smedt, and M. Weske. *Research Challenges for the Integration of Business Process and Decision Management*. Submitted to the Information Systems Research Journal.

[2] E. Bazhenova, S. Buelow, and M. Weske. "Discovering Decision Models from Event Logs". In: *Business Information Systems: 19th International Conference, BIS 2016, Leipzig, Germany, July, 6-8, 2016, Proceedings*. Edited by W. Abramowicz, R. Alt, and B. Franczyk. Cham: Springer International Publishing, 2016, pages 237–251. ISBN: 978-3-319-39426-8. DOI: 10.1007/978-3-319-39426-8_19.

[3] E. Bazhenova and M. Weske. *Optimal Acquisition of Input Data for Decision Taking in Business Processes*. Submitted to the 32nd ACM Symposium on Applied Computing.

[4] B. von Halle and L. Goldberg. *The Decision Model: A Business Logic Framework Linking Business and Technology*. Taylor and Francis Group, 2010.

[5] F. Hoffmann, B. Baesens, C. Mues, T. Van Gestel, and J. Vanthienen. "Inferring descriptive and approximate fuzzy rules for credit scoring using evolutionary algorithms". In: *European Journal of Operational Research* 177.1 (2007), pages 540–555.

[6] L. Janssens, E. Bazhenova, J. De Smedt, J. Vanthienen, and M. Denecker. "Consistent Integration of Decision (DMN) and Process (BPMN) Models". In: *Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), Ljubljana, Slovenia, June 13-17, 2016.* 2016, pages 121–128.

[7] E. Kornyshova and R. Deneckère. "Decision-making Ontology for Information System Engineering". In: *ER*. Springer-Verlag, 2010.

[8] OMG. *Decision Model And Notation (DMN), v. 1.1*. 2016.

[9] A. Rozinat and W. van der Aalst. "Decision Mining in ProM". In: *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*. 2006, pages 420–425. DOI: 10.1007/11841760_33.

[10] J. Vanthienen, G. Wets, and G. Chen. "Incorporating fuzziness in the classical decision table formalism". In: *International journal of intelligent systems* 11.11 (1996), pages 879–891.

[11] M. Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012, pages I–XV, 1–403. ISBN: 978-3-642-28615-5.

[12] A. Zarghami, B. Sapkota, M. Z. Eslami, and M. van Sinderen. "Decision as a Service: Separating Decision-making from Application Process Logic." In: *EDOC*. IEEE, 2012. ISBN: 978-1-4673-2444-1.

Runtime data-driven software evolution in enterprise software ecosystems

Runtime models for self-adaptive monitoring

Thomas Brand

System Analysis and Modeling
Hasso-Plattner-Institut
thomas.brand@hpi.uni-potsdam.de

There are several reasons for changing and thus adapting the configuration for monitoring of a software system, for example if the setup of the system itself was changed. This report will look at self-adaptive monitoring in the context of the evolution of an enterprise software product. In doing so the interrelationship between the quality of knowledge about software systems and the required resources to obtain it shall be considered as the driver for self-adaptive monitoring. This report deals with runtime models as knowledge stores for self-adaptive monitoring and introduces a tool for related experiments.

1 Introduction

The reasons for adapting the monitoring configuration of an enterprise software system can be manifold. Monitoring can for example be intensified when investigating incidents and anomalies. An approach how self-adaptive monitoring can be applied for this purpose is described in [3]. Also changed information demands or an altered system setup can make an adaptation of the monitoring configuration necessary. An approach how to automatically adapt the monitoring to changing requirements is described in [7].

Another scenario for adaptive monitoring is the situation when the number of possible measurement points is larger than the number of measurements that can or shall be performed in parallel. In this case the measurements need to be performed sequentially in order to cover all measurement points and to gain an overall measurement result. Additional information about the measurement points and suitable algorithms for deciding where to measure when and how may help to increase the quality of the measurement result.

The report at hand will closer look into the just mentioned scenario. The hypnosis behind the effort is, that focusing the activity of intensive monitoring to alternating parts of running software systems can save resources and yield similar results compared to those from thoroughly monitoring of the system. This requires controlling the focus sensibly. Software system runtime models can allow such controlling to be performed in a computer-aided or automated way. An idea on how to build-up such a runtime model from runtime data is further discussed in this report.

Knowledge about how customers actually use the products of a software manufacturer and its ecosystem partners is a valuable input for well-grounded software

evolution decisions. The gathered information can be used to validate assumptions and over time detect changes in the users' behavior. But the effort that can be allowed for obtaining this information is likely limited as in the scenario described above. Thus related data might not be obtained and processed continuously and simultaneously for the entire software system or across the whole population of software product installations.

This report also contains a description of a simulator for later experiments in this research field. The tool allows simulating a population of enterprise application installations. Those simulated software systems may differ in the configuration of the software product and in the users' behavior.

2 Runtime model for self-adaptive monitoring of enterprise software systems

The Merriam-Webster's Dictionary defines the verb to monitor as "to watch, keep track of, or check usually for a special purpose" [6]. In order to do this in an automated fashion some sort of controllable sensors are required. The controlling entity needs to be aware of the measurement points, where those sensors are or can be placed as well as of the sensors' status. This knowledge can be captured in a specific runtime model which represents this aspect of the running software system. The controller may for example use the model as input for decision making.

A definition of runtime models is given in [1]. The following two subsections contrast this definition with the characteristics that at least some installations of major enterprise software products show. This allows illustrating challenges with runtime models for this type of software system and why those systems might need to be treated as black boxes when creating or maintaining their runtime models. The third subsection will than point out the importance of the runtime model accuracy for the measurement results of self-adaptive monitoring. The fourth subsection will indicated how augmenting runtime models with additional information might lead to better measurement results.

2.1 Creating a runtime model

The runtime model of a software system might not be intrinsically tied to the models produced as artifacts from the Model Driven Engineering (MDE) process as requested in [1]. Not for every part of an enterprise application such models might be available or contain the necessary information, especially if the system is composed of components from different parties of the software ecosystem. This then has the implication that the sensors runtime model needs to be created without models or other artifacts from the software development process.

2.2 Maintaining a runtime model

The runtime model might not be a causally connected self-representation of the associated system as defined in [1] – at least not bi-directionally. The challenge is to recognize changes in the software system without a corresponding notification mechanism. An Enterprise Java Beans (EJB) specific example of such an advantageous notification mechanism is described in [2] and [8]. It propagates notifications about changes in the EJB-application when they occur and thus they can promptly be reflected in the runtime model. But such a change notification mechanism might not be available for example due to historic, technological or economical reasons. Then changes have to be detected by observing the system to reflect them in the runtime model, for example by analyzing the runtime data of the system.

This implies that the runtime model might represent the related software system less accurate. Either because not every change was detected or the detection requires time and thus the model is only updated with a delay. The quality of the model as representation of the system depends on the effort spent to keep it up-to-date.

That implication might be an issue for most use cases of runtime models. But there might be also several cases where a vague runtime model is sufficient. In such cases the benefits of controllable and moderate costs for validating the model against the status of the underlying system outweigh the disadvantages of uncertainty. This is for example the case when the purpose of monitoring is usage measurement for software evolution decisions. Because in this case the uncertainties about a software system get leveled out by the duration of the observation and the aggregation of the measurement results across the population of observed software product installations.

2.3 Runtime model and measurement result quality

Knowledge about the available measurement points and their status is required in order to control and adapt monitoring. Such knowledge can be captured in a runtime model. The accurateness and up-to-dateness of the knowledge and thus of the runtime model may also impact the quality of the monitoring measurement results. For example if the monitoring controller is not aware of a measurement point through the runtime model it cannot activate it to measure. Or if a sensor has been removed by a third party but remains enlisted as active in the runtime model then the controller might not utilize all available resources for monitoring as it could have already activated another measurement point instead.

2.4 Augmented runtime models

Assuming it is not feasible to measure at all known and required measuring points continuously and in parallel then decisions need to be made about where to measure when and how. Coincidence is one way of selecting measurements points in space and points in time when to measure. But especially if the measured object has more and less dynamic parts or also intermediary measurement results matter

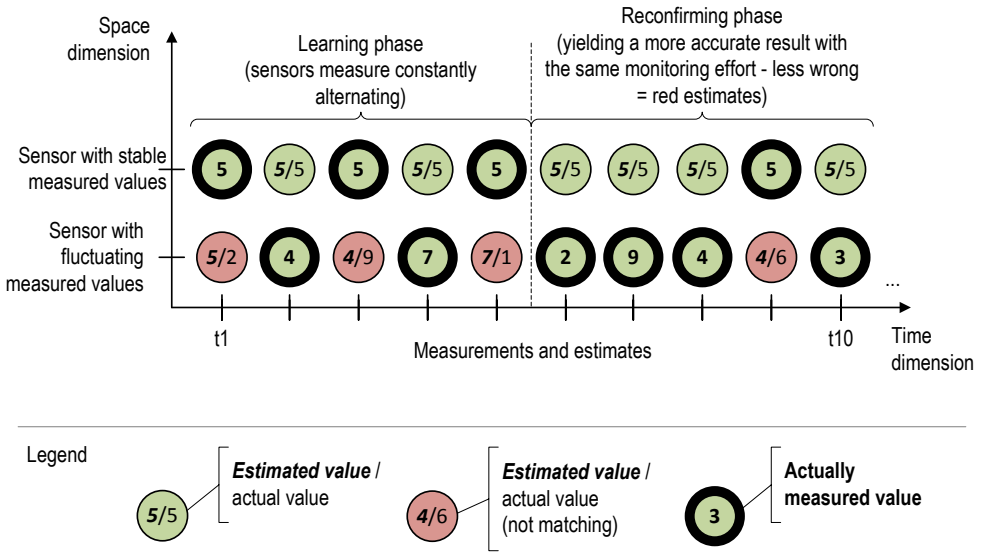


Figure 1: Depicts a scenario where at a point of time only one sensor out of two can measure, while for the other sensor an value needs to be estimated. If additional information about previous measurements is provided through the sensors runtime model then the manager for adaptive monitoring can improve the measurement result during the reconfirming phase by activating the sensors sensibly and thus reduce the number of wrong estimates.

then considering additional information might yield better monitoring results. A runtime model might be augmented or linked to such additional information for more sophisticated decision making.

This information may for example comprise insights from earlier measurements, such as the correlation of measured values. An interesting relationship in the space dimension might be for example that the measured values from multiple measuring points usually change proportionally. In the time dimension an interesting information might be that the measured values of a particular measurement point are usually stable and not very fluctuating. This information would allow substituting some actual measurements with well estimated values as illustrated in Figure 1.

Comparatively little monitoring resources would then be required to reconfirm the correlations from time to time once they had been discovered. This would free resources for more reasonable monitoring tasks.

3 Enterprise Application Simulator

The research effort related to this report focuses on the evolution of enterprise software products. M. Fowler describes such enterprise applications in [4] as being about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data. He also emphasizes

concurrency caused by multiple users, the integration with other applications and a heterogeneous technology mix as well as a complex business logic.

In this section a lightweight simulator for enterprise applications is presented because using actual software systems for conducting first experiments is impractically. For example software customers may not want to share their data due to privacy reasons. For the experiments regarding self-adaptive monitoring it is also necessary to have control over the monitoring configuration of the software systems and to be able to continuously change it according to the requirements of the experiments.

Setting up and operating diverse configurations of an enterprise software product in a laboratory environment would required product specific skills and comparatively many hardware resources. Also it might be cumbersome to simulate specific scenarios during the experiments when using a function rich and complex software product. Those circumstances led to the implementation of the Enterprise Application Simulator (EAS).

The tool allows simulating a population of enterprise software systems. Those systems shall represent customer specific installations of software products, which consist of software components provided by the software manufacturer or third party software ecosystem partners as symbolized at the top of Figure 2. As depicted at the bottom of this figure each system shall possess an individualized configuration comprising a set of software components. The configurations shall be diverse – similar to how they would be in reality.

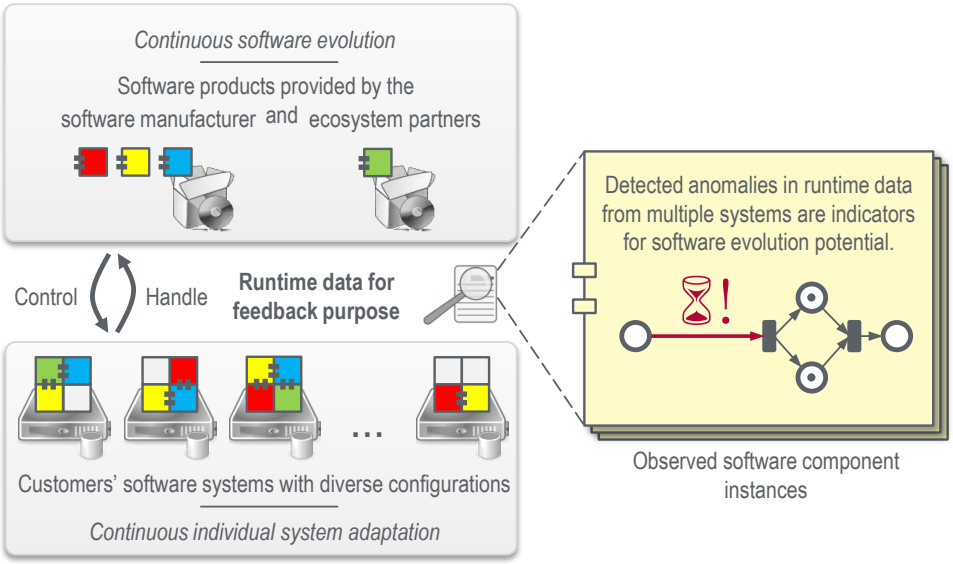


Figure 2: Population of simulated software systems for researching how to control, handle and analyze runtime data. The configurations of the systems shall represent diverse sets of software components provided from different parties of the software ecosystem.

3.1 Simulator foundation

In order to ease simulator logic development and operation, the EAS is based on the IBM Integration Bus (IIB) and IBM MQ (IMQ) middleware. Both IBM software products are usually used for enterprise application integration and for dealing with large amounts of complex data. The software is available for research purposes through the IBM Academic Initiative [5].

IMQ formally called WebSphere MQ and MQSeries provides a transport mechanism for program intercommunication based on messaging. It supports reliable and asynchronous message exchange using message queues with support for transactions. Messages can also be distributed to multiple consumers using the publish subscribe messaging pattern. Employing messaging for the simulator allows for example to easily observe, pause and manipulate the communication.

Logic to route messages, for example by moving them between message queues can be implemented using the IBM Integration Bus formally known as WebSphere Message Broker. Besides routing it provides support for message validation and transformation. Its visual modeling capabilities make it easy to implement simulator logic as well. It also supports operating multiple server instances on which different simulated enterprise applications can be executed.

Furthermore the IIB provides comprehensive extension possibilities, which additionally qualify it as a foundation for the EAS.

3.2 Simulator capabilities

Support for development and operation The IIB Toolkit allows to visually model so called message flows. For the EAS this capability is utilized to model the behavior of software product components with a UML activity diagram-like notation. In Figure 3 an example of such a model is pointed out by the green marker *A*. The available notation elements are marked with a *B*. They are a simulator specific extension to the IIB.

Once the components of a software product have been modeled, the individual enterprise software system configurations need to be created. For this purpose the EAS utilizes the concept of subflows supported by the IIB. The modeled software product components can be added as such subflows to a component diagram which represents the configuration of a particular software system. An example of such a diagram is pointed out with the green marker *C* in Figure 3.

After completing a software system configuration it can be packaged and deployed as an IIB application on a so called integration server. Those servers are runtime environments for message flows and other logic. In the context of the EAS they represent a software system. Such a server is depicted in Figure 3 close to the marker *D*. The IIB Toolkit also provides comprehensive debugging support.

Support for traceability Traceability is required to understand what actually happened during the execution of the EAS and to contrast those insights with the monitoring and runtime data processing results of the conducted experiments. The EAS

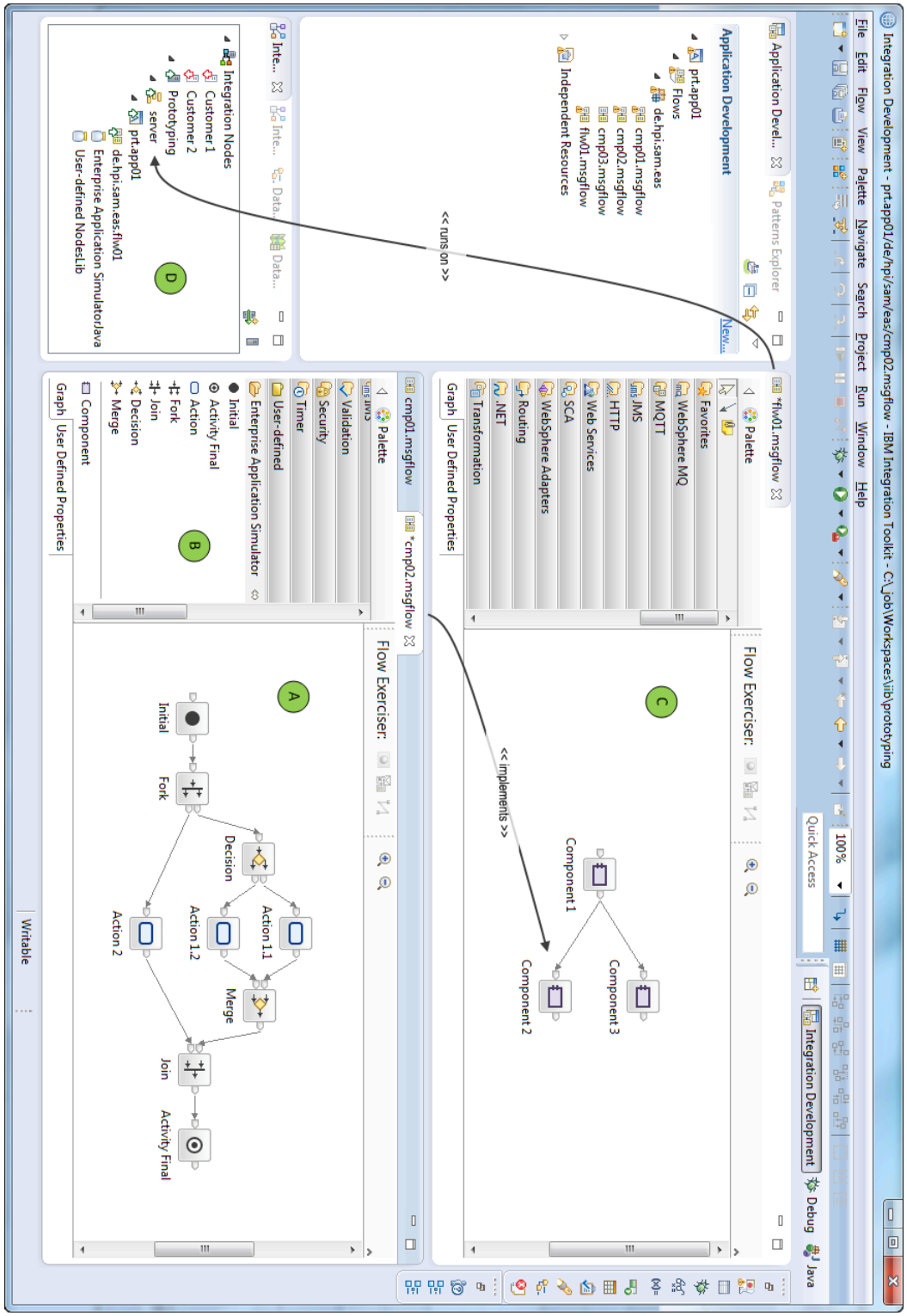


Figure 3: IBM Integration Bus (IIB) Toolkit used to visually model and operation Enterprise Application Simulator (EAS) logic.

uses a chain of messages to simulate the workflow of a business transaction. Each node which is used to model the behavior of a simulated software component creates a copy of the message it received from the preceding node. Examples of those nodes can be seen in Figure 4 in the subelements compartment of *Component n*. The full list of nodes is available in Figure 3 at marker *B*. Besides other potential message changes, a node always appends an entry to the history list of the new message about the passage of the workflow before the node forwards the message to the next node. All messages with the same workflow ID are related to the same business transaction. Thus the final output message of a simulated business transaction contains the complete protocol of the related workflow.

Once a simulated software component has finished its work, it then publishes a message about the event to a topic, please see the *Activity Final* node in Figure 4. Other components interested in the event can thereupon start their processing, for example *Component n+1* in the same figure. For tracing purposes the *Event Detector* archives all events in the *Event Store* and thus keeps track about every component intercommunication.

Furthermore regular logging is supported, such as logging in Java classes.

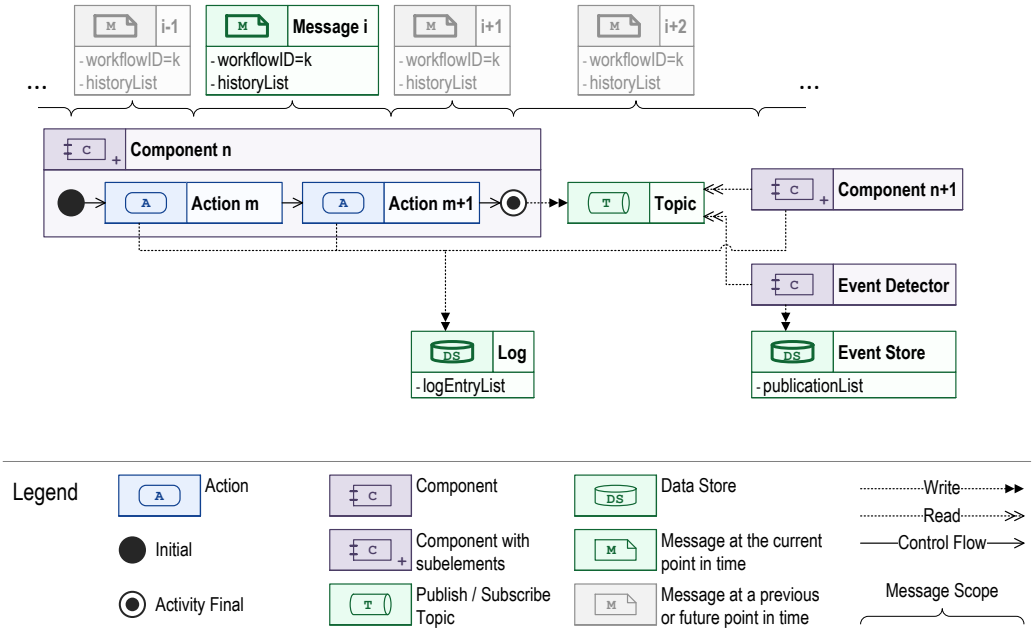


Figure 4: The traceability support of the Enterprise Application Simulator (EAS) is depicted in a simplified manner. It comprises logging, storing events from the intercommunication of the simulated software components and keeping a history protocol for each business transaction workflow, which is maintained and passed on with the corresponding messages. The figure also indicates implementation details about the publish/subscribe- and thus topic-based intercommunication between the simulated software components.

Support for concurrency Concurrency in the context of the EAS means that parts of a workflow can be processed in parallel. A workflow shall mean an instance of a business process to handle one business transaction. To enable concurrency inside a simulated software component the *Fork* and *Join* message flow nodes can be used in the activity diagram as depicted in Figure 3, markers *A* and *B*. The *Fork* causes two threads to be created which work independently. The *Fork* and *Join* nodes must not be confused with the *Decision* and *Merge* nodes which enable alternative routes through a business process. Another way to cause concurrency is to connect more than one component to an output terminal of another component in the component diagram as depicted in Figure 3, marker *C*. In this example the components 2 and 3 are connected to the output terminal of component 1. Thus both receive the event that component 1 completed its work via the publish subscribe mechanism and start their processing independently.

Support for structural adaptation during runtime Structural adaptation shall allow simulating architectural changes of a software system during runtime. Structural adaptation is currently enabled by allowing to change during runtime how components are wired together. The initial configuration gets specified through the component diagram, which is shown in Figure 3, marker *C*. During runtime it is possible to change this configuration and alter which other components get triggered by a work-completed-event published by a component. For this and parameter adaptation purposes an EAS extension creates typically one IIB user-defined configurable service for each simulated software component. Accessing those configurable services through IIB APIs or its web user interface allows altering the corresponding configuration for each component at runtime.

Support for changing behavior during runtime Changing the behavior can be achieved through parameter adaptation during runtime, for example by altering a threshold for a routing decision. Parameters are accessible via IIB user-defined configurable services, which have already been described above. The parameter values can be modified through IIB APIs or its web user interface.

Support for complex simulation logic The IIB supports manipulating and handling messages in various ways and thus allows implementing very sophisticated simulation logic for the software components. After this logic has been implemented for example with the Java Standard Edition and the IIB related Java APIs it could be triggered through the *Action* nodes depicted in the activity diagram in Figure 3, marker *A*.

4 Brief summary and outlook

In its first part this report points out how adaptive monitoring might be useful for efficient usage measurement, which itself can serve as a basis for sound software evolution decisions. It explains the benefit of a runtime model as knowledge store for

adaptive monitoring and the idea to mine the required runtime model from runtime data produced by the observed software system. To elaborate further on those ideas experiments about adaptive monitoring shall be conducted with the EAS presented in the second part of the report.

References

[1] G. Blair, N. Bencomo, and R. B. France. "Models@Run.Time". In: *Computer* (2009). DOI: 10.1109/MC.2009.326.

[2] J. Bruhn and G. Wirtz. "mKernel: A Manageable Kernel for EJB-based Systems". In: *Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007. ISBN: 978-963-9799-09-7.

[3] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring. "Self-adaptive Software System Monitoring for Performance Anomaly Localization". In: *Proceedings of the 8th ACM International Conference on Autonomic Computing*. ACM, 2011. DOI: 10.1145/1998582.1998628.

[4] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002. ISBN: 0-321-12742-0.

[5] *IBM Academic Initiative*. Program details. URL: https://developer.ibm.com/academic/programdetails (last accessed 2016-10-01).

[6] *Merriam-Webster's online dictionary*. Defintion of the verb to monitor. URL: http://www.merriam-webster.com/dictionary/monitoring (last accessed 2016-09-22).

[7] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas. "DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems". In: *Software Engineering for Self-Adaptive Systems II*. Springer-Verlag Berlin, Heidelberg, 2013. DOI: 10.1007/978-3-642-35813-5_11.

[8] T. Vogel and H. Giese. "Adaptation and Abstract Runtime Models". In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2010. DOI: 10.1145/1808984.1808989.

Power of Greediness on Real World network

Ankit Chauhan

Algorithm Engineering
Hasso-Plattner-Institut
ankit.chauhan@hpi.uni-potsdam.de

Large real-world networks typically follow a power-law degree distribution. To study such networks, numerous random graph models have been proposed. However, real networks are not drawn at random. In fact, the behavior of real-world networks and random graph models can be completely opposite. Brach, Cygan, Łacki, and Sankowski [SODA 2016] introduced two natural deterministic conditions: (1) a power-law upper bound on the degree distribution (PLB-U) and (2) power-law neighborhoods, that is, the degree distribution of degrees of neighbors of each vertex is also upper bounded by a power law (PLB-N). They showed that many real-world networks satisfy both deterministic properties and exploit them to design faster algorithms for a number of classical graph problems like transitive closure, maximum matching, determinant, PageRank, matrix inverse, counting triangles and maximum clique.

We complement the work of Brach et al. by showing that a number of well-studied random graph classes exhibit both aforementioned PLB-properties and additionally also a power-law lower bound on the degree distribution (PLB-L). In this work we study three classical NP-hard combinatorial optimization problems on deterministic PLB networks. It is known that on general graphs with maximum degree Δ, a greedy algorithm, which chooses nodes in the order of their degree, only achieves a $\Omega(\ln \Delta)$-approximation for Minimum Vertex Cover and Minimum Dominating Set, and a $\Omega(\Delta)$-approximation for Maximum Independent Set. We prove that the PLB-U property suffices such that the greedy approach achieves a constant-factor approximation for all three problems. We also show that a PTAS cannot be expected, even if all three PLB-properties hold. For all three combinatorial optimization problems we prove APX-completeness for graphs with PLB-U, PLB-L and PLB-N property.

1 Overview

A wide range of real-world networks exhibit a degree distribution that resembles a power-law [3, 20]. This means that the number of vertices with degree k is proportional to $k^{-\beta}$, where $\beta > 1$ is the power-law exponent, a constant intrinsic to the network.This applies to Internet topologies [11], the Web [5, 17], social networks [1], power grids [21], and literally hundreds of other domains [19]. Networks with a power-law degree distribution are also called scale-free networks and have been widely studied.

To capture the degree distribution and other properties of scale-free networks, a multitude of random graph models have been proposed. These models include Preferential Attachment [5], the Configuration Model [2], Chung-Lu Random Graphs [8]

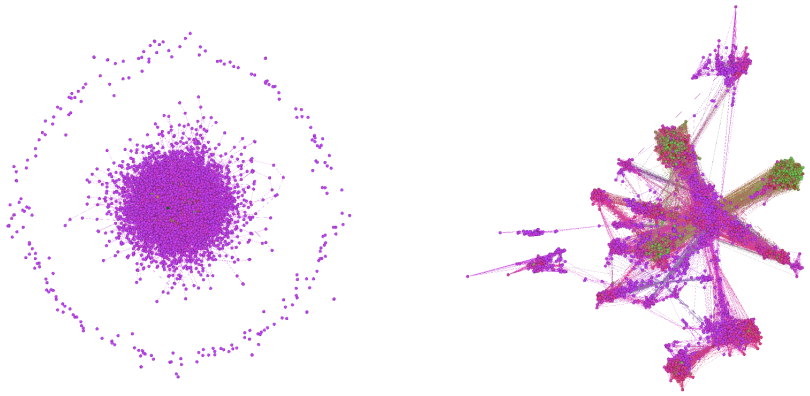


Figure 1: Graph generated by Chung-Lu Model vs. the Snippet of the DBLP network [18].

and Hyperbolic Random Graphs [16]. Despite the multitude of random models, none of the models truly has the same set of properties as real world networks.

This motivates studying deterministic properties of scale-free models, as these deterministic properties can be checked for real-world networks. To describe the properties of scale-free networks without the use of random graphs, [2] define (α, β)-Power Law Graphs. The problem of this model is that it essentially demands a perfect power-law degree distribution, whereas the degree distributions of real networks normally exhibit slight deviations from power-laws. Therefore, (α, β)-Power Law Graphs are too constrained and do not capture most real networks (Figure 1). To allow for those deviations in the degree distribution [6] define buckets containing nodes of degrees $\left[2^i, 2^{i+1}\right)$. Now we define these properties formally,

Definition 1 (PLB-U [6])**.** *Let G be an undirected n-vertex graph and $c_1 > 0$ be a universal constant. We say that G is power law bounded (PLB-U) for some parameters $1 < \beta = \mathcal{O}(1)$ and $t \geqslant 0$ if for every integer $d \geqslant 0$, the number of vertices v, such that $\deg(v) \in \left[2^d, 2^{d+1}\right)$ is at most*

$$c_1 n(t+1)^{\beta-1} \sum_{i=2^d}^{2^{d+1}-1} (i+t)^{-\beta}.$$

Definition 2 (PLB-L)**.** *Let G be an undirected n-vertex graph and $c_2 > 0$ be a universal constant. We say that G is power law bounded PLB-L for some parameters $1 < \beta = \mathcal{O}(1)$ and $t \geqslant 0$ if for every integer $\lfloor \log d_{min} \rfloor \leqslant d \leqslant \lfloor \log \Delta \rfloor$, the number of vertices v, such that $\deg(v) \in \left[2^d, 2^{d+1}\right)$ is at least*

$$c_2 n(t+1)^{\beta-1} \sum_{i=2^d}^{2^{d+1}-1} (i+t)^{-\beta}.$$

Since PLB-U property alone can capture a much broader class of networks, for example empty graphs and rings hence this lower-bound is important to restrict networks to real world power-law networks. In definition of PLB-L d_{min} is necessary because in real world power law network minimum degree is not always 1.

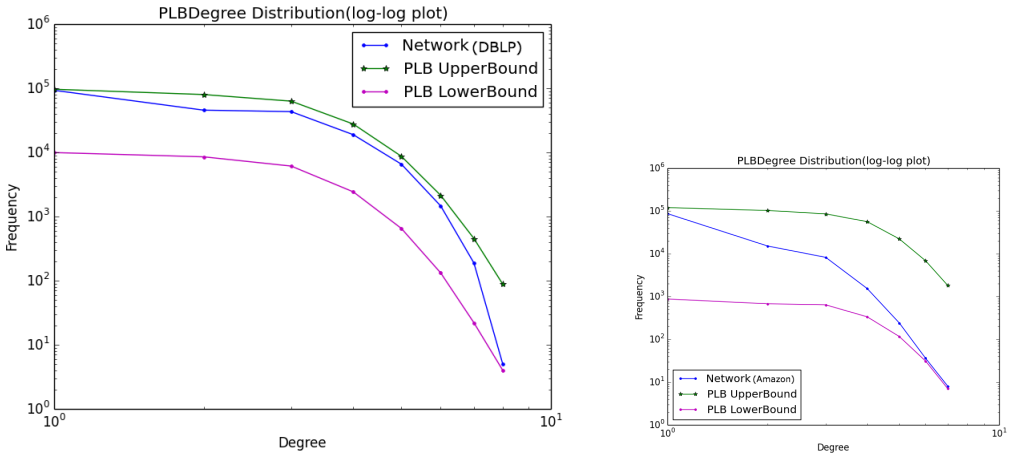


Figure 2: PLB-U and PLB-L properties in real world networks.

Definition 3 (PLB-N [6]). *Let G be a PLB (multi-)graph with parameters $\beta > 2$ and $t \geqslant 0$, and let $c_2 > 0$ be an universal constant. We say that G has PLB neighborhoods (PLB-N) if for every vertex v of degree k, the number of neighbors of v of degree at least k is at most $c_3 \max\left(\log n, (t+1)^{\beta-2} k \sum_{i=k}^{n-1} i(i+t)^{-\beta}\right)$.*

2 Approach

In this section we will take a glance at the techniques and useful lemma needed for proving our main lemma 5. From it we will be able to derive bounds on the size of solutions of covering problems as well as better approximation guarantees for the greedy dominating set algorithm.

Definition 4. *A greedy algorithm is an* α-approximation *for problem P if it produces a solution set S with $\alpha \geqslant \frac{|S|}{|OPT|}$ if P is a minimization problem and with $\alpha \geqslant \frac{|OPT|}{|S|}$ if P is a maximization problem.*

Now we state our main lemma,

Lemma 5 (Potential Volume Lemma). *Let G be a (multi-)graph with the PLB-U property for some $\beta > 2$, some constant $c_1 > 0$ and some constant $t \geqslant 0$. Let S a solution set for which we can define a function $g\colon \mathbb{R}^+ \to \mathbb{R}$ continuously differentiable and $h(x) := g(x) + C$ for some constant C such that*

(i) g *non-decreasing,*

(ii) $g(2x) \leqslant c \cdot g(x)$ *for all* $x \geqslant 2$ *and some constant* $c > 0$,

(iii) $g'(x) \leqslant \frac{g(x)}{x}$,

then it holds that $\sum_{x\in S} h(\deg(x))$ *is at most*

$$\left(c\frac{(\beta-1)^2}{(\beta-2)\left(1-\left(\frac{t+2}{t+1}\right)^{1-\beta}\right)} g\left(\left(c_1\frac{\beta-1}{\beta-2}\frac{n}{M}\cdot 2^{\beta-1}\cdot(t+1)^{\beta-1}\right)^{\frac{1}{\beta-2}}\right)+C\right)\cdot|S|,$$

where $M(n)\geqslant 1$ *is chosen such that* $\sum_{x\in S}\deg(x)\geqslant M$.

2.1 Minimum Dominating Set

The idea for lower-bounding the size of a dominating set is essentially the same as the one by [23] and by [13] in the context of (α,β)-Power-Law Graphs.

Corollary 6. *For a multigraph without loops and isolated vertices and with the PLB-U property with parameters* $\beta>2$, $c_1>0$ *and* $t\geqslant 0$, *every dominating set has an approximation factor of at most*

$$2\frac{(\beta-1)^2}{(\beta-2)\left(1-\left(\frac{t+2}{t+1}\right)^{1-\beta}\right)}\left(\left(c_1\frac{\beta-1}{\beta-2}\cdot 2^{\beta}\cdot(t+1)^{\beta-1}\right)^{\frac{1}{\beta-2}}\right)+1$$

2.1.1 The Greedy Algorithm

Corollary 6 says that simply taking all nodes already gives a constant approximation factor, but now we want to show that using the classical greedy algorithm actually guarantees an even better approximation factor. To understand what happens, we shortly recap the algorithm.

Algorithm 1: Greedy Dominating Set

Require: undirected graph $G=(V,E)$

$C\leftarrow\emptyset$
$D\leftarrow\emptyset$
while $|D|<|V|$ **do**
  $u\leftarrow \operatorname{argmax}_{v\in(V\setminus C)}\left(N^+(v)\setminus D\right)$
  $C\leftarrow C\cup\{u\}$
  $D\leftarrow D\cup N^+(u)$
end
return C

The following is an adaptation of the proof for the greedy Set Cover algorithm to the case of unweighted Dominating Set.

Theorem 7 ([15]). *Let* S *the solution of the greedy algorithm and* OPT *an optimal solution for* Dominating Set. *Then it holds that*

$$|C|\leqslant\sum_{x\in OPT} H_{\deg(x)+1},$$

Table 1: Comparison of the approximation ratios achieved by greedy algorithms on networks with an upper bound on the power-law degree distribution (PLB-U) and on general graphs. While on general graphs, greedy achieves only a logarithmic or polynomial approximation, greedy achieves a constant-factor-approximation on graphs with PLB-U.

Problem	**General Graph**	**Graphs with PLB-U**
Minimum Dominating Set	$\mathcal{O}(\ln \Delta)$ [15]	$\Theta_n(1)$
Minimum Vertex Cover	$\mathcal{O}(\ln \Delta)$	$\Theta_n(1)$
Maximum Independent Set	$\mathcal{O}(\Delta)$ [10]	$\Theta_n(1)$
Minimum Connected Dominating Set	$\mathcal{O}(\ln \Delta)$ [22]	$\Theta_n(1)$

where H_k is the k-th harmonic number.

Corollary 8. *The greedy algorithm gives a $H_{\Delta+1}$-approximation for Dominating Set, where Δ is the maximum degree of the graph.*

Also, we can use similar argument to prove the corollary below.

Corollary 9. *The greedy algorithm gives a H_Δ-approximation for Vertex Cover, where Δ is the maximum degree of the graph.*

Proof of corollary 6. Let OPT denote an arbitrary minimum dominating set. It holds that

$$\sum_{x \in \text{OPT}} \deg(x) + 1 \geqslant n$$

and since we assume that there are no nodes of degree 0, it also holds that

$$\sum_{x \in \text{OPT}} \deg(x) \geqslant \frac{n}{2},$$

giving us (iv) with $M := \frac{n}{2}$. We can choose $h(x) := x + 1$ with $g(x) = x$. Now g satisfies (i), (ii) with $c = 2$ and (iii). With lemma 5 we can now derive

$$\begin{aligned} n &\leqslant \sum_{x \in \text{OPT}} \deg(x) + 1 = \sum_{x \in \text{OPT}} h(\deg(x)) \\ &\leqslant \left(2 \frac{(\beta-1)^2}{(\beta-2)\left(1-\left(\frac{t+2}{t+1}\right)^{1-\beta}\right)} \left(\left(c_1 \frac{\beta-1}{\beta-2} \cdot 2^\beta \cdot (t+1)^{\beta-1} \right)^{\frac{1}{\beta-2}} \right) + 1 \right) \cdot |\text{OPT}| \end{aligned} \qquad \square$$

Since every vertex cover is also a dominating set hence following corollary follows:

Corollary 10. *For a multigraph without loops and isolated vertices and with the PLB-U property with parameters $\beta > 2$, $c_1 > 0$ and $t \geqslant 0$, every vertex cover has an approximation factor of at most*

$$2 \frac{(\beta-1)^2}{(\beta-2)\left(1-\left(\frac{t+2}{t+1}\right)^{1-\beta}\right)} \left(\left(c_1 \frac{\beta-1}{\beta-2} \cdot 2^\beta \cdot (t+1)^{\beta-1} \right)^{\frac{1}{\beta-2}} \right) + 1.$$

2.2 Hardness of Covering Problems on PLB-(U,L,N) Graphs

Brach et al. [6] showed that in the graphs with PLB-U and PLB-N properties where $\beta \geqslant 3$ finding maximum clique is polynomial time solvable. This proved that the finding maximum clique is not NP-hard for all the power-law graphs as proved in [12]. This arises the question whether this property helps for the covering problem. Unfortunately, answer for this is no. We give the embedding technique of cubic graphs to PLB-(U,L,N) graphs proving that the covering problems are not just NP-hard, they are APX-hard.

Table 2: Comparison of the approximation lower bounds for polynomial-time algorithms (assuming P $\neq$ NP) on networks with an upper (PLB-U) and lower (PLB-L) bound on the power-law degree distribution and with PLB neighborhoods (PLB-N) with the approximation lower bounds on general graphs. Even with the additional properties of PLB-L and PLB-N the problems on graphs with PLB-U remain APX-hard, i.e. these problems cannot admit a PTAS. Better lower bounds for each problem are in respective theorem, $\Omega(1)$ hides the PLB-L parameters β, t and constant c_2.

Problem	**General Graph**	**Graph with PLB-(U,L,N)**
Minimum Dominating Set (MDS)	$\Omega(\ln \Delta)$ [7]	$1 + \Omega(1)$
Minimum Vertex Cover (MVC)	$\geqslant 1.3606$ [9]	$1 + \Omega(1)$
Maximum Independent Set (MIS)	$\Omega(\Delta))$ [4]	$1 + \Omega(1)$

Definition 11 (Embedded-Approximation-Preserving Reduction [23])**.** *Given an optimal substructure problem O, a reduction from an instance on graph $G = (V, E)$ to another instance on a (power law) graph $G' = (V', E')$ is called* embedded approximation-preserving *if it satisfies the following properties:*

(1) G is a subset of maximal connected components of G';

(2) The optimal solution of O on G', $\textsc{opt}(G')$, is upper bounded by $C \cdot \textsc{opt}(G)$ where C is a constant correspondent to the growth of the optimal solution.

Having shown an embedded-approximation-preserving reduction, we can use the following lemma to show hardness of approximation.

Lemma 12 ([23])**.** *Given an optimal substructure problem O, if there exists an embedded-approximation-preserving reduction from a graph G to another graph G', we can extract the inapproximability factor δ of O on G' using ε-inapproximability of O on G, where δ is lower bounded by $\frac{\varepsilon C}{(C-1)\varepsilon+1}$ when O is a maximization problem and by $\frac{\varepsilon+C-1}{C}$ when O is a minimization problem.*

We will use this framework as follows: First, we show how to embed cubic graphs into graphs with PLB-U, PLB-L and PLB-N. Then, we derive the value of C as in lemma 12 for each problem we consider. Last, we use lemma 12 together with the known inapproximability results for the considered problems on cubic graphs to derive the approximation hardness on graphs with PLB-U, PLB-L and PLB-N.

We start by showing the embedding of cubic graphs into multigraphs with PLB-U, PLB-L and PLB-N. In the embedding, we will use the following gadget to fill up the degree sequence of our multigraphs.

Definition 13 ($\vec{d}$-Regular cycle $RC_n^{\vec{d}}$ [23]). *Given a degree sequence $\vec{d} = (d_1, \cdots, d_n)$, a $\vec{d}$-regular cycle $RC_n^{\vec{d}}$ is composed of two cycles. Each cycle has n vertices and the two i^{th} vertices in each cycle are adjacent to each other by $d_i - 2$ multi-edges. That is, a $\vec{d}$-regular cycle has $2n$ vertices and the two i^{th} vertices have degree d_i.*

Lemma 14. *Any cubic graph G can be embedded into a multigraph G_{PLB} having the PLB-U, PLB-L and PLB-N properties for any $\beta > 1$ and any $t \geqslant 0$.*

Proof. Suppose we are given β and t. We now want to determine c_1 and c_2 of PLB-U and PLB-L respectively. Let n be the number of nodes in graph G and let $N = cn$ be the number of nodes in G_{PLB} for some constant c to be determined. Also, we have to ensure that $N - n$ is even to get a graphical degree sequence since our gadgets always have an even number of nodes. To hide a cubic graph in the respective bucket of G_{PLB}, we need

$$c_1 N(t+1)^{\beta-1} \sum_{i=2}^{3} (i+t)^{-\beta} = c_1 N(t+1)^{\beta-1} \left(\frac{1}{(2+t)^\beta} + \frac{1}{(3+t)^\beta} \right) \geqslant n. \quad (1)$$

Also, we have to ensure to choose c_1 big enough so that the bucket containing the maximum degree Δ can hold two vertices. Otherwise we could not hide an appropriate $\vec{d}$-Regular cycle in that bucket, resulting in an empty bucket, which violates the PLB-L property for $c_2 > 0$. This second condition implies

$$c_1 N(t+1)^{\beta-1} \sum_{i=2^{\lfloor \log \Delta \rfloor}}^{2^{\lfloor \log \Delta \rfloor + 1} - 1} (i+t)^{-\beta} \geqslant 2. \quad (2)$$

As we will see, we can choose the constant c_1 arbitrarily large, so the former conditions are no real restrictions. Then we choose the maximum degree Δ such that

$$d_{max(G_{PLB})} = (cn)^{\frac{1}{\alpha-1}}.$$

In our embedding we just fill the buckets until they reach their respective lower bounds, except for bucket 1 which might get up to n nodes. Bucket 1 might get filled with a cycle of $\left\lceil c_2 N(t+1)^{\alpha-1} \left(\frac{1}{(2+t)^\alpha} + \frac{1}{(3+t)^\alpha} \right) \right\rceil - n$ nodes. By filling a bucket (other than bucket 1) we might deviate by at most two from the lower bound of that bucket. Then, we add additional gadgets of size two until we have exactly N nodes. To ensure

that this is possible we need the following inequality to hold true

$$\begin{aligned} & n + \sum_{d=0}^{\lfloor \log \Delta \rfloor} \left(2 + c_2 N (t+1)^{\beta-1} \sum_{i=2^d}^{2^{d+1}-1} (i+t)^{-\beta} \right) \\ & \leqslant \frac{N}{c} + 2 \log N^{\frac{1}{\beta-1}} + c_2 N (t+1)^{-1} + \frac{c_2}{\beta - 1} N \\ & \leqslant N \left(\frac{1}{c} + \eta + \frac{c_2}{t+1} + \frac{c_2}{\beta - 1} \right) \\ & \leqslant N, \end{aligned}$$

i.e. after filling all buckets to their lower bound, there is still some slack until we reach N. From this last condition we can derive

$$c \geqslant 1 + \frac{\eta + c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)}{1 - \eta - c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)} > 1 + \frac{c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)}{1 - c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)},$$

since η can be arbitrarily small. We choose $\eta = c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)$ to obtain

$$c = 1 + \frac{2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)}{1 - 2c_2 \left(\frac{1}{t+1} + \frac{1}{\beta-1} \right)}.$$

Now we can essentially choose c_1 arbitrarily large and c_2 arbitrarily small, guaranteeing a large enough gap to have a graphical degree sequence and to guarantee $c > 1$. At the same time our choice of c guarantees that we can fill the graph with exactly N nodes. Furthermore, since every node has a constant number of neighbors, G_{PLB} also fulfills PLB-N, which always allows us at least $c_3 \log N$ many neighbors. □

3 Related Work

Maximum Independent set and minimum dominating set have been studied widely on power law graphs. Ferrante et al. [12] proved that these problems remain NP-hard on power law graphs for $\beta > 0$. Also, in [23], Shen et al. proved that MIS, MDS are APX-hard for $\beta > 1$. Shen et al. also proved that for $\beta > 1$ there is no $1 + \frac{1}{1120\zeta(\beta)3^{\beta}} - \varepsilon$ for MIS and $1 + \frac{1}{3120\zeta(\beta)3^{\beta}}$ for MDS problem on power law graphs. Later Gast et al. in [13] proved that MDS give logarithmic approximation bound on the power law graph when $\beta \leqslant 1$. Then Hauptmann et al. in [14] give first non constant bound on approximation ratio of MIS for $\beta \leqslant 1$. Then analyse greedy algorithm for positive influence dominating sets on power law graph $2 < \beta < 3$ and proved greedy gives $1 + (\frac{1}{\beta-1} + \ln \sqrt{\beta - 2} \frac{\beta-1}{\beta-2})(1 - \frac{1}{\sqrt{\beta-2}^{\frac{\beta-1}{\beta-2}} - 1})^{1-\beta}$ approximation ratio. Wang et al. proposed greedy algorithm [24] for positive influence dominating sets on social networks and showed that $\ln(\Delta)$ approximation ratio for the greedy algorithms, where Δ is the maximum degree of the input graph. It is to be noted that all the result are proved on the (α, β)-power law model defined by Aiello et al. [2].

4 Future Work

As the deterministic property helped to identify why greedy algorithms good but it is still not known why the greedy algorithms give very good approximation than the our proved lower bound hence one of the direction in which I want to proceed, is to identify such more deterministic properties and analyse the algorithms. As, nobody have asked the questions of parameterized problems on the PLB-(U,L,N) graphs which seems very interesting to me and approaches I have tried for the parameterized problems have shown some positive results hence I will be working in this direction for a while. Also, after proving hardness I will try to give faster algorithms for real world networks by exploiting PLB properties.

References

[1] L. A. Adamic, O. Buyukkokten, and E. Adar. "A social network caught in the Web". In: *First Monday* 8.6 (2003). DOI: 10.5210/fm.v8i6.1057.

[2] W. Aiello, F. Chung, and L. Lu. "A random graph model for massive graphs". In: *32nd Symp. Theory of Computing (STOC)*. Portland, Oregon, USA, 2000, pages 171–180. ISBN: 1-58113-184-4. DOI: 10.1145/335305.335326.

[3] R. Albert and A.-L. Barabási. "Statistical mechanics of complex networks". In: *Reviews of modern physics* 74.1 (2002), page 47. DOI: 10.1103/RevModPhys.74.47.

[4] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. "Derandomized Graph Products". In: *Computational Complexity* 5.1 (1995), pages 60–75. DOI: 10.1007/BF01277956.

[5] A.-L. Barabási and R. Albert. "Emergence of Scaling in Random Networks". In: *Science* 286 (1999), pages 509–512. DOI: 10.1126/science.286.5439.509.

[6] P. Brach, M. Cygan, J. Łącki, and P. Sankowski. "Algorithmic Complexity of Power Law Networks". In: *27th Symp. Discrete Algorithms (SODA)*. 2016, pages 1306–1325. ISBN: 978-1-61197-433-1. DOI: 10.1137/1.9781611974331.ch91.

[7] M. Chlebík and J. Chlebíková. "Approximation hardness of dominating set problems in bounded degree graphs". In: *Inf. Comput.* 206.11 (2008), pages 1264–1275. DOI: 10.1016/j.ic.2008.07.003.

[8] F. Chung and L. Lu. "Connected Components in Random Graphs with Given Expected Degree Sequences". In: *Annals of Combinatorics* 6.2 (2002), pages 125–145. DOI: 10.1007/PL00012580.

[9] I. Dinur and S. Safra. "On the hardness of approximating minimum vertex cover". In: *Annals of Mathematics* 162 (2005), pages 439–485. DOI: 10.4007/annals.2005.162.439.

[10] D.-Z. Du, K.-I. Ko, and X. Hu. *Design and Analysis of Approximation Algorithms*. Springer, 2011. ISBN: 978-1-4614-1700-2. DOI: 10.1007/978-1-4614-1701-9.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. "On Power-law Relationships of the Internet Topology". In: *Symp. Communications Architectures and Protocols (SIGCOMM)*. 1999, pages 251–262. DOI: 10.1145/316194.316229.

[12] A. Ferrante, G. Pandurangan, and K. Park. "On the hardness of optimization in power-law graphs". In: *Theoret. Comput. Sci.* 393.1–3 (2008), pages 220–230. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2007.12.007.

[13] M. Gast, M. Hauptmann, and M. Karpinski. *Inapproximability of Dominating Set in Power Law Graphs*. 2012. arXiv: 1212.3517 [cs.CC].

[14] M. Hauptmann and M. Karpinski. *On the Approximability of Independent Set Problem on Power Law Graphs*. 2015. arXiv: 1503.02880 [cs.DS].

[15] M. Kao. *Encyclopedia of Algorithms*. Springer, 2008. ISBN: 978-0-387-30770-1. DOI: 10.1007/978-0-387-30162-4.

[16] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguná. "Hyperbolic geometry of complex networks". In: *Physical Review E* 82.3 (2010), page 036106. DOI: 10.1103/PhysRevE.82.036106.

[17] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. "Trawling the Web for Emerging Cyber-Communities". In: *Computer Networks* 31.11-16 (1999), pages 1481–1493. DOI: 10.1016/S1389-1286(99)00040-7.

[18] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. June 2014. URL: http://snap.stanford.edu/data (last accessed 2016-10-01).

[19] M. E. J. Newman. "Random graphs as models of networks". In: *Handbooks of Graphs and Networks*. Wiley-VCH, 2003, pages 35–68. DOI: 10.1002/3527602755.ch2.

[20] M. E. J. Newman. "The Structure and Function of Complex Networks". In: *SIAM Review* 45.2 (2003), pages 167–256. DOI: 10.1137/S003614450342480.

[21] A. G. Phadke and J. S. Thorp. *Computer Relaying for Power Systems*. John Wiley & Sons, Ltd, 2009. ISBN: 978-0-470-05713-1. DOI: 10.1002/9780470749722.

[22] L. Ruan, H. Du, X. Jia, W. Wu, Y. Li, and K.-I. Ko. "A greedy approximation for minimum connected dominating sets". In: *Theoret. Comput. Sci.* 329.1–3 (2004), pages 325–330. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2004.08.013.

[23] Y. Shen, D. T. Nguyen, Y. Xuan, and M. T. Thai. "New techniques for approximating optimal substructure problems in power-law graphs". In: *Theoret. Comput. Sci.* 447 (2012), pages 107–119. DOI: 10.1016/j.tcs.2011.10.023.

[24] F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi, and S. Shan. "On positive influence dominating sets in social networks". In: *Theoretical Computer Science* 412.3 (2011). Combinatorial Optimization and Applications, COCOA 2009, pages 265–269. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2009.10.001.

Towards the Interactive Rendering of Dynamic 3D Point Clouds

Sören Discher

Computer Graphics Systems Group
Hasso-Plattner-Institut
soeren.discher@hpi.de

Advances in 3D scanning technology allow for the creation of increasingly detailed, discrete, digital models of real-world surfaces in ever-shorter amounts of time. To explore, inspect, and interact with those 3D point clouds, specialized spatial data structures, level-of-detail-concepts, and client-server-based rendering techniques need to be combined. Typically, the involved data structures and level-of-detail-concepts are optimized for a specific point distribution, expecting the data set to remain static. Hence, they are not per se applicable to data sets that may change dynamically (e.g., based on user input). This report discusses the weaknesses of state-of-the-art point-based rendering approaches with respect to dynamic data sets and develops concepts for a real-time rendering system supporting static and dynamic 3D point clouds of any given size.

1 Introduction

Using today's in-situ and remote sensing technology, buildings, cities, or complete countries can be captured at minimal time and cost. The resulting 3D point clouds, i.e., discrete, digital representations of real-world surfaces, grant precise and up-to-date information about objects and structures within an area. Thus, they have become a fundamental data type for a variety of geospatial applications, facilitating, among others, urban planning and development [20], the documentation and preservation of cultural heritage [16], as well as homeland security [1]. Geospatial applications handling 3D point clouds are typically faced by an ever-increasing density (e.g., several hundred points per m^2) and capturing frequency (e.g., several times a year) while at the same time being limited by processing strategies that do not scale as well as limited storage capabilities. As a remedy, they frequently have to sacrifice precision and density of the data [12]. To overcome these limitations, out-of-core or external memory algorithms have to be applied. As an example, arbitrarily large 3D point clouds can be visualized by only rendering the most relevant points for the current view frustum and rendering technique [13]. Thus, an efficient access to subsets of the data based on different attributes – especially a point's spatial position – is required. This is usually achieved by combining spatial data structures and level-of-detail concepts. Combined with client-server-based rendering approaches these external memory algorithms are applicable to a multitude of platforms, ranging from high-end desktop computers and virtual reality systems to low-end mobile devices [19].

Figure 1: Raw 3D scans of cultural heritage may include *unwanted* objects that are not supposed to be part of the final model. Typically, those objects have to be identified and removed manually.

Real-time rendering systems for massive 3D point clouds typically apply several preprocessing steps that optimize the involved data structures for each data set individually:

- To minimize data consumption and data traffic, point based compression techniques may be applied.
- To ensure a highly-efficient selection of subsets of the 3D point cloud based on their spatial position, points have to be sorted into a spatial data structure that corresponds to the given point distribution.
- To support the selection of subsets based on additional per-point attributes (e.g., thematic data, temporal information), it may be necessary to combine several spatial data structures and to balance them accordingly (i.e., so-called multi-layered spatial data structures).

Usually, a 3D point cloud is expected to remain static after that initial preprocessing, as any subsequent changes to the data may necessitate a time-consuming re-balancing of the underlying spatial data structures. However, modifying the data might be unavoidable depending on the application scenario:

Streaming 3D Point Clouds 3D point clouds are traditionally created in a sequential process, i.e., the area in question first has to be captured completely, before the resulting raw data can be visualized and checked for potential errors (e.g., noise or insufficient point density). Such errors can be identified and fixed way more efficiently by *streaming* and visualizing points immediately upon their creation. However, that requires the underlying spatial data structures to support the constant integration of additional points in real-time.

Documenting Cultural Heritage Applying 3D scanning technology to document cultural heritage usually involves some manual, interactive editing of the raw point data: Some objects – albeit being scanned correctly – aren't supposed to be part of the actual model and thus have to be removed manually (Figure 1). At times, color values of surfaces can be inconsistent across consecutive scans (Figure 2). Harmonizing those color values correctly tends to require user input and thus needs to be conducted interactively.

Figure 2: Inaccuracies in the capturing process may lead to color values being inconsistent across consecutive scans. Harmonizing them often requires recompressing the corresponding subsets of the data.

With respect to those application scenarios, existing rendering concepts and techniques for static 3D point clouds need to be expanded to support (1) the interactive

addition and removal of points as well as (2) the interactive modification of per-point attributes (e.g., color values). If the point data is compressed, the latter requires the decompression and re-compression of arbitrary subsets of the data in real-time. In the following, state-of-the-art point based rendering approaches for static data sets are presented, before concepts and techniques are discussed that allow for the real-time rendering of arbitrary large dynamic 3D point clouds.

2 Background

A general overview of point-based rendering techniques is given by Gross and Pfister. Several techniques focus on the appearance of a 3D point cloud, typically aiming for a photorealistic and, thus, hole-free visualization [11, 18], which can be accomplished by applying an adequate size and orientation to each point. These attributes can be calculated in a preprocessing step [21] or on a per-frame basis [14]. While the former approach provides a better rendering performance, it's only applicable to static data sets. Meanwhile, non-photorealistic rendering techniques [5] highlight the inherent fuzzyness of a 3D point cloud. Thus, they are less dependent on a hole-free visualization and provide an acceptable image quality even for inaccurately sized and oriented points [22].

2.1 Out-of-Core-Rendering

To make point-based rendering techniques applicable to arbitrary large data sets, out-of-core or external memory algorithms have to be applied. Out-of-core rendering approaches for 3D point clouds [8, 15] use spatial data structures and level-of-detail concepts to aggregate or generalize points, usually based on spatial attributes. Depending on the application scenario as well as the overall structure and spatial distribution of the data, different spatial data structures may prove to be the most efficient:

Uniform Grid A uniform grid is the simplest form of a spatial data structure as it simply divides a space into a fixed number of equally size grid cells. Individual cells – or voxels in the case of a three-dimensional grid – are accessed via indices; level-of-detail concepts are not applied. Due to their simple characteristic, uniform grids can be expanded and regressed very efficiently. However, varying point densities are not taken into account, making that data structure inefficient for non-uniformly distributed data since a large number of cells will remain empty and unused.

Quadtree A quadtree is defined as a recursive subdivision of a *two-dimensional* area into equally sized *quadrants* until a minimal number of points for each cell is reached. That data structure is most efficient when the data is primarily distributed horizontally, which is typically true for aerial data, such as airborne laser scans. Points can be added to and removed from a quadtree with minimal

performance overhead as long as the points are located within the bounds defined by the tree's root node.

Octree In contrast to a quadtree, an octree is a recursive subdivision of a *three-dimensional* space into equally sized *octants* (Figure 3). Again, the recursion stops once a minimal number of points for given a cell is reached. Octrees are prefarable to quadtrees whenever the data is non-uniformly distributed both horizontally and vertically, as it's usually the case for terrestrial data (e.g., mobile mapping).

Kd-Tree As a generalization of quadtrees and octrees, kd-trees are binary trees, that organize points in a k-dimensional space. The splitting planes can be positioned freely alongside the respective coordinate axes, thus allowing to create perfectly balanced tree structures independently of the data's spatial distribution. While this minimized tree traversal times during rendering, kd-trees come with the trade-off of a more complex construction and update process compared to quadtrees or octrees. Thus, using them to organize dynamic data sets is not encouraged.

(Multi-)Layered Data Structures Depending on the use case, a combination of multiple spatial data structures might be favorable. For example, multiple terrestrial laser scans within a larger region may be organized most efficiently by combining separate octrees with an overlaying grid or quadtree. As another example, multi-layered data structures may be used to organize points based on multiple per-point attributes: Richter et al [15] demonstrate this by splitting semantically rich 3D point clouds based on each point's surface category (e.g., building, vegetation); for each surface category a separate kd-tree is maintained (Figure 4).

2.2 Client-Server-based Rendering

While being applicable to arbitrary large data sets, out-of-core rendering algorithms usually require a direct access to the data, which generally restricts their application to high-end desktop computers. To visualize massive 3D point clouds even on low-end mobile devices, such algorithms have to be combined with existing client-server-based approaches that limit workload and data traffic on client-side by using a central server infrastructure to maintain and distribute the data.

Rendering directly on the server and only transferring the rendered images to the client is commonly referred to as a thin client approach [3]. As an alternative, rendering can be delegated to the client-side (thick client approach). In that case, the server is responsible for selecting and configuring the data before transferring it to the client – ideally in a format optimized for rendering purposes [19]. While a thin client approach notably reduces the minimal hardware requirements on client side, a thick client approach is usually more feasible when massive amounts of clients have to be served due to the lower workload inflicted on server side. Furthermore, a

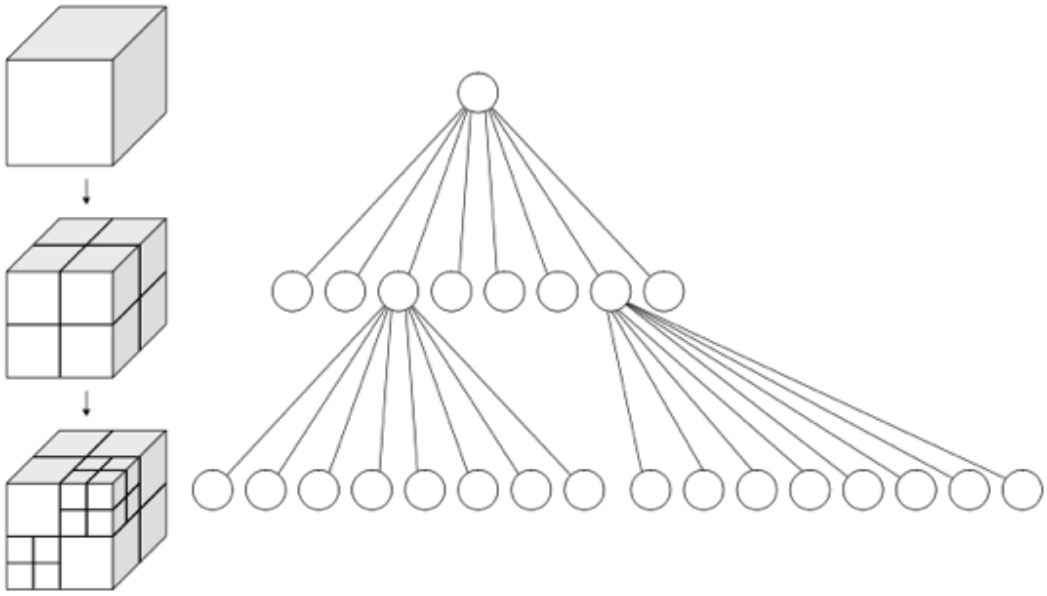


Figure 3: Octrees recursively subdivide a three-dimensional space into a number of equally sized octants. They are most efficient if the data is distributed non-uniformly along all three coordinate axes.

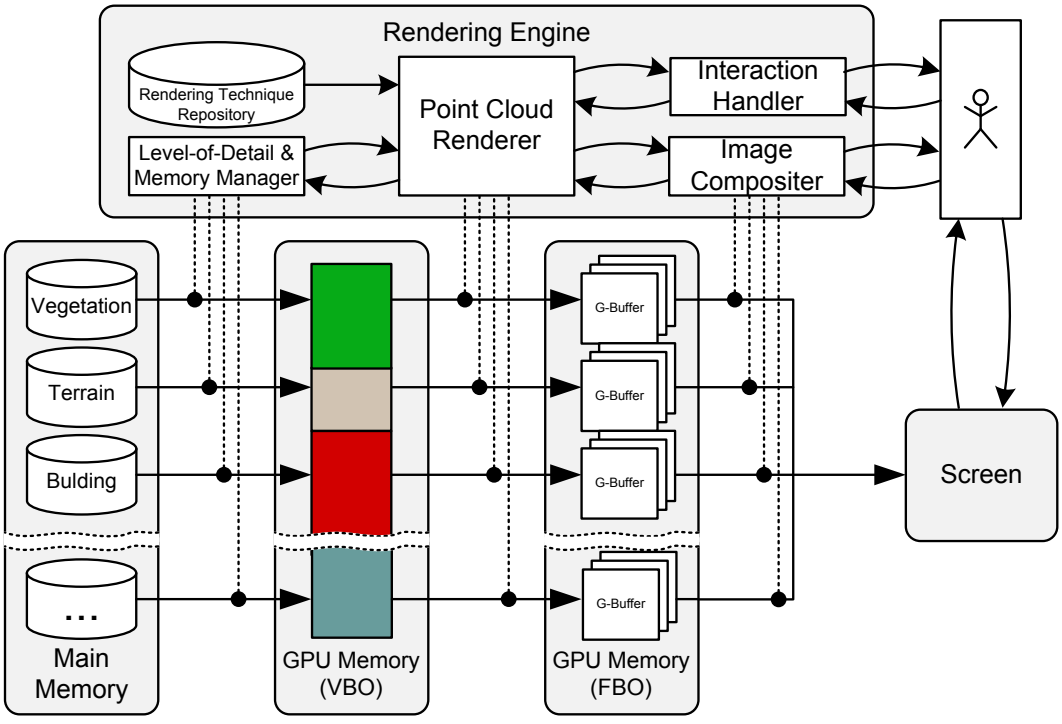


Figure 4: Example of a multi-layered rendering approach for massive, semantically rich 3D point clouds introduced by [15]. Depending on their surface category, points are sorted into a set of separately rendered spatial data structures.

thick client approach tends to be more resilient to poor network conditions since the visualization can still be adjusted to user interactions when connection to the server has been lost (albeit some relevant data might be missing).

When it comes to handling dynamic data sets, differences between both approaches are minimal: Constantly streaming newly captured points may require changes to the underlying spatial data structures (see section 2.1) but does affect the amount of data that has to be transferred between server and clients only slightly (thick client) or not all (thin client). On the other hand, interactively editing point clouds in a client-server-based environment is a lot more complex to implement, as it requires to reliably synchronize user-made changes to the data across several clients. While that challenge will not be further discussed in this report, it should be considered in future work.

2.3 Point Cloud Compression

To minimize overall data consumption and to speed up data transmission, 3D point clouds need to be compressed. State-of-the-art approaches provide good compression ratios at little [9] to none [10] information loss. A popular strategy is the use of height-map based encoding over planar [2] or non-planar domains (e.g., spheres or cones) [7] to encode point positions; another group of authors proposes to utilize spatial data structures [4, 17]. However, when handling dynamic 3D point clouds, arbitrary large subsets of the data need to be compressed and decompressed in real-time, which is not supported by most compression techniques. One approach, that is real-time capable, while also being applicable locally and incrementally, was recently introduced by Golla and Klein [6].

3 Interactive Rendering of Dynamic 3D Point Clouds

Visualizing massive 3D point clouds requires organizing points in a way that facilitates the selection of the most relevant subsets of the data for a given view. This can be achieved by applying specialized spatial data structures and level-of-detail concepts. However, those are typically focussed on static data sets. Integrating, removing, and repositioning points might necessitate a time consuming and inefficient rebalancing of the corresponding spatial data structure. Thus, state-of-the-art rendering concepts and techniques for static point clouds are not directly applicable to dynamic data sets but need to be modified first. Generally, those modifications follow one of the strategies listed below.

3.1 Update-optimized data structures

As opposed to spatial data structures that are heavily optimized towards a specific point distribution (e.g., kd-trees), one strategy relies on using less optimized spatial data structures that in return can be modified more efficiently, ideally in real time. In

its simplest form, that could refer to a fixed sized queue of recently captured points that is regularly cleared in first-in-first-out order, thus, allowing for the real-time visualization of point cloud streams, albeit more in the sense of an automatically updated video with minimal user-based interaction. Another example for such a spatial data structure is a uniform grid, that allows to efficiently integrate, remove, or reposition points during rendering. As discussed in section 2.1, this comes at the cost of a significantly higher memory consumption as well as a notably worse tree traversal time and, thus, rendering performance. Depending on the use case however, those negative aspects can be minimized by artificially restricting the maximum point density applied during rendering since the full resolution might not be required.

3.2 Hybrid Rendering

A more flexible strategy is a so-called hybrid rendering approach that combines highly optimized spatial data structures for static point clouds with the strategy discussed above (Figure 5): Newly captured points are not immediately integrated into the primary spatial data structure (e.g., a kd-tree) but rather added to a temporary, update-optimized data structure (e.g., a queue or uniform grid) of fixed size that is stored in main memory. While points managed by the temporary data structure are tested against the current view frustum, no level-of-detail calculations are necessary, thus, speeding up the selection of points to render. When the temporary data structure reaches its maximum capacity, it is cleared and the corresponding points are transferred to secondary storage. In regular intervals, the primary spatial data structure is updated to incorporate points that have been recently removed from the temporary data structure.

The frequency in which these updates are conducted, can be adjusted dynamically with respect to the overall performance and workload of the rendering system, thus improving the scalability and efficiency of the proposed strategy. Such a hybrid approach also allows for the interactive removal of points by (1) labeling the respective points accordingly to exclude them from being rendered and (2) removing them from the spatial data structure during the asynchronous update process. Finally, points can be repositioned by removing and immediately re-adding them.

4 Conclusion and Outlook

State-of-the-art real-time rendering systems for 3D point clouds are typically designed with static data sets in mind: They achieve high frame rates and minimal memory consumption even for massive data sets due to being highly optimized towards a specific point distribution and data characteristic. However, even small changes to the data — such as integrating, removing, and repositioning points or modifying per-point attributes — initiate a time-consuming update process during which an interactive visualization of the data is not possible. As discussed in this report, real-time rendering concepts and techniques that are applicable to both static and dynamic data sets, require a tradeoff between an optimal rendering performance

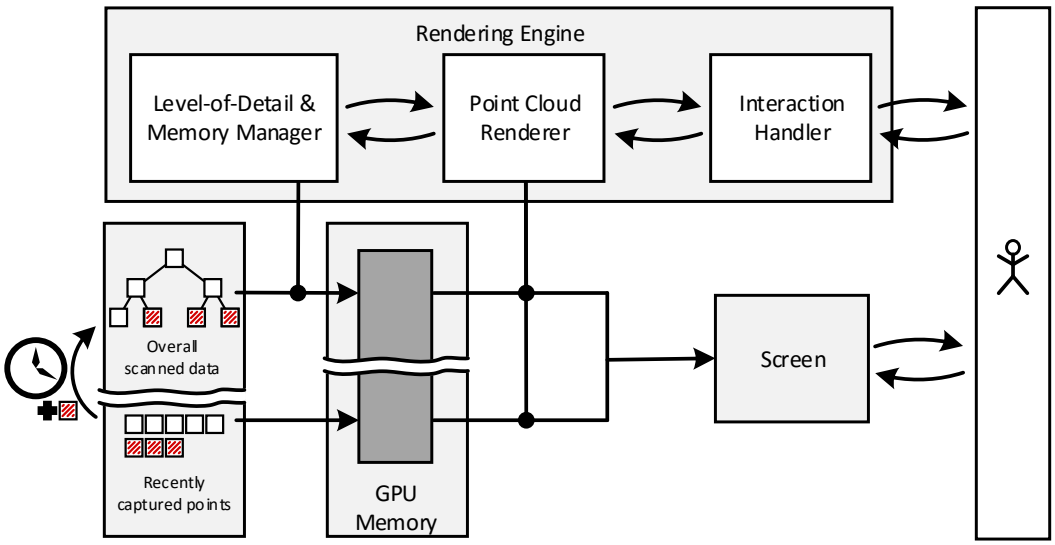


Figure 5: Overview of a hybrid rendering approach: Recently scanned points are stored in a fixed sized, update-optimized data structure. In regular intervals points are transferred into the primary spatial data structure enabling level-of-detail and external memory based rendering.

and memory efficiency on the one side and a highly efficient update process on the other side. For instance, alternative spatial data structures that can be updated more efficiently come at the cost of an increased traversal time. Alternatively, smaller changes to the data can be aggregated before being applied in bulk in regular intervals; however, that requires the introduction of additional data structures to cache incoming changes.

Regarding data compression, existing approaches for static data sets are only applicable if they allow for the local compression and decompression of arbitrary parts of the data in real-time. As for visualizing dynamic 3D point clouds in client-server-based architectures, additional challenges arise if a collaborative editing of the data should be supported, as that requires the synchronization of editing operations across all involved clients.

5 Acknowledgements

This work was funded by the Federal Ministry of Education and Research (BMBF), Germany within the InnoProfile Transfer research group “4DnD-Vis”. I would like to thank the Zamani Project group within the Geomatics Division of the University of Cape Town, virtualcitySYSTEMS, and the Faculty of Architecture at the Cologne University of Applied Sciences for providing data sets.

References

[1] J. Coutinho-Rodrigues, A. Simão, and C. H. Antunes. "A GIS-based multicriteria spatial decision support system for planning urban infrastructures". In: *Decision Support Systems* 51.3 (2011), pages 720–726. ISSN: 0167-9236.

[2] J. Digne, R. Chaine, and S. Valette. "Self-similarity for accurate compression of point sampled surfaces". In: *Computer Graphics Forum*. Volume 33. 2. 2014, pages 155–164.

[3] J. Döllner, B. Hagedorn, and J. Klimke. "Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps". In: *17th International Conference on 3D Web Technology*. 2012, pages 97–100. ISBN: 978-1-4503-1432-9.

[4] Y. Fan, Y. Huang, and J. Peng. "Point cloud compression based on hierarchical point clustering". In: *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*. 2013, pages 1–7.

[5] M. Goesele, J. Ackermann, S. Fuhrmann, C. Haubold, R. Klowsky, D. Steedly, and R. Szeliski. "Ambient point clouds for view interpolation". In: *ACM Transactions on Graphics* 29.4 (2010), 95:1–95:6. ISSN: 0730-0301.

[6] T. Golla and R. Klein. "Real-time point cloud compression". In: *International Conference on Intelligent Robots and Systems*. 2015, pages 5087–5092.

[7] T. Golla, C. Schwartz, and R. Klein. "Towards efficient online compression of incrementally acquired point clouds". In: *Vision, Modeling & Visualization*. 2014, pages 17–22. ISBN: 978-3-905674-74-3.

[8] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti. "An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees". In: *The Visual Computer* 29.1 (2013), pages 69–83. ISSN: 1432-2315.

[9] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. "The quantized kd-tree: Efficient ray tracing of compressed point clouds". In: *2006 IEEE Symposium on Interactive Ray Tracing*. 2006, pages 105–113.

[10] M. Isenburg. "LASzip: lossless compression of LiDAR data". In: *Photogrammetric Engineering & Remote Sensing* 79.2 (2013), pages 209–217.

[11] H.-J. Kim, A. Cengiz Öztireli, M. Gross, and S.-M. Choi. "Adaptive surface splatting for facial rendering". In: *Computer Animation and Virtual Worlds* 23.3-4 (2012), pages 363–373.

[12] F. Lafarge and C. Mallet. "Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation". In: *International journal of computer vision* 99.1 (2012), pages 69–85. ISSN: 1573-1405.

[13] S. Nebiker, S. Bleisch, and M. Christen. "Rich point clouds in virtual globes – A new paradigm in city modeling?" In: *Computers, Environment and Urban Systems* 34.6 (2010), pages 508–517. ISSN: 0198-9715.

[14] R. Preiner, S. Jeschke, and M. Wimmer. "Auto Splats: Dynamic point cloud visualization on the GPU". In: *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*. 2012, pages 139–148. ISBN: 978-3-905674-35-4.

[15] R. Richter, S. Discher, and J. Döllner. "Out-of-Core visualization of classified 3D point clouds". In: *3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014*. 2015, pages 227–242.

[16] H. Rüther, C. Held, R. Bhurtha, R. Schroeder, and S. Wessels. "From point cloud to textured model, the zamani laser scanning pipeline in heritage documentation". In: *South African Journal of Geomatics* 1.1 (2012), pages 44–59.

[17] R. Schnabel and R. Klein. "Octree-based Point-Cloud Compression." In: *SPBG* 6 (2006), pages 111–120.

[18] M. Schütz and M. Wimmer. "High-quality point-based rendering using fast single-pass interpolation". In: *2015 Digital Heritage*. Volume 1. 2015, pages 369–372.

[19] M. Schütz and M. Wimmer. "Rendering large point clouds in web browsers". In: *Proceedings of CESCG* (2015), pages 83–90.

[20] I. Tomljenovic, B. Höfle, D. Tiede, and T. Blaschke. "Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation". In: *Remote Sensing* 7.4 (2015), pages 3826–3862. ISSN: 1573-1405.

[21] J. Wu and L. Kobbelt. "Optimized sub-sampling of point sets for surface splatting". In: *Computer Graphics Forum* 23.3 (2004), pages 643–652.

[22] H. Xu, M. X. Nguyen, X. Yuan, and B. Chen. "Interactive silhouette rendering for point-based models". In: *Proceedings of the First Eurographics conference on Point-Based Graphics*. 2004, pages 13–18. ISBN: 3-905673-09-6.

Coverage Considerations for Software Fault Injection

Lena Feinbube

Operating Systems and Middleware
Hasso-Plattner-Institut
lena.feinbube@hpi.uni-potsdam.de

Software fault injectors insert failure causes into a running program, to test its fault tolerance. The fault injection experiments should be representative of real-world faults and cover the space of potential failure causes adequately. Coverage criteria can assess the quality of a set of fault injection experiments, and direct the testing process. This article discusses how to assess the coverage of software fault injection testing from a theoretical perspective, and at the example of a highly available OpenStack.

1 Introduction

Software fault injection (SFI) is an acknowledged approach to experimentally assess the dependability of software systems. SFI tools generally insert potential failure causes – such as software and hardware faults, and error states – into a program during runtime. The resulting behavior can be observed to either evaluate the effectiveness of fault tolerance mechanisms, or to measure the severity of the failing outcomes. As a dynamic approach, SFI offers several advantages over static verification tools: It can be implemented in a scalable fashion. SFI works also for arbitrarily complex programs, where static methods fail due to state space explosion. The input space of possible failure causes can be explored incrementally.

Nevertheless, SFI, due to its experimental nature, arguably lacks the absolute guarantees of formal methods. While formal methods promise exhaustiveness by checking the entire state space and an entire fault model, SFI is rather used as an ad hoc tool. Therefore, the quality of a set of SFI experiments needs to be examined critically. Research questions include: What implications regarding the dependability do the outcomes of SFI experiments have? Can these implications be quantified as dependability metrics? How can we measure and compare the effectiveness and efficiency of different SFI tools? To what extent does a set of tests and fault injection experiments cover the space of potential behaviors?

In this article, we will discuss existing coverage criteria for software, how the notion of coverage can be extended to fit SFI, and implications for the development of new fault injectors.

1.1 Why Coverage Measures?

When testing software, we need to "ensure that prior to the start of testing the objectives of testing are known and agreed upon and that the objectives are set in terms that can be measured." Test objectives should be "quantified, reasonable, and

achievable". [18] When testing software, also by means of fault injection, there is an inherent trade-off between increased confidence in the correctness of the system, and cost in terms of time and effort. [13] For instance, when hardware faults are injected, not all faults have the potential of actually disturbing the system. Faults injected into unused memory regions or registers which are about to be overwritten, for example, will probably not have an effect on the program's behavior. Several methods for reducing the fault space, while maintaining similar levels of significance, have been proposed. They based either on *pre-injection analysis* or *post-injection analysis*. [1]

A coverage measure, also referred to as coverage metric or "test adequacy criterion" [6], is a measure of quality for software testing which can be used to quantify the mentioned confidence-effort trade-off. It describes how well a set of experiments, a test suite, or a set of fault injection runs, covers the space of possible behaviors.

Generally, higher coverage means that greater confidence can be placed on the test results. If a set of experiments with high coverage indicates program correctness, that is probably the case. To improve coverage, the tests can be improved or tests for yet uncovered program parts can be added.

> Our goal, then, should be to provide enough testing to ensure that the probability of failure due to hibernating bugs is low enough to accept. "Enough" implies judgement. [8]

Coverage measures are important in risk management, because they provide the testers with an empirical assessment of the adequacy of their verification activities. They can also play a central role in process assurance and provide exit criteria for when sufficient testing has been done.

2 Related Work

A broad variety of coverage measures have been discussed in different contexts of the software development process. On a low level of abstraction, well-established coverage criteria for the injection of hardware faults (implemented in software – software-implemented (hardware) fault injection (SWIFI) – or hardware) exist. On the software level, the term "coverage" usually relates the coverage of a program by (unit) test cases. It is measured for instance as code, model, or feature coverage.

2.1 Test Coverage

Software testing literature has defined a broad spectrum of test coverage metrics. These metrics quantify to which extent a test suite "covers" the program under consideration. Some coverage criteria include:

- **Function Coverage** requires that each function is called at least once during testing. Analogously, module or source code line coverage can be defined.
- **Condition Coverage** requires that each condition, i.e., each leaf-level boolean expression, must evaluate to both true and false. **Decision Coverage** requires

that each decision, or branch, in the program has taken all outcomes at least once.

- A stricter combination of condition and decision coverage, **Modified Condition/Decision Coverage (MC/DC)** requires that each condition be shown to *independently affect the outcome of the decision*. For a statement such as `if (A || B) { C = true; } else { C = false; }`, at least three assignments of A and B need to be tested: (true, false), (false, true) – which will make C evaluate to true, and (false, false) – which will make C evaluate to false. Since MC/DC requires a large amount of test cases, it is assured mainly in safety-critical scenarios, and required by aviation standard DO-178C [14] and automotive standard ISO 26262 [15]. [1]

Function, module and source code line coverage can be seen as *structural coverage criteria* – they are measured with regards to the static structure of the program and are not susceptible to state space explosion. The number of necessary test cases grows linearly with the program size. On the other hand, condition and decision coverage are here denoted as *state-based coverage criteria*.

Code-level coverage criteria are optimized by test case selection and generation techniques such as *symbolic execution* [12] and *concolic execution* [17].

2.2 Fault and Failure Space Coverage

The term "fault coverage" has been used both as a *dependability measure*, and as a *measure of the quality of dependability assessment*.

For a number F failed experiments out of a total of N experiments, the *fault coverage factor* is estimated as follows: [4]

$$c_{est} = 1 - \frac{F}{N} \approx 1 - P(Failure|Fault) = P(Recovery) \tag{1}$$

Intuitively, it describes the probability of an injected fault resulting in an externally observable failure. [16] The converse probability is the probability of a recovery from the error state before a failure occurs.

Due to technical as well as resource limitations, the number N can be much lower than the entire space of conceivable injection points. Therefore, fault coverage is merely a statistical estimate of the actual recovery probability. To derive the overall system reliability from this estimate, the probability of fault existence would also need to be known. This is rarely the case, especially in software. [2]

However, the fault coverage factor is not primarily a measure of fault injection quality, but rather of the reliability of the system under test.

[1] MC/DC is among the strictest coverage criteria actually measured in real-world applications. Even stricter criteria, such as *Multiple Condition Coverage* are conceivable: this criterion requires all combinations of conditions within a decision point to be tested, resulting in 2^n test cases for decisions with n variables.

Hudak et al. [10] present a study of hardware and software fault injection, where coverage was evaluated using three empirically determined probabilities: The probability of *detection* – $Pr\{detecting\ an\ error \mid a\ fault\ is\ active\}$, the probability of *recovery* – $Pr\{recovering\ successfully \mid an\ error\ is\ detected\}$, and the probability of *aborting* – $Pr\{aborting \mid successful\ recovery\ is\ not\ possible\}$.

Ghosh et al. [7] present an assessment of the fault injection testing they implemented in a distributed, component-based system.

Their coverage metric is based on the two-dimensional classification of [5], which views fault and code coverage as two orthogonal dimensions.

Fault injection has been motivated as a means for increasing test coverage [3].

2.3 Research Gap

Various coverage measurement, test case selection and assessment tools exist for both low-level hardware fault models, and success-space software testing. But what about failure-space software fault injection testing, based on software fault models? The research on *coverage criteria for the injection of software faults* remains sparse. Here, we explore potential approaches to such coverage criteria.

3 Coverage Criteria for SFI: Theory

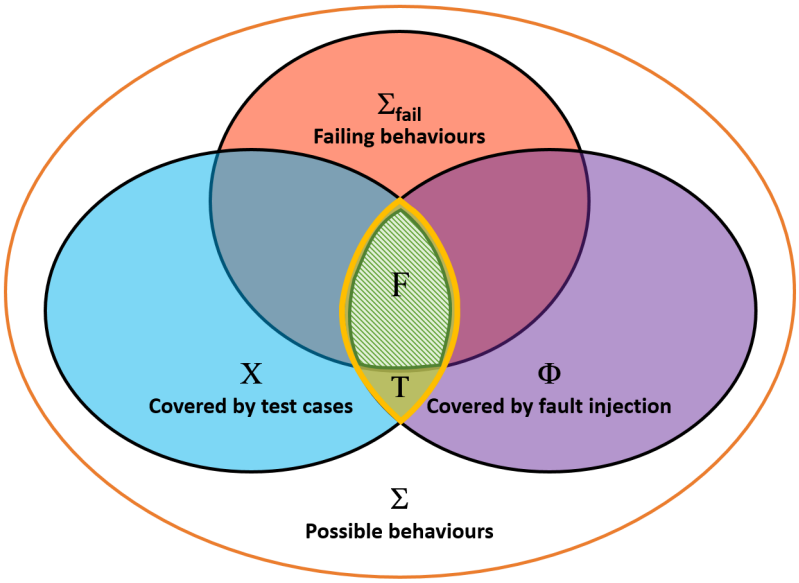


Figure 1: Coverage in relation to the overall space of possible behaviors. A behavior in this context is a state trace leading from an input to a final program output. This definition can be extended to include timing, which plays a role in several failure classes.

Coverage metrics, especially in the context of software, are usually controversial and can only approximate the actual coverage: ideally, the metric would reflect the proportionate coverage of the state space of the program – which means that any perfect coverage computation would soon have to surrender to the state space explosion problem.

Figure 1 visualizes the space of possible system behaviors with regard to theoretical coverage metrics. In this abstraction,

- $\frac{\Sigma_{fail}}{\Sigma}$ is correlated with the failure rate. It does not approximate or accurately describe the failure rate, unless a uniform distribution of behavior probability can be assumed, which is unrealistic.
- *Test case coverage* $\approx \frac{X}{\Sigma}$
- *Fault injection coverage* $\approx \frac{\Phi}{\Sigma}$
- T denotes the behaviors which are covered by both the code coverage and the fault coverage metric.
- F denotes the set of behaviors which we try to maximize by applying testing and fault injection with high coverage.

Actually computing any of the depicted metrics is close to impossible – for Σ_{fail} and Σ, it would require enumerating the entire state space of the program, which is usually hindered due to complexity or non-determinism; for X and Φ we have already discussed that coverage metrics are usually approximate numbers; finally, for F and T, the overlap between the covered behaviors needs to be determined. This overlap describes to what degree tests and fault injection experiments cover the same cases. If the fault injection only takes place while running the test suite, then $\Phi \subset X$. Even in this case, a program run may fail simply because it does not fulfil the test requirements, or because fault injection also took place. In the more general case, the overlap and thus the target metric F are even harder to compute theoretically.

4 Case Study: Highly Available OpenStack

Here, we will discuss the proposed coverage metric at the example of a highly available OpenStack setup. OpenStack[2] is often referred to as the "operating system of the cloud", and constitutes an open source implementation of a cloud management stack, broken down into different services. Prior and ongoing research by the author addresses potential fault injection points into the OpenStack cloud management stack. [9]

[2] https://www.openstack.org/ (last accessed 2016-10-20).

Figure 2 depicts the exemplary OpenStack setup we will use in this case study. It has been described by Brian Seltzer in his blog[3] and has been chosen here as it incorporates representative, state of the art fault tolerance features, while maintaining simplicity.

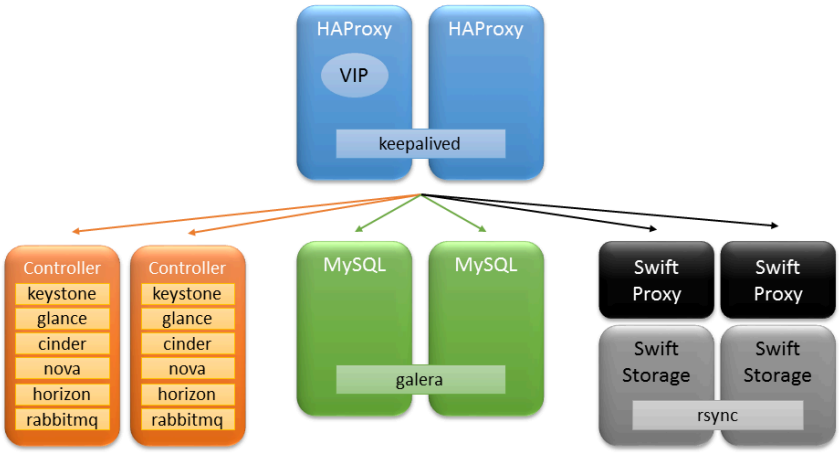


Figure 2: Example OpenStack architecture.

OpenStack is exemplary of real-world systems in the sense that it consists of diverse software modules which are based on individual codebases and have different dependability traits. Each codebase might have its own testing strategy and test coverage, so the overall test case coverage discussed in the previous section is already non-trivial to determine.

Here, we focus on the fault injection coverage, i.e., on the failure space. Based on a typical implicit fault model, where physical nodes may fail, network links can be unreliable, and computation can become slow due to CPU load, we assume the following fault injection experiments are run:

1. A script kills each physical host once (8 experiments);
2. Within each single virtual machine, the CPU quota are artificially throttled (8 experiments);
3. The management network is artificially partitioned between the controller nodes (1 experiment).

The first observation is that this fault model is composed of different layers at which dependability can be threatened. Under some assumptions, mappings between the layers might be possible, for example, a network partition might be equivalent to

[3] http://behindtheracks.com/2014/04/openstack-high-availability-controller-stack/ (last accessed 2016-10-20).

the outage of one isolated controller node – but this is speculative, as the fault tolerance against network partitions requires sophisticated detection mechanisms as well as proper isolation and hence needs to be implemented differently from crash fault tolerance. Therefore, coverage needs to be considered at each layer. This is somewhat different from software test coverage criteria, which are computed at the source code layer only, abstracting away the execution environment.

In the following paragraphs, we will discuss how the overall amount of necessary fault injection experiments can be computed and how fault injection can be guided by coverage considerations.

State-based coverage computations To evaluate coverage with regards to state, the overall set of possible combinations of faults needs to be considered. For example, with N nodes this would naïvely correspond to the size of the power set, 2^N. Even under the one-fault-at-the time assumption, multiple components can move from the operational to the failed state one after another. Therefore, the power set computation is realistic only if the fault distributions are assumed to be memoryless, and so are the fault tolerance mechanisms: In this case only the ordering of faults does not matter. Otherwise, for each member of the power set, each possible sequence of faults occurring would need to be considered – the combinatorial explosion here may be hard to handle.

The above mentioned experiments obviously do not cover the entire state space, because they do not include multiple faults occurring at the same time.

Structural coverage computations In order to evaluate the coverage of fault injection in a structural manner, an understanding of the logical dependability structure of the system is needed. A structural dependability modelling language can visualize this understanding. For example, a reliability block diagram (RBD) [11] of the system could look like Figure 3. In this simple case, each OpenStack service needs to work for the overall system to fulfil its specification, but each service is internally dually redundant.

In RBDs, as long as a path from the start to the end exists, the system can be assumed operational. Therefore, one coverage criterion could be that *each independent path through the RBD should be interrupted by fault injection*. Interrupting each parallel structure at each possible place covers each path. In our case, this means that $4 * 2 = 8$ experiments are necessary. The experiments sets mentioned above at the node and virtual machine layers satisfy this requirement.

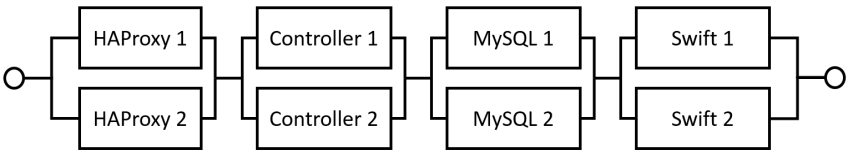


Figure 3: Reliability block diagram (RBD) of the example OpenStack setup. As long as one path from start to end exists, the system is assumed operational.

Another, stricter criterion could be that *for each independent path through the RBD, all other paths but this one should be interrupted by fault injection*. In our case, there are $2^4 = 16$ paths to success through the RBD, which corresponds to the minimal number of fault injection experiments.

The structural considerations outlined here work well for the first two assumed sets of experiments, but a different structural model would be required for the third experiment, which includes network partitions. Here, structural coverage would be based on the detailed network topology, and the available redundant paths in it.

5 Discussion

We have discussed the estimation of coverage criteria for SFI from a theoretical point of view and at the example of a highly available OpenStack system. The main conclusion is that while quantification of coverage has many benefits, it is challenging to compute and can only approximate the truth, even more so for complex multi-layered systems whose failure behavior is not understood.

Fault models need to be defined at each layer of the software stack, and the coverage of fault injection should be evaluated in a likewise differentiated manner. Because fault tolerance is implemented in different ways, it needs to be exercised with fault injection at each layer. For example, crash failures at the node level may be survivable using a simple active-passive failover, network partitions can be tolerated with quorum devices, virtual machine failures require detection and handling in the VMM. In our considerations, we have only discussed software layers and so far neglected the hardware, which again has different fault models (e.g. single or multiple bit flips in various locations) and tolerance mechanisms (e.g. error correcting codes).

The OpenStack example has shown that structural considerations are more realistic than state-based ones. Dependability modelling can yield a structure of the system based on which the SFI experiments can be evaluated.

References

[1] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson. "Assembly-level pre-injection analysis for improving fault injection efficiency". In: *European Dependable Computing Conference*. Springer. 2005, pages 246–262.

[2] B. Beizer. "Software is different". In: *Annals of Software Engineering* 10.1 (2000), pages 293–310.

[3] J. M. Bieman, D. Dreilinger, and L. Lin. "Using fault injection to increase software test coverage". In: *Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on*. IEEE. 1996, pages 166–174.

[4] W. Bouricius, W. C. Carter, and P. Schneider. "Reliability modeling techniques for self-repairing computer systems". In: *Proceedings of the 1969 24th national conference*. ACM. 1969, pages 295–309.

[5] R. A. DeMillo, T. Li, and A. P. Mathur. "A Two Dimensional Scheme to Evaluate the Adequacy of Fault Tolerance Testing". In: *Third Int'l Workshop on Integrating Error Models with Fault Injection*. IEEE. 1994, pages 54–56.

[6] W. Du and A. P. Mathur. "Vulnerability testing of software system using fault injection". In: *Purdue University, West Lafayette, Indiana, Technique Report COAST TR* (1998), pages 98–02.

[7] S. Ghosh, A. P. Mathur, J. R. Horgan, J. J. Li, and W. E. Wong. "Software Fault Injection Testing on a Distributed System – A Case Study". In: *Proc. of the 1st International Quality Week Europe, Brussels, Belgium* (1997).

[8] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. *A practical tutorial on modified condition/decision coverage*. 2001.

[9] L. Herscheid, D. Richter, and A. Polze. "Experimental Assessment of Cloud Software Dependability Using Fault Injection". In: *Doctoral Conference on Computing, Electrical and Industrial Systems*. Springer. 2015, pages 121–128.

[10] J. Hudak, B.-H. Suh, D. Siewiorek, and Z. Segall. "Evaluation and comparison of fault-tolerant software techniques". In: *IEEE Transactions on Reliability* 42.2 (1993), pages 190–204.

[11] M. C. Kim. "Reliability block diagram with general gates and its application to system reliability analysis". In: *Annals of Nuclear Energy* 38.11 (2011), pages 2456–2461.

[12] J. C. King. "Symbolic execution and program testing". In: *Communications of the ACM* 19.7 (1976), pages 385–394.

[13] A. Petrenko, G. Bochmann, and M. Yao. "On fault coverage of tests for finite state specifications". In: *Computer Networks and ISDN Systems* 29.1 (1996), pages 81–106. ISSN: 0169-7552. DOI: http://dx.doi.org/10.1016/S0169-7552(96)00019-0.

[14] Radio Technical Commission for Aeronautics (RTCA). *Software Considerations in Airborne Systems and Equipment Certification*. Standard DO-178C. European Organisation for Civil Aviation Equipment (EUROCAE), 2012.

[15] *Road vehicles — Functional safety*. Standard ISO 26262. International Organisation for Standardization, 2011.

[16] H. Schirmeier. *Efficient Fault-Injection-based Assessment of Software-Implemented Hardware Fault Tolerance (Doctoral dissertation)*. 2016.

[17] K. Sen. "Concolic testing". In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM. 2007, pages 571–572.

[18] H. Zhu, P. A. Hall, and J. H. May. "Software unit test coverage and adequacy". In: *ACM computing surveys (csur)* 29.4 (1997), pages 366–427.

Improving Self-Healing by Estimating the Impact of Adaptation Rules on the Utility at Runtime

Sona Ghahremani

System Analysis and Modeling Group
Hasso-Plattner-Institut
sona.ghahremani@hpi.uni-potsdam.de

Rule-based self-adaptation approaches prescribe the adaptation to be executed if the system or environment satisfy certain conditions. In contrast, utility-driven approaches often enable optimal decisions by searching the possible space of adaptations to find the optimal one. We propose a combination of rule-based and utility-driven approaches to achieve the beneficial properties of each of them. With our combined approach we target the architecture-based self-healing of software systems.

1 Overview

There are various ways how self-adaptation can be realized. On the one hand *rule-based* approaches [5, 9] prescribe the adaptation to be executed for specific events and under specific conditions by adaptation rules and usually result in a scalable solution, however, often only with at most sufficient adaptation decisions. On the other hand *utility-driven* approaches [4, 11] often enable optimal decisions.

Therefore, we propose to combine rule-based and utility-driven approaches to achieve the beneficial properties of each of them in such a manner that the drawbacks can be avoided targeting in particular the architecture-based self-healing of software systems. Defining the utilities as well as the adaptation rules in a pattern-based way at the architectural level allows us to combine both in an incremental scheme to estimate the impact of each rule application on the overall utility and its cost at runtime. Based on these estimations, we select at runtime the most promising sequence of rule applications for execution at runtime. The approach enables a scalable solution and further results in good adaptation decisions with respect to the resulting utility if certain restrictions apply.

The concrete benefits of the approach are that the improvement concerning the utility is maximized in case alternative repair rules have a different impact for a concrete repair needs, that the cost for reestablishing the utility is minimized in case alternative repair rules have the same impact on utility but different costs for a concrete repair need, and that the loss of utility over time (lost reward) for reestablishing the utility is minimized in case of multiple concrete repair needs. We demonstrate these benefits of the approach in a comparative study with a static approach which does not compute the adaptation impact on the utility at runtime but uses static estimates.

2 Architectural Self-Adaptation and Runtime Models

To realize self-adaptation, a software system is equipped with a feedback loop that monitors and analyzes the system and if needed, plans and executes adaptation to the system. For this purpose, the feedback loop uses knowledge (cf. *MAPE-K* [10]). To achieve architectural self-adaptation, the feedback loop maintains a *runtime model* [2], as part of its knowledge, which represents the architecture of the system under adaptation. As the running example for the system under adaptation, we use *mRUBiS* [12], a component-based system realizing an online marketplace that hosts an arbitrary number of shops. Each shop consists of 18 components and runs isolated from the other shops on the marketplace.

Therefore, we equip mRUBiS with a MAPE-K feedback loop that uses a runtime model representing the mRUBiS architecture. This model conforms to the metamodel shown in Figure 1.

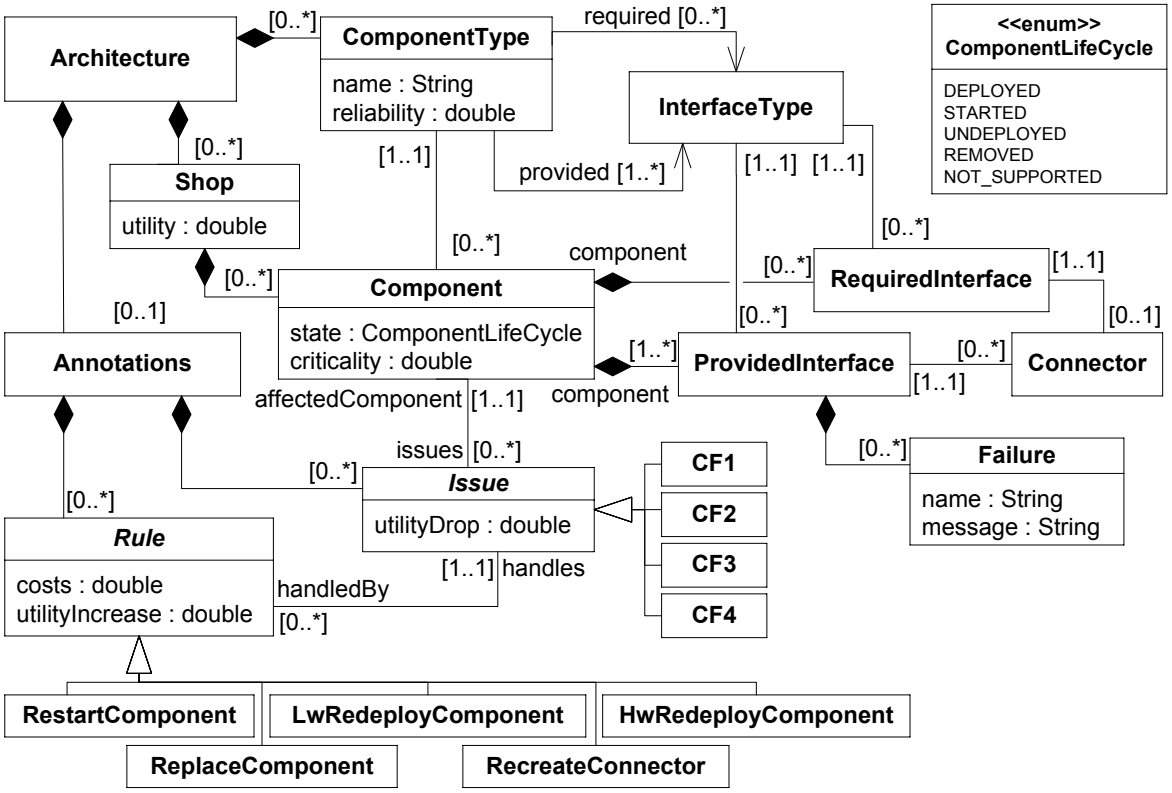


Figure 1: Simplified Metamodel of the Runtime Model.

3 Pattern-Based Architectural Utility

Desirable or undesirable issues for a dynamic architecture can be expressed as positive respectively negative model *patterns* such that concrete issues relate to occurrences of these patterns in a runtime model G. We denote that an occurrence as a match m for a pattern P in the runtime model G exists by $G \models_m P$. Therefore, we propose defining a utility function for a dynamic architecture represented in a runtime model with patterns.

For any utility function for a dynamic architecture must hold that (i) the optimal architectural configuration where all the system goals are optimally fulfilled must gain the maximum utility and that (ii) if any constraint or goal is violated, it must lead to a decrease of the utility. Thus, we employ *positive architectural utility patterns* $\text{Pa}^+ = \{P_1^+, \ldots, P_k^+\}$ and capture their impact on the utility accordingly using utility sub-functions U_i^+ to address case (i). Similarly, we employ *negative architectural utility patterns* $\text{Pa}^- = \{P_1^-, \ldots, P_n^-\}$ and capture their impact on the utility accordingly using utility sub-functions U_j^- to address case (ii). It has to be noted that the impact of each pattern occurrence on the overall utility, which is captured by a utility sub-function, may vary for each occurrence depending on the specific characteristics of the context (e.g., for self-healing, the attributes of a failing component may indicate the criticality of this component or the severity of the observed failures, which determines the negative impact on the utility).

As an example, we use *mRUBiS* [12], a component-based system realizing an online marketplace that hosts an arbitrary number of shops. Each shop consists of 18 components and runs isolated from the other shops. We are particularly interested into self-healing, that is, the automatic repairing of runtime failures by architectural self-adaptation. Therefore, we equip mRUBiS with a MAPE-K loop that uses a runtime model representing the mRUBiS architecture. The model captures the hosted Shops, their structuring into Components (with ProvidedInterfaces and RequiredInterfaces) and Connectors linking required and provided interfaces, as well as runtime Failures. Thereby, each Component is of a specific ComponentType.

For the example, Figure 2 shows the positive architectural utility pattern P_1^+ with its utility sub-function U_1^+. For each started (i.e., running) component of a shop, the utility of the shop is increased by U_1^+. We may define U_1^+ as the product of the criticality of the specific component, the reliability of the component type, and the connectivity of the component (i.e., the number of associated connectors). This pattern is applied to all shops on the marketplace to obtain the total positive impact on the overall utility of the marketplace.

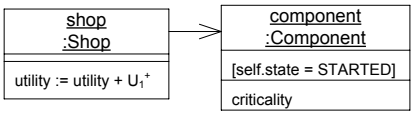


Figure 2: Positive Architectural Utility Pattern P_1^+.

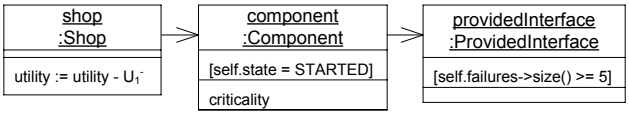


Figure 3: Negative Architectural Utility Pattern P_1^-.

Similarly, Figure 3 presents the negative architectural utility pattern P_1^- with its utility sub-function U_1^- for the example. This pattern matches if the usage of a provided interface of a started component in a shop has caused five or more failures (exceptions), which decreases the utility of the shop by U_1^-. We may define U_1^- similar to U_1^+ and apply this pattern for all shops to obtain the total negative impact on the overall utility.

In general, we define *multiple* positive and negative patterns, that is, $\text{Pa}^+ = \{P_1^+, \ldots, P_k^+\}$ and $\text{Pa}^- = \{P_1^-, \ldots, P_n^-\}$. All matches of these patterns determine the overall utility $U(G)$ for the current architecture represented in the runtime model G. We define the set of all matches for the positive pattern P_i^+ in G as $M_i^+(G) = \{m | P_i^+ \models_m G\}$ and the set of all matches for the negative pattern P_j^- in G as $M_j^-(G) = \{m | P_j^- \models_m G\}$. Given these sets, the overall utility $U(G)$ can be defined and computed as follows:

$$U(G) := \sum_{i=1}^{k} \sum_{m \in M_i^+(G)} U_i^+(G, m_i) - \sum_{j=1}^{n} \sum_{m \in M_j^-(G)} U_j^-(G, m_j) \quad (1)$$

Hence, the overall utility is the sum of all U_i^+ for each match in $M_i^+(G)$ accumulated over all k positive patterns in Pa^+ minus the sum of all U_j^- for each match in $M_j^-(G)$ accumulated over all n negative patterns in Pa^-. As discussed previously, the impact of a match on the overall utility is influenced by the specific context of the match. Thus, the utility sub-functions U_i^+ and U_j^- are paramterized by the runtime model G and the specific match m, which capture the context of the match.

In general, matches of positive and negative patterns result from changes of the environment. While existing matches of positive patterns are usually not affected by the adaptation rules (i.e., the adaptation does not do any harm to the system), matches of negative patterns should be addressed by the rules (i.e., the adaptation repairs the system).

Finally, the kind of utility functions as presented here is restricted to linear functions, which are often used for optimization (cf. [7]).

4 Utility-Driven Rule-Based Adaptation

The outlined manner in section 3 to define the utility functions for architectural runtime models in principle allows to compute the utility for each possible architectural configuration and build an optimization based approach by searching the configu-

ration space. However, such a solution would be rather wasteful if the utility would have to be computed for each configuration completely anew.

In contrast the suggested incremental rule-based approach suggest for a restricted case to estimate the impact on the utility of each rule application at runtime and derive in a greedy manner from this a good sequence of rule applications concerning the adaptation decisions.

Monitor During the Monitoring phase, the observed details regarding the changes of the model are collected form the adaptable software. The monitoring activity aggregates and filters the gathered data to determine a symptom that needs to be handed to the analysis activity [13].

For our example with the runtime architecture of mRUBiS (cf. metamodel in Figure 1), we may observe the life cycle state of a component (e.g., to monitor whether a component has stopped, crashed, or even been removed) as well as Failures such as exceptions that occur when using a provided interface.

Analyze In the analysis phase the observed changes are analyzed to update the known set of negative patterns. New matches are determined through applying an event-property-change mechanism and all old matches have to be checked if they are still valid.

As a first step we can compute the utility incrementally rather than for each configuration anew. Given a former runtime model G and an updated version G', the set of new occurrences for the negative patterns are $M_i^{-,new} = M_i^-(G')\backslash M_i^-(G)$ and for positive patterns are $M_i^{+,new} = M_i^+(G')\backslash M_i^+(G)$. Similarly, $M_i^{-,del} = M_i^-(G)\backslash M_i^-(G')$ and $M_i^{+,del} = M_i^+(G)\backslash M_i^+(G')$ capture the matches for patterns that are no longer valid. We can thus define the changes of a utility function $U(G)$ accordingly by a utility change function $U_\Delta(G', G)$ as $U(G') - U(G)$ as:

$$
\begin{aligned}
&-\sum_{i=1}^{k} \sum_{m \in M_i^{+,del}} U_i^+(G, m_i) + \sum_{i=1}^{k} \sum_{m \in M_i^{+,new}} U_i^+(G', m_i) \\
&-\sum_{j=1}^{n} \sum_{m \in M_j^{-,del}} U_j^-(G, m_j) + \sum_{j=1}^{n} \sum_{m \in M_j^{-,new}} U_j^-(G', m_j)
\end{aligned}
\tag{2}
$$

Figure 4 describes an analysis phase that checks for and detects the negative pattern introduced in Figure 3 such that the annotation CF2 pointing to the affected component is created by an in-place model transformation. The occurrence of the negative pattern results in a drop in the utility of the shop by U_1^- which is stored in the issue CF2 as utilityDrop to be used later for ordering the issues. Here we omit the details to exclude multiple annotations for the same negative pattern in the model transformation for the purpose of simplicity.

Plan Based on the marking for any new or remaining issues in form of matches m for negative patterns, the approach incrementally proceeds during the planning phase by (1) computing the set of all possible rule applications, (2) selecting for each

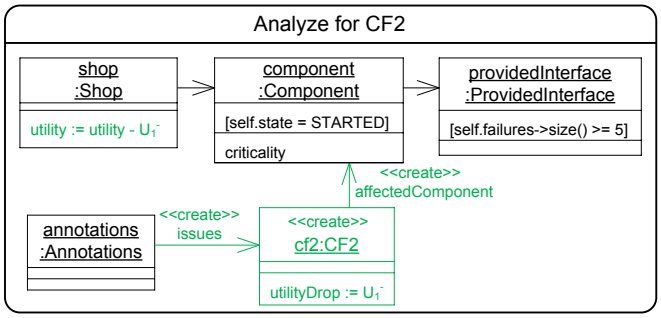
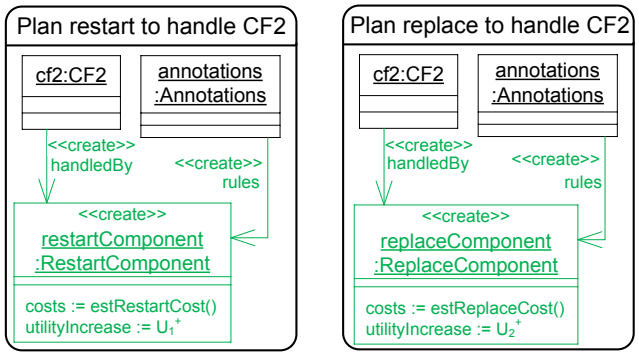


Figure 4: Marking a Negative Architectural Utility Pattern.



(a) (b)

Figure 5: Rules for Planning an Adaptation.

issue the best rule application based on estimates for the impact on utility and cost, and (3) finally ordering the best rule application for all issues to minimize the lost reward.

Based on these assumptions we can also compute all matches for rules incrementally, if for the related negative pattern P_i^- the set of new matches $M_i^{-,new}$ is given.

Figure 5 illustrates a simplified view of a planning rule for the example to repair the issue `CF2`.

Execute The execute phase takes over the ordered list of adaptation rule matches from the planning phase and executes them in the given order.

5 Experimental Evaluation

To evaluate the proposed approach we use a component-based system realizing an online marketplace mRUBiS that hosts 100 shops each including 18 components with different criticality and connectivity. We employ an alternative approach to compare to our rule-based utility-driven approach.

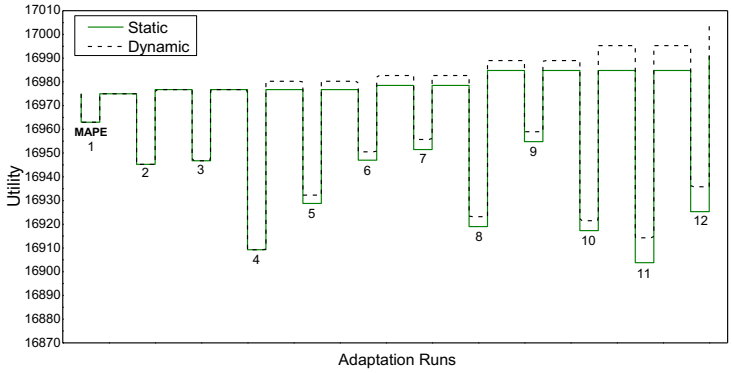


Figure 6: Utility Changes During the System Life Cycle.

Static Approach: In this approach the cost and utilityIncrease attributes of the rules are defined at design time, hence, for each CF the repair rule is selected off-line. The utilityDrop caused by each issue CF is also estimated at design time which leads to a fixed order in which the issues are addressed.

Dynamic Approach: The proposed dynamic approach, estimates the impact of the different adaptation plans at runtime and selects the one with largest impact. The order in which CFs are addressed is decided based on the runtime observations regarding the affectedComponent and the drop in the overall utility caused by the issue.

To evaluate the general performance of the proposed rule-based utility-driven approach we tested it on mRUBiS for 12 adaptation cycles (MAPE cycles) against the static approach. The experiment was conducted under similar circumstances in which during each adapt cycle both approaches face the same issues. The measurements where done and averaged for 1000 rounds for each approach. As presented in Figure 6, in each cycle, occurrences of CF issues result in utility drop for both approaches followed by a MAPE loop during which the adaptation takes place. Application of adaptation rules causes an increase in the utility, the system life cycle continues with the same utility until the next failure occurs and the utility drops again. We distinguish between applying an adaptation rule on the runtime model E' during the execute and executing it on the running system underneath E'' such that $E := E' + E''$.

Figure 6 confirms that our approach achieves higher utility values compared to the static approach.

As argued earlier, the overhead of the dynamic approach is quite negligible considering the required time to apply the adapt rules on the running system rather than the runtime model. Therefor this feature is skipped in Figure 6.

Figure 7 shows the difference between the overhead caused by our *MAPE* phases and the one of static approach compared to applying the rules on the system. The difference between the computation time for the dynamic approach compared to the static one is approximately 1.32 ms and thus only 0.04 % and therefore is negligible regarding the benefits of the approach. Note that also the time required for *MAPE'*

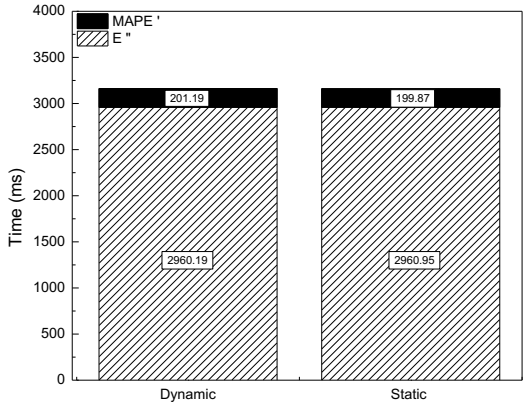


Figure 7: Overhead of the Static and Dynamic Approaches Compared to the Time for Applying the Rules on the System.

phase in the dynamic approach is approximately 6.8 % of the time E'' requires to make changes on the running system.

6 Related Work

There have been several approaches which are more flexible regarding runtime changes. Such approaches do not explicitly describe all the possible configurations of the system a priori, the runtime reasoning has to calculate utility values for all of them, thus encountering scalability and efficiency issues [6]. There have been plenty of research on utility functions and utility-driven decision-making policies [1, 7, 8]. The outlined utility-driven approaches pursue a search-based optimization in the solution space that often do not scale well for complex systems with large solution spaces. Such approaches manage to find the optimal configuration but there is no guarantee to reach the result within a reasonable time in case of the need for a quick adapt plan. The proposed approach estimates the utility for each potential adaptation strategy in an incremental scheme taking into account the current change events that affect the archived reward. We transform the definition of utility values form utility of each configuration to the utility per change event and this allows us to scale well for large systems.

On the other end of the spectrum there are rule-based adaptation approaches with a set of predefined adaptation strategies which apply off-line planning determined and developed at design time regardless of the runtime data. The adaptation logic is coded as an event-condition-action policy. They employ the predefined rules to achieve a predefined goal configuration and the adaptation policy is embedded into the system [5, 9]. Change events trigger the adaptation rules and they result in changes in system configuration [9]. Every rule-based adaptation methodology includes a set of adaptation rules, each attributed to a specific event or context and become applicable as the related event occurs [3].

The rule-based strategies basically predefine the whole adaptation process and they are recognized to be efficient and stable in predictable domains and support the early validation [6]. These approaches provide quick recovery from the goal violation however they often result in sub-optimal solutions since they ignore the unforeseen scenarios at design time.

The proposed approach is distinguished from the existing work as it is fast and efficient since it benefits from a rule-based adaptation and does not struggle with scalability issues and the lack of optimality in a rule-based approach is compensated for through combining it with a utility-driven approach which optimized decisions at runtime. However, unlike the optimization-based approaches, the incremental manner of estimating the utility function over the patterns makes the approach suitable for large complex systems.

7 Discussion

We proposed a novel approach to improve self-healing reward by combining utility-driven and rule-based adaptation at the architectural level in order to achieve the benefits of each of them. The approach addresses the requirements of scalability and optimality regarding the utility. The overhead of our approach is approximately 0.04 % compared to the static alternative, which is negligible considering the provided improvements in the overall utility. By defining the utility functions over architectural patterns, our approach is able to look for an optimal solution at runtime where adaptation rules handle the scalability issue. Here we define linear utility functions for dynamic architectures and link them to the adaptation rules as future work, we plan to investigate non-linear utility functions.

References

[1] M. Amoui, M. Derakhshanmanesh, J. Ebert, and L. Tahvildari. "Achieving dynamic adaptation via management and interpretation of runtime models". In: *Journal of Systems and Software* 85.12 (2012), pages 2720–2737.

[2] G. Blair, N. Bencomo, and R. B. France. "Models@run.time". In: *Computer* 42.10 (2009), pages 22–27. DOI: 10.1109/MC.2009.326.

[3] S.-W. Cheng. "Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation". PhD thesis. School of Computer Science, Carnegie Mellon University, Pittsburgh, USA, 2008.

[4] N. Esfahani, A. Elkhodary, and S. Malek. "A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems". In: *IEEE Transactions on Software Engineering* 39.11 (2013), pages 1467–1493.

[5] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J.-M. Jézéquel. "Modeling and Validating Dynamic Adaptation". In: *Models in Software Engineering*. Volume 5421. LNCS. Springer, 2009, pages 97–108.

[6] F. Fleurey and A. Solberg. "A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems". In: *MoDELS'09*. Volume 5795. LNCS. Springer, 2009, pages 606–621.

[7] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. "Using Architecture Models for Runtime Adaptability". In: *IEEE Software* 23.2 (2006), pages 62–70.

[8] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. "Using Product Line Techniques to Build Adaptive Systems". In: *SPLC'06*. IEEE, 2006, pages 141–150.

[9] J. Keeney and V. Cahill. "Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework". In: *POLICY '03*. IEEE, 2003, pages 3–14.

[10] J. O. Kephart and D. Chess. "The Vision of Autonomic Computing". In: *Computer* 36.1 (2003), pages 41–50.

[11] J. O. Kephart and W. E. Walsh. "An Artificial Intelligence Perspective on Autonomic Computing Policies". In: *POLICY'04*. IEEE, 2004, pages 3–12.

[12] T. Vogel. *Modular Rice University Bidding System (mRUBiS)*. 2013. URL: http://www.mdelab.de (last accessed 2016-10-01).

[13] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker. "Model-driven Architectural Monitoring and Adaptation for Autonomic Systems". In: *ICAC'09*. ACM, 2009, pages 67–68.

Programming Models for Consistent Memory Access in Shared Something Architectures

Andreas Grapentin

Operating Systems and Middleware
Hasso-Plattner-Institut
andreas.grapentin@hpi.uni-potsdam.de

Recent generations of large-scale computing systems generally fall into one of two categories, in respect to memory access. On one end of the spectrum, there are tightly-coupled multiprocessor systems, where the entire physical address space is byte-addressable by all processing units, and on the other side, there are distributed systems, where individual self-contained systems are grouped together by network interconnects, and memory access beyond system boundaries needs to be implemented through message passing. Future generations of large-scale systems promise to break this pattern by introducing a *shared something* architecture, in which memory is attached to the processing units via network interfaces, and is made accessible by all processing units in its entirety, albeit with limited cache coherence. While programming models for cache coherent shared memory systems as well as distributed memory systems are well understood, the possible benefits of a shared something architecture, as well as the challenges posed to applications in terms of correctness and performance on such platforms need further investigation, in order for suitable programming models to be found.

1 Memory Access Overview

Memory access is a central aspect of computing. Over the course of its life cycle, a typical computer program will issue many thousand read and write instructions in order to process data from the machines main memory. Recent developments in system architecture have coined the phrase *Memory-centric Computing*, or *Memory-centric Architecture*, indicating that the importance of data and memory in computing is only going to grow. While the process of reading from and writing to cells in memory sounds straightforward, there are two aspects in modern computing architectures that complicate things significantly. Firstly, the concurrent nature of modern *Symmetric Multiprocessing (SMP)* systems may lead to interleaved read and write access from independently running processing units to the same memory cell. The order in which concurrent reads and writes are executed may, depending on the application, alter the result of the program if not guarded correctly by the programmer. Secondly, the integration of multiple levels of caches, while providing a significant performance boost to applications through decreased memory latency, creates a hierarchy of redundant information that the processing units need to keep maintained. If one processing unit in the system changes a piece of information in main memory, then another processing unit that still may have that piece of information stored in its local cache needs to invalidate their version of the information in order for all pro-

cessing units on the system to have a consistent view of the main memory. Without that mechanism, a read of that memory cell would yield a wrong result, as it would be fetched directly from the cache holding the outdated value, instead of from the main memory. This behavior would break the systems expectations for strict consistency. Maintaining a strict consistency across the views of the main memory for the processing units of a system is achieved by special cache coherence protocols and interprocess communication. While necessary, this interprocess communication poses a significant overhead to the overall performance of a system and is one of the factors detrimental to the scaling of modern centralized SMP systems, the other ones being the latency and throughput of far memory access in *Non-uniform Memory Access (NUMA)* architectures and bottlenecks in the communication to shared resources and buses.

This situation is different on distributed systems. Distributed systems are clusters of independent smaller computing systems that are connected via network interconnects. These kinds of systems do generally not expose native shared memory interfaces to all the connected machines. Instead, individual machines may communicate via *Message Passing Interface (MPI)* or distribute the workload in a way that communication between the machines is eliminated, apart from an initial setup phase, and a finalization phase. One standardized approach for such a workload distribution is *MapReduce*. These approaches are most effective for highly parallel computing workloads, which also benefit the most from the almost limitless parallelism of distributed systems. Accessing remote memory from one node in the distributed system to another or establishing a shared memory region across all nodes is possible, but needs to be implemented as an additional software layer. In that case, concurrency problems analogous to SMP systems arise. However, as all the remote memory access interfaces used are implemented in software, custom locking mechanisms can be implemented and used to maintain strict consistency, or weaker consistency levels as described by Nitzberg et al. [9], depending on the requirements of the application. The overhead of such approaches is significant though, as locking and synchronization mechanisms for strict consistency across the nodes introduces network latency to the memory access.

2 Shared Something Architecture

The notion of a *Shared Something* architecture lies conceptually between an SMP, or *shared memory* architecture and a distributed, or *shared nothing* architecture, and promises to mitigate the scaling problem of the shared memory architecture while being more tightly coupled than shared nothing systems. These systems are generally understood to be composed of numerous processing units with a small amount of local memory each, all of which share a byte-addressable pool of global memory. The communication between the processing units as well as access to the shared memory is realized through high speed network interconnects. This architecture is similar to that of an SMP system in that there is a pool of globally shared, byte-addressable memory. The key difference is, that in order to improve the scaling behavior of the

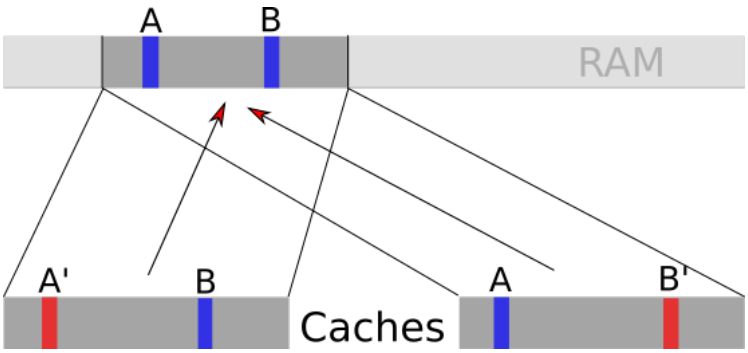


Figure 1: *False sharing* may lead to data corruption in shared something architectures, as cache lines may be altered and written back to memory concurrently, without strict consistency enforcement.

system, there is no transparent cache coherence enforced among the processing units, which removes the need of interprocess communication on memory access. Additionally, the memory hierarchy is flat, which eliminates NUMA characteristics from the behavior of the system.

This architecture also behaves very similarly to a distributed system in that each of the processing units could be interpreted as a node in a distributed system, containing a certain amount of dedicated local memory, behaving independently of the other nodes and being connected to the system via high speed network interconnects. The key difference to a distributed, or *shared nothing* system is that there is a large globally shared memory. However, since that shared memory is byte addressable directly via the hardware instructions provided by the processing units, there is no possible way to implement a strict consistency protocol in software, allocations can be made freely and reads and writes are not governed by a software layer. Also, contrary to SMP systems, there is no transparent cache coherence protocol to maintain strict consistency. Consequently, dedicated programming models need to be utilized by the software developer on the system, in order for programs to behave correctly while accessing shared memory under strict consistency requirements.

The shared something architecture promises to circumvent the issue of the scaling barrier for SMP systems, while being a lot more compact in design than distributed systems, by outsourcing the task of maintaining a consistent view on shared memory regions to the developer, instead of the system. While this allows the developer to fine tune how much synchronization and consistency is really needed, and hence gain a significant performance boost, it also poses challenges to the correctness of shared memory access. Depending on the currently unknown specifics of the actual hardware, there are several situations where unguarded memory access could lead to errors in application behavior, and even data corruption. One such scenario is outlined in Figure 1. Two unrelated values A and B are stored close to each other in memory, such that they are placed in the same cache line upon being accessed by a processing unit. Two processing units now access this cache line concurrently, where one processing unit changes A to A', and the other changes B to B'. The cache lines

Listing 1: Pseudocode description of the *out*, *in* and *read* function of the tuple spaces interface.

```
def out(N, P2, ..., Pn):
  t = n—tuple (N, P2, ..., Pn)
  insert t into the tuple space
  return

def in(N, P2, ..., Pn):
  t = find matching tuple (blocking)
  remove t from tuple space
  return t

def read(N, P2, ..., Pn):
  t = find matching tuple (blocking)
  return t
```

have now diverged and will be written back to memory sequentially, resulting in one of the writes to be lost. This scenario is more generally known as *false sharing* [2]. In SMP systems, false sharing is a problem for performance, as cache coherence protocols will ensure strict consistency across memory writes, such that the cache line will be invalidated before the write operation can take place, but in shared something systems it becomes a problem for correctness, because no strict consistency is enforced, and incorrect data may be written back to memory. Consequently, in order for concurrent global memory access to behave correctly, programmers writing applications for the shared something architecture need to ensure the required level of consistency manually, or need to utilize a suitable programming model for shared memory on the shared something architecture that guarantees the required level of consistency.

3 Programming Models for Distributed Shared Memory

In the world of distributed systems, there are a number of different approaches to emulate shared memory for distributed programs. This is done mostly for convenience, as most developers are very familiar with the shared memory abstraction. Nitzberg et al. have compared several of these approaches [9] in respect to the granularity of allocations, coherence semantics and the underlying synchronization primitives, and found them to be viable mechanisms for distributed communication.

One of the distributed memory systems Nitzberg et al. evaluated is *Tuple Spaces*. Tuple Spaces, as formally specified in the context of the *LINDA* programming language by Gelernter [5], is an interface definition for a distributed storage of immutable n-tuples. Arbitrary tuples can be produced into the storage, and can then be retrieved at a later time, where the retrieval of a tuple can be parametrized in a way that al-

Listing 2: Pseudocode of a producer / consumer queue based on tuple spaces.

```
def produce(value):
  out(('queue', value))

def consume():
  t = in('queue', None)
  value = t[1]
  do_something(value)
```

lows efficient filtering. The basic functions defined in this interface are described in pseudocode in Listing 1.

The *out* function in this context is used to produce tuples into the tuple space. An n-tuple in this context can be thought of as a composite with n elements. The *out* call should not block the programs execution. The *in* and *read* calls are very similar, both search for a tuple in the space that matches their list of parameters, there is a distinction that is made here in respect to formal and actual parameters. When retrieving a tuple from the tuple space, actual parameters passed to the *in* and *read* functions must match the values present in the tuple, while formal parameters can be thought of as wildcards, or variables that will be filled with the values read from the tuple upon retrieval. The *in* and *read* calls will block if no matching tuple can be found. If a matching tuple is found, both functions will, depending on the programming language, either assign values to the given formal parameters, or return the found tuple. The *in* function will additionally delete the matched tuple. An example of a simple distributed producer/consumer queue using these functions is outlined in Listing 2.

The described functions can now run concurrently in a distributed system. The library that implements this interface is then responsible to maintain the pool of tuples across the distributed system and to synchronize the calls to *out* such that the information remains consistent and no tuples are lost. An additional requirement is, that tuples may not be retrieved twice, so calls to *in* and *read* must be guarded as well to maintain consistency. This model can be applied to the shared something architecture in order to hand control of the memory access synchronization back to the library developer and relieving some responsibilities from the application developer. The tuple spaces library would be responsible to synchronize calls to *out* on the concurrently working processing units on the shared something architecture, such that strict consistency is guaranteed and no data corruption is caused by false sharing. The overhead introduced through this amount of synchronization is expected to be less than the overhead of a fully fledged cache coherence protocol, while not sacrificing correctness. This should allow for better scaling of tightly coupled shared something systems, as opposed to SMP systems. Several implementations for tuple spaces exist, the most notable ones being *JavaSpaces*, *Linda* and *PyLinda*.

The author would like to address, that the names of the functions of this interface seem poorly chosen, as they, in the case of *out* and *in*, do not properly reflect what

the intent of the function is, and in the case of *read*, conflict directly with the POSIX function of the same name. More widely accepted names for these functions would have been, due to the inherent similarities to stack datastructures, *push*, *pop* and *peek* respectively.

4 Experimental Evaluation

At the time of this writing, the underlying hardware of the shared something architecture is not yet publicly available. However, there are projects dedicated to the emulation of certain aspects of the architecture, for example the *Fabric Emulation* project provided directly by HPE on GitHub. This project provides an automated setup of virtual machines where each of the machines has access to a common shared memory device. It is currently not clear how similar reads and writes to this shared memory device are compared to access to the real hardware. Nevertheless, the architecture is laid out similarly to the proposed hardware. A set of independent processing units, each with a set of local resources, having access to a shared memory pool where consistency is not enforced.

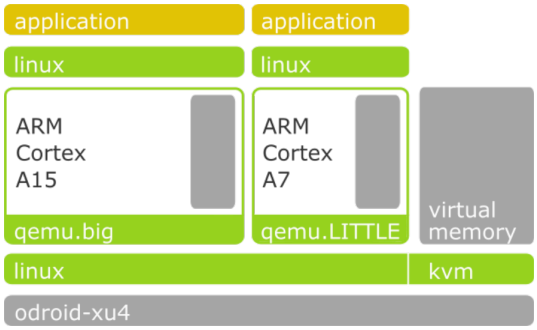


Figure 2: The architecture of the embedded shared something emulator. Separate virtual machines are executed on heterogeneous cores to balance a shared workload.

In addition to the fabric emulation, I have worked on leveraging heterogeneous embedded such as the *Odroid XU4*. This particular board contains two processing units, which are binary compatible but differ in computing power. The original idea of this platform was to provide a system that would be able to migrate the complete workload to the stronger or weaker set of cores, to optimally accommodate the total system load. However, when all cores are active simultaneously, and virtual machines are deployed and assigned to mutually exclusive processing units, it resembles an architecture where heterogeneous processing units have access to a shared memory pool that would allow very efficient migration of workload. This setup is illustrated in Figure 2. However, the focus on heterogeneous systems has diminished in the

context of the shared something architecture, so this thread of thought has not been followed further yet.

Beyond that, HP has claimed to have a cycle exact simulator for their *The Machine* project, which implements the shared something architecture, and which could help in determining the performance of shared memory management tools on these architectures. We are currently in the process of discussing access to this simulator with HP.

5 InstantLab integration

In order to make experiments on the shared something architecture repeatable and more accessible, they can be integrated into the *InstantLab* platform. InstantLab is an *Infrastructure as a Service (IaaS)* platform developed and designed to be able to provide access to predefined test environments for specific lab experiments by Neuhaus et al. [8]. The platform is still under active development and is used to provide the infrastructure for the steadily growing historic operating systems gallery maintained by the operating systems and middleware chair, as well to support the exercises in the operating systems lectures. The underlying virtualisation techniques used by InstantLab and the Fabric Emulation tool are very similar in nature, which allows an integration of the FabricI Emulation infrastructure as a predefined test environment into InstantLab.

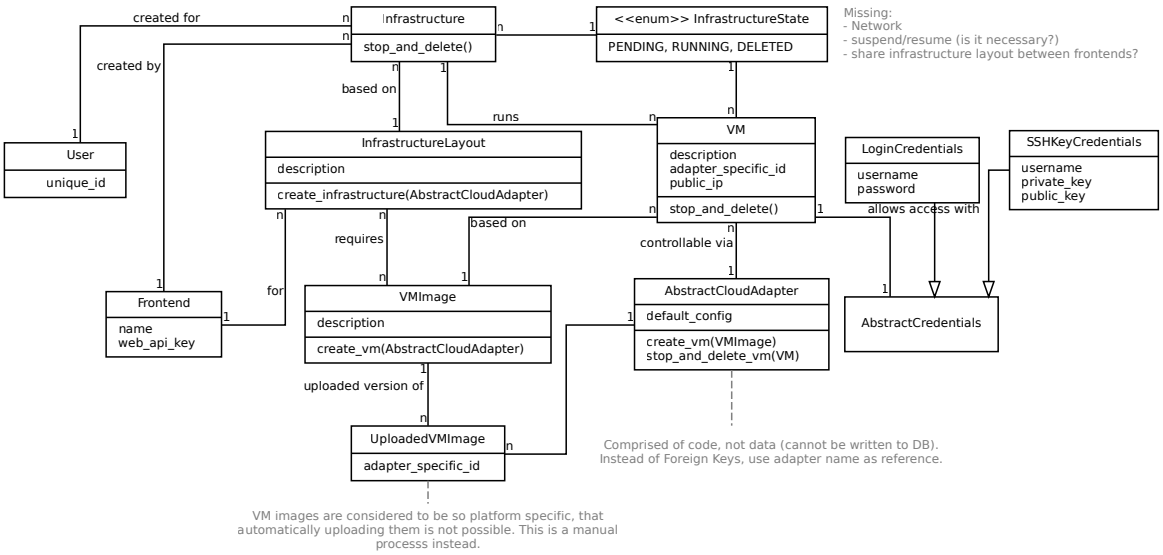


Figure 3: The data model of InstantLab. A shared something architecture lab would be set up as an *InfrastructureLayout*, where the separate virtual machines are attached as dormant instances of *VMImage*.

Architecturally, InstantLab is separated into three tiers. The first tier consists of frontends, which stand separate from the infrastructure providing service. These frontends communicate via a web-based API to a middleware. The middleware then employs a set of cloud stack adapters, to allow maximum flexibility in the management of the infrastructure resources. One of these cloud stack adapters is a *qemu/kmv* adapter that we developed specifically for optimized configurability. We have integrated support for shared memory devices into this adapter, which will allow the setup of lab experiments for this emulated shared something architecture in the future. The data model of InstantLab is outlined in Figure 3.

6 Future Work

The next steps of my work will include implementing a naïve version of tuple spaces on the shared something architectures using separate memory regions and interprocess communication to achieve strict consistency across the machines. This implementation will then be tested on the fabric emulation infrastructure for correctness and performance. Afterwards, I will need to leverage the properties of the shared memory to improve the performance of the synchronization without sacrificing correctness or strict consistency properties of the approach. The properties of the naïve and shared memory implementations compared might allow an estimate of the performance that the approach would be able to achieve on the real hardware.

Afterwards, I want to look into additional shared memory abstractions for distributed systems, and implement these with optimizations for the shared something architecture, and compare how different levels of consistency semantics are suited for the architecture and what kind of overhead they produce. These results will allow for a model of overhead per consistency level on the architecture and a guideline for programmers to follow and optimize for. Examples for shared memory implementations I want to evaluate in this context are *Ivy* [7] for strict consistency, *Munin* [1] for weak consistency and *Dash* [6] for an example for a hardware based approach.

Additionally, I would like to look into other distributed programming models to evaluate their relevance for my thesis. One example of such a programming model is *split c* [3], a parallel extension to the C language that emulates a shared global address space for distributed programs. This model shows the potential of being useful in the context of managing an existing global address space while not sacrificing the power and familiarity of the memory allocation methods of C. A second example is *Partitioned Global Address Space* [4], as a different way to manage large amounts of shared memory by making most of that memory local to single processes. This introduces an explicit semantics to memory sharing that may be useful in mitigating the challenges of the shared something architecture.

The InstantLab Integration will also need to be realized. This could then be used to offer a programming environment of the shared something architecture to students, in order to evaluate how programmers approach the challenges of this architecture, and how well the implemented memory management approaches hold up in practice.

An integration of an exercise on this platform into the Operating Systems lecture, or an accompanying seminar might also be thinkable.

Finally, once the systems implementing shared something architecture are publicly available, we can start implementing and evaluating these concepts on real machines and measure the performance of the theorized approaches. We will compare the performance of the theorized approaches against direct ports of shared memory approaches for distributed systems on the same hardware, in the hope of showing a significant performance boost, while reducing the memory footprint due to reduced duplication.

References

[1] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. *Munin: Distributed shared memory based on type-specific memory coherence*. Volume 25. 3. ACM, 1990.

[2] W. J. Bolosky and M. L. Scott. "False sharing and its effect on shared memory performance". In: *Proceedings of the Fourth symposium on Experiences with distributed and multiprocessor systems*. 1993.

[3] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. Von Eicken, and K. Yelick. "Parallel programming in Split-C". In: *Supercomputing'93. Proceedings*. IEEE. 1993, pages 262–273.

[4] M. De Wael, S. Marr, B. De Fraine, T. Van Cutsem, and W. De Meuter. "Partitioned global address space languages". In: *ACM Computing Surveys (CSUR)* 47.4 (2015), page 62.

[5] D. Gelernter. "Generative communication in Linda". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7.1 (1985), pages 80–112.

[6] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. *The directory-based cache coherence protocol for the DASH multiprocessor*. Volume 18. 2SI. ACM, 1990.

[7] K. Li. "IVY: A Shared Virtual Memory System for Parallel Computing." In: *ICPP (2)* 88 (1988), page 94.

[8] C. Neuhaus, F. Feinbube, A. Polze, and A. Retik. "Scaling Software Experiments to the Thousands." In: *CSEDU (1)*. 2014, pages 594–601.

[9] B. Nitzberg and V. Lo. "Distributed shared memory: A survey of issues and algorithms". In: *Distributed Shared Memory-Concepts and Systems* (1991), pages 42–50.

Metamaterial Mechanisms

Alexandra Ion

Human Computer Interaction group
Hasso-Plattner-Institut
alexandra.ion@hpi.uni-potsdam.de

Recently, researchers started to engineer not only the outer shape of objects, but also their internal microstructure. Such objects, typically based on 3D cell grids, are also known as metamaterials. Metamaterials have been used, for example, to create materials with soft and hard regions.

So far, metamaterials were understood as materials – we want to think of them as machines. We demonstrate metamaterial objects that perform a mechanical function. Such metamaterial mechanisms consist of a single block of material the cells of which play together in a well-defined way in order to achieve macroscopic movement. Our metamaterial door latch, for example, transforms the rotary movement of its handle into a linear motion of the latch. Our metamaterial Jansen walker consists of a single block of cells – that can walk. The key element behind our metamaterial mechanisms is a specialized type of cell, the only ability of which is to shear.

In order to allow users to create metamaterial mechanisms efficiently we implemented a specialized 3D editor. It allows users to place different types of cells, including the shear cell, thereby allowing users to add mechanical functionality to their objects. To help users verify their designs during editing, our editor allows users to apply forces and simulates how the object deforms in response.

1 Introduction and Background

Researchers in human-computer interaction have explored the use of personal fabrication tools, such as 3D printers [14] to help users design the external shape of 3D objects [15]. In order to add functionality to 3D printed objects, researchers integrated electronics, even printed optics [16], or loudspeakers [6].

Researchers also started to design the inside of 3D objects by changing the structure of the 3D printed object itself. Initial projects optimized only a single parameter, such as the object's strength-to-weight ratio [7] or the position of the object's center of mass [12].

Recently, researchers started to push internal structures even further and created objects that consist internally of large numbers of 3D cells organized on a regular grid [13]. Since these objects allow each cell to be designed differently, the resulting objects literally offer thousands of degrees of freedom. These types of structures have also been referred to as metamaterials. Metamaterials are artificial structures with mechanical properties that are defined by their usually repetitive cell patterns, rather than the material they are made of [11].

Based on this concept, researchers have created objects with unusual behaviors, such as metamaterials that collapse abruptly when compressed [9], that shrink in

two dimensions upon one-dimensional compression [2], or objects that mix layers of soft and hard cells in order to emulate different materials [1].

So far, metamaterials have been understood as materials. The main contribution of this paper is that we want to think of them as machines.

In this paper, we push the concept of metamaterials further by creating objects that allow for controlled directional movement. This allows users to create objects that perform mechanical functions. Our objects thereby implement devices that transform input forces and movement into a desired set of output forces and movement – also known as mechanisms.

2 Metamaterial Mechanisms

Figure 1a shows an example of a metamaterial mechanism: a door latch mechanism. Its interior is a regular grid of 3D cells; however, the cells are of different types. Figure 1b shows how applying a force causes the cells to deform in a controlled way, thereby performing the intended mechanical function. In the example, rotating the door handle causes cells inside of the object to deform, ultimately pulling the latch towards the left and thereby unlocking the door.

While most of the object consists of rigid cells (cells that are reinforced with a diagonal), the object also contains several rectangular regions of cells that lack such a diagonal reinforcement. As shown in Figure 2, these are the key to creating mechanisms, as they are able to deform in a very specific way: when subjected to an external force, these cells shear and thereby apply a force to their neighboring cells. Our recent paper [5] details how this basic principle allows creating mechanisms.

Metamaterial mechanisms are simple. While the traditional door latch mechanism consists of several parts, including an axle, bearings, springs, etc., the metamaterial door latch in Figure 1 consists of a single block of material, as it is groups of cells inside the object that perform the mechanical function.

While in previous work metamaterials were typically generated by scripts [9, 11], creating mechanisms requires a dedicated design/engineering process that we argue is best performed by means of an interactive editor. Figure 1c shows a preview of the custom 3D editor we created specifically to allow users to create and modify metamaterial mechanisms. It contains a range of functions that help users assemble specialized cells into basic mechanisms and to assemble such basic mechanisms into more complex mechanisms and simple machines.

Using this editor, we have created a series of demo objects shown in Figure 3. Additionally to the door latch from Figure 1, we demonstrate a Jansen walker, a functional pair of pliers, and a pantograph. Our examples were printed on an *Ultimaker 2* 3D printer using *NinjaFlex* filament (in pink).

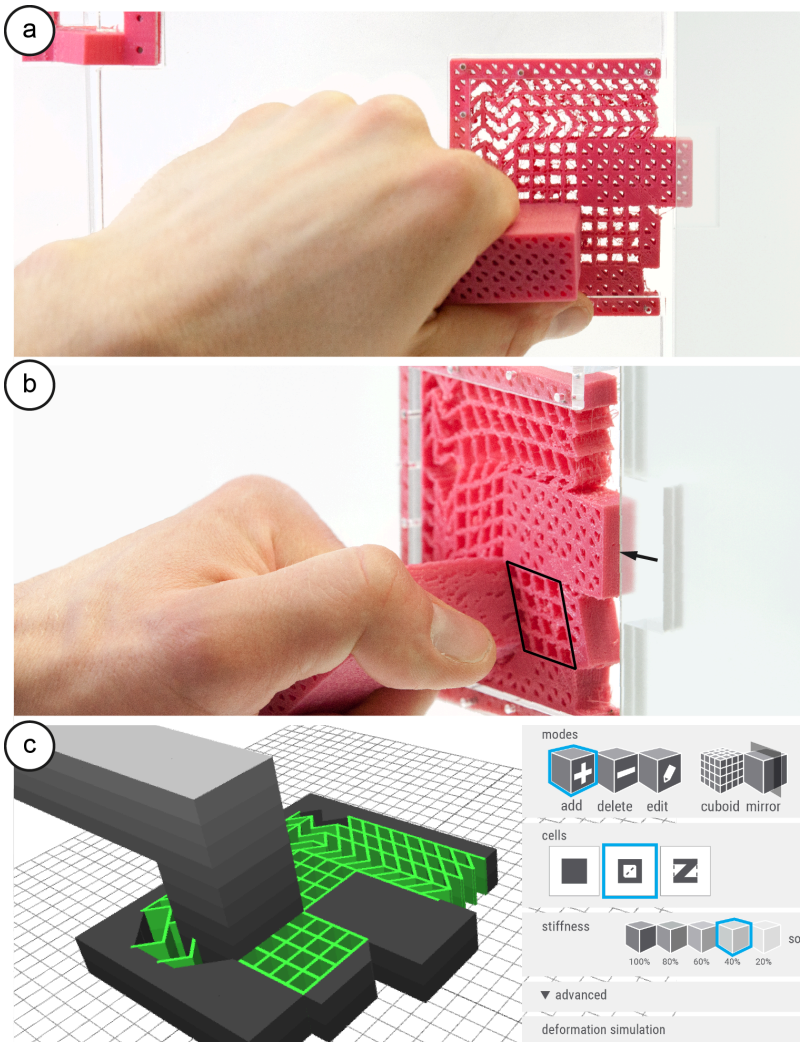


Figure 1: (a) This door latch is implemented as a metamaterial mechanism; it consists of a single block of material based on a regular grid of cells that together implement handle, latch, and springs. (b) Turning the handle causes the central hinge array to deform and to pull the latch inward, which unlocks the door. (c) We created this mechanism in our custom editor. Here, we placed two hinge arrays that mechanically couple the handle to the latch, and cells that couple to the doorframe.

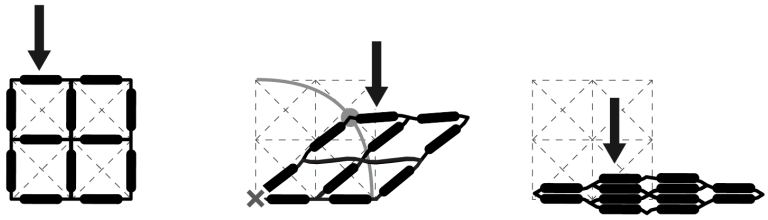


Figure 2: When a shear cell in this 2×2 block is subject to compression forces, it complies by shearing on a circular trajectory until its members are packed tightly.

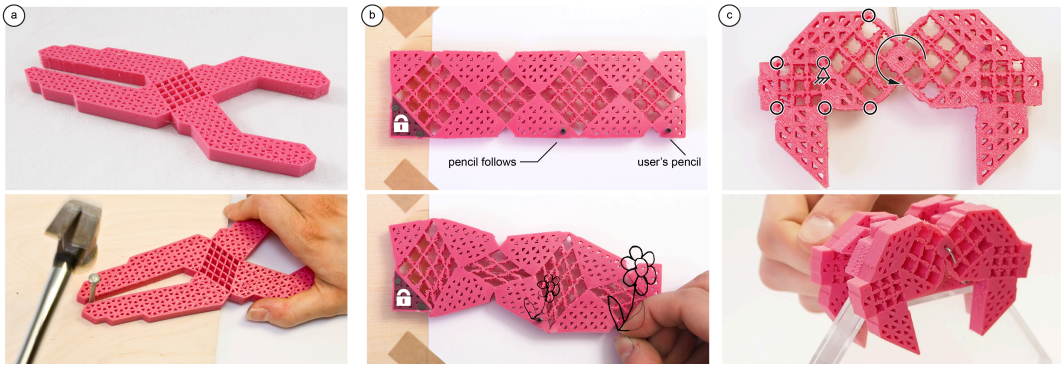


Figure 3: We demonstrate the concept of metamaterial mechanisms at the example of (a) a functional pair of pliers, (b) a pantograph that copies drawings, and (c) a Jansen walker.

3 Contribution, Benefits, and Limitations

Our main contribution is the concept of metamaterial mechanisms. Our main software contribution is a specialized editor that allows users to create them.

We extend the research field of metamaterials by contributing a general purpose approach to creating mechanisms. Mechanisms are a new genre of metamaterial structures that is of higher complexity and that exploits more degrees of freedom than previous work in this field, and that allow metamaterials to tackle problems they have traditionally not been able to address.

Compared to traditional multi-part mechanisms, metamaterial mechanisms offer several benefits. (1) The resulting devices consist of a single part. They can thus be created using particularly simple fabrication processes, such as single-material 3D printers (e.g., fused deposition modeling printers). (2) As they consist of a single piece, they require no assembly. (3) Since the movement is performed by deformation there is virtually no friction, no need for lubrication, and thus for maintenance [4].

However, the resulting designs are also subject to limitations. Adding more cells increases the stiffness, and as a result, metamaterial mechanisms are not suitable for mechanisms that are to be operated with very small forces. Furthermore, our approach is unable to produce continuous rotation. Objects such as the Jansen walker, for example, thus require a separate axle. Also, cell designs are limited by the quality of the 3D printer. In particular, shear cells work best if their internal hinges are thin, which requires high-resolution 3D printers. Finally, while our editor vastly simplifies the creation of metamaterial mechanisms, any type of mechanical engineering requires experience – and metamaterial mechanisms are no exception here.

4 Metamaterial Mechanism Editor

To allow users to design, fabricate, and test metamaterials containing mechanisms we implemented the specialized editor shown in Figure 1c.

The main intent behind it is not only to make the editing process more efficient than the more traditional script-based editing, but also to provide users with an overview of their design, encouraging design by trial-and-error.

Our editor is based on interaction techniques known from voxel editors (such as [3]). However, in addition our software also offers specific supports for creating mechanisms, such as tools for drawing hinge arrays, etc. In order to allow users to validate their designs, the editor also allows them to apply forces and see how the object deforms in order to then refine their design directly inside the editor, before exporting to the 3D printer.

4.1 Walkthrough

Figure 4 illustrates how we created the door handle.

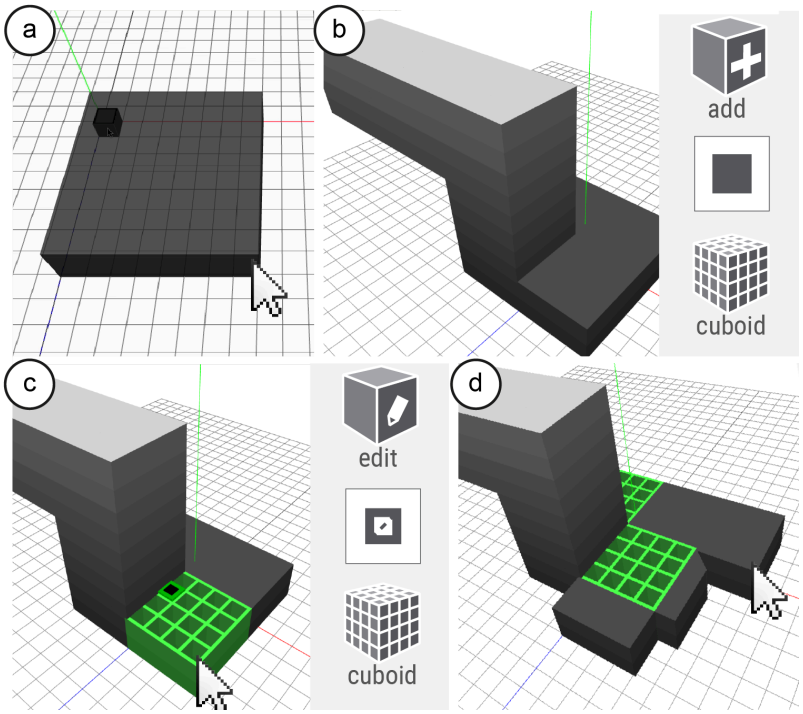


Figure 4: Walkthrough of the creating door latch mechanism. The UI elements on the right show the active tools for the respective interaction steps.

(a) We start by creating a block of rigid cells using the add brush (we can remove cells using the delete brush). Here we use the tool in cuboid mode, which allows us to draw a filled rectangular region at once by just drawing the diagonal. (b) By

adding another two cuboids on top, we create the handle. (c) We select the shear brush. Still in cuboid mode, we paint the central hinge array using a single drag interaction, which causes rigid cells to turn into shear cells. Even though the block of material we painted on is two cells high, the shear brush paints cells all the way through – as we can tell from the sidewall now being all green. This is one of the features of this brush: since shear cells backed by rigid cells would still be rigid, thus have no effect, the shear brush always cuts shear cells through the entire object.

We now verify our design directly from within the editor, as illustrated by Figure 5. (a) We select the anchor tool and use it to place a few anchor points at the bottom, indicating that the door latch is here rigidly connected to the doorframe. (b) Now we use the force tool to apply a force to the door handle. We attach a force arrow to one of the handle's cell vertices. As we are building up the force by dragging the force tool the system already responds by showing the resulting deformation of our door latch.

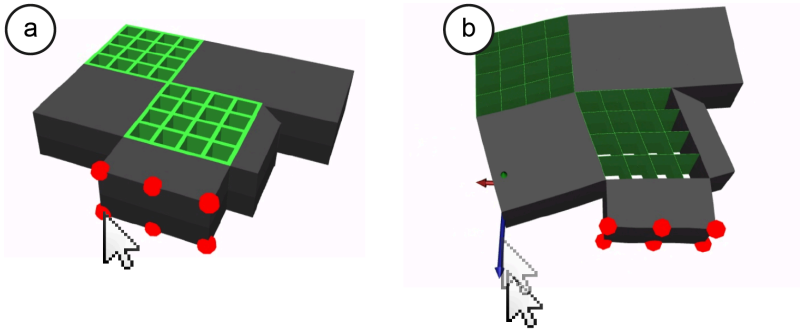


Figure 5: To simulate the deformation in real-time in the editor, (a) users set anchor points and (b) adjust forces using the force tool.

4.2 Multiple Dimensions

While the door latch mechanism actuates in only two dimensions, our editor also supports placing mechanisms in 3 dimensions. The latch mechanism shown in Figure 6, for example, combines a horizontal hinge array (blue) and a vertical hinge array (green) in order to create a mechanism that users operate by pressing down, sliding over, and releasing.

Our editor color-codes mechanisms automatically according to their orientation in space. This is intended to provide users with a fast overview of the main dimensions of action in their devices and to help recognize hinge arrays from odd viewing angles. In the latch example, green denotes "shearing on the x/z plane" and blue stands for "shearing on the x/y plane". Analogously, red stands for "shearing on the y/z plane".

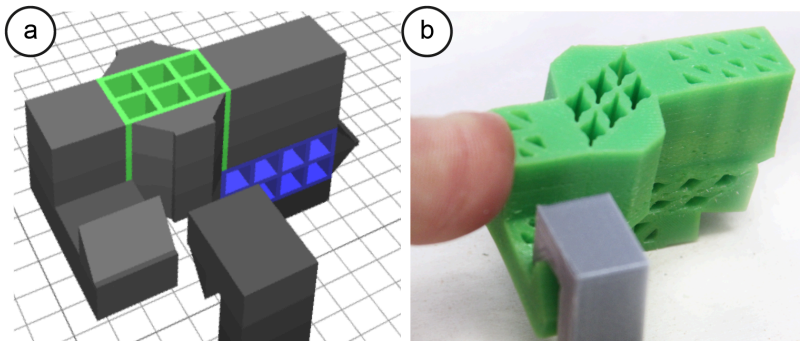


Figure 6: This latch requires the ability to shear on two planes, i.e., on the x/z plane denoted in green, and on the x/y plane indicated in blue.

Note that hinge arrays can overlap. In this case, cells at the intersection bear the combination of all holes. These cells are rendered as the additive mixture of the involved colors, such as yellow, for cells at the intersection between green and red.

4.3 Integration with Other Metamaterial Systems

The shear cell is the main element that enables metamaterial mechanisms. However, to allow for the integration with metamaterials by other researchers, the editor can be extended to allow for other cell types. In order to allow users to explore their own cell types, we offer the advanced panel shown in Figure 7. Users compose cells from individual edges by selecting the respective edges. The editor automatically adds all custom cells used in the current model to the cells panel for quick re-use.

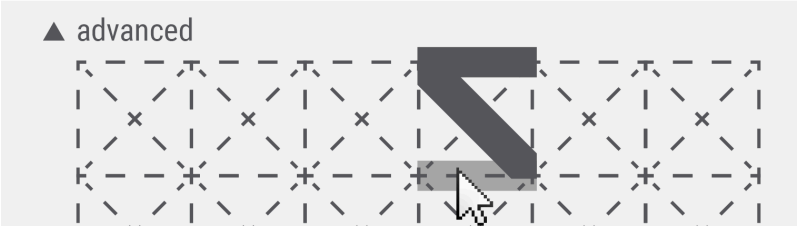


Figure 7: Users compose custom cells by adding individual edges in the "advanced" panel.

Furthermore, users can also create and store groups of cells, for example to create auxetic materials [2, 8], as shown in Figure 8. Since metamaterial mechanisms adhere to the standard structure of 3D cell grids that is common for metamaterials, they integrate with earlier research [10, 13].

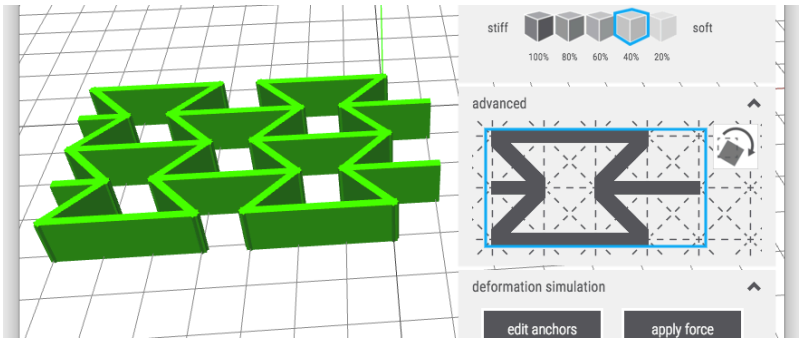


Figure 8: Users add groups of cells, here they create an auxetic material [8] from a 4×2 group of cells.

5 System Implementation

In the following, we provide details on the internal processes implemented in our metamaterial mechanisms editor.

5.1 Import

Users can import mesh geometries directly into our editor. We voxelize the meshes using binvox[1] according to the cell size that the user defined.

5.2 Editor

Our 3D editor is based on WebGL and uses three.js. Internally, the editor creates a dictionary of cells that can be accessed using their position on the grid. Each cell is defined by the 8 vertices making up its bounding box by and the edges that define how the vertices are connected. Note that not all 8 vertices need to be connected by edges.

All vertices lie on our uniform 3-dimensional grid. To generate the 3D cells' structures, i.e., to generate 3D beams from 1D edges, we apply an offset to the vertices' positions on the GPU. Since WebGL does not offer geometry shaders, we use a vertex shader and pass the offset direction and the cell's position with each vertex. The 8 vertices that form a beam are offset uniformly from the two edge vertices on the GPU. To pass additional information about the color and thickness of beams to the shader, we generate a texture where each pixel holds these data for one cell. The color maps directly and the thickness is encoded in the alpha component. In the shader, every vertex looks up its thickness in the texture and calculates the offsets for the new vertices that render a beam from an edge. This enables us to emulate a geometry shader in WebGL and perform all geometry processing on the GPU, which keeps the user experience of our editor smooth.

[1] http://www.cs.princeton.edu/~min/binvox/ (last accessed 2016-10-20).

5.3 Simulation

For simulating the deformation of the user's cell structure, we use the finite elements solver karamba,[2] which is a plugin for Grasshopper/Rhinoceros. We implemented a custom C#-Grasshopper-component that receives the mesh data (vertices and edges) and the data for the simulation (anchored vertices, force and vertex where the force applies) via a web socket connection. When the simulation is complete, a second custom component receives the transformed mesh vertices and sends them back to the editor. The vertices are kept in the same order within the array as they were received from the editor. We run the simulation on a separate machine to keep the editor running smoothly.

Maintaining the order of the vertices is important to enable geometry processing on the GPU. In the editor, we generate another texture and store the transformed vertices, where XYZ is mapped to RGB. The shader knows the vertex' undeformed position on the grid and looks up the deformed position in the texture.

Depending on the size of the object that is simulated, solving for the deformation can lead to perceivable delays. To compensate for this, our editor interpolates the deformation while the response from the simulation is pending. To do so, we pass the last force where we received the transformation from the server, and the current force that was submitted to the simulation and interpolate the vertex transformation linearly.

5.4 Export

We generate an .stl file for the user that is ready to be 3D printed. Our export is based on OpenJSCAD. In this step, we refine the cell structure from the simplified editor view to our beams with stiff members and thin living hinges. For every edge that belongs to a shear cell, we create a beam with a thick part in the middle. Edges that are part of rigid cells are generated as simple straight beams. Finally, we use OpenJSCAD's built-in render engine, which we invoke directly from our 3D editor to perform the union operations and generate the .stl file.

6 Conclusions and Future Work

In this paper, we introduced metamaterial mechanisms. While metamaterials so far had been understood as materials, the main contribution of this paper is that we think of them as machines.

On the most basic level, it was the shear cell that allowed us to implement this new perspective on metamaterials. The shear cell allowed us to redirect forces and thus to create basic mechanisms, compound mechanisms, and ultimately simple machines.

[2] http://www.karamba3d.com/ (last accessed 2016-10-20).

While our approach offers tangible benefits for users (e.g., it solves mechanical problems in a single part, thereby eliminates the need for assembly), we see the main promise of this work in that it allows us to achieve a deeper integration between the structural and the mechanical functions of materials.

For future work, we plan to continue on this path by investigating how to integrate logical functions into material.

6.1 Acknowledgements

We want to thank all the co-authors of this paper [5]: Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch.

We thank David Lindlbauer for his insights and for printing many of our prototypes. We also thank Louis Kirsch, Moritz Hilscher, David Stangl, Arthur Silber, Friedrich Horschig and Noel Danz for their contribution to earlier versions of this work.

References

[1] B. Bickel, M. Bächer, M. Otaduy, H. R. Lee, H. Pfister, M. Gross, and W. Matusik. "Design and fabrication of materials with desired deformation behavior". In: *ACM Transactions on Graphics* 29.4 (2010). ISSN: 0730-0301. DOI: 10.1145/1833351.1778800.

[2] J. C. Á. Elipe and A. D. Lantada. "Comparative study of auxetic geometries by means of computer-aided design and engineering". In: *Smart Materials and Structures* 21.10 (2012), page 105004. ISSN: 0964-1726.

[3] J. Hiller and H. Lipson. *VoxCAD*. Sept. 2016. URL: https://sites.google.com/site/voxcadproject/ (last accessed 2016-10-20).

[4] L. L. Howell, S. P. Magleby, and B. M. Olsen. *Handbook of Compliant Mechanisms*. John Wiley and Sons, 2013.

[5] A. Ion, J. Frohnhofen, L. Wall, R. Kovacs, M. Alistar, J. Lindsay, P. Lopes, H.-T. Chen, and P. Baudisch. "Metamaterial mechanisms". In: *Proceedings of UIST'16*. 2016. DOI: 10.1145/2984511.2984540.

[6] Y. Ishiguro and I. Poupyrev. "3D printed interactive speakers". In: *Proceedings of CHI'14*. 2014, pages 1733–1742. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557046.

[7] L. Lu, A. Sharf, H. Zhao, Y. Wei, Q. Fan, X. Chen, Y. Savoye, C. Tu, D. Cohen-Or, and B. Chen. "Build-to-Last: Strength to Weight 3D Printed Objects". In: *ACM Transactions on Graphics* 33.4 (2014). ISSN: 1557-7333. DOI: 10.1145/2601097.2601168.

[8] M. Mir, M. N. Ali, J. Sami, and U. Ansari. "Review of Mechanics and Applications of Auxetic Structures". In: *Advances in Materials Science and Engineering* (2014), pages 1–17. ISSN: 1687-8434. DOI: 10.1155/2014/753496.

[9] T. Mullin, S. Deschanel, K. Bertoldi, and M. Boyce. "Pattern transformation triggered by deformation". In: *Physical Review Letters* 99.8 (2007), pages 1–4. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.99.084301.

[10] J. Panetta, Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, and D. Zorin. "Elastic Textures for Additive Fabrication". In: *ACM Transactions on Graphics* 34.4 (2015). DOI: 10.1145/2766937.

[11] J. Paulose, A. S. Meeussen, and V. Vitelli. "Selective buckling via states of self-stress in topological metamaterials". In: *arXiv preprint* (2015), page 12. arXiv: 1502.03396.

[12] R. Prévost, E. Whiting, S. Lefebvre, and O. Sorkine-Hornung. "Make It Stand: Balancing Shapes for 3D Fabrication". In: *ACM Transactions on Graphics* 32.4 (2013). ISSN: 0730-0301. DOI: 10.1145/2461912.2461957.

[13] C. Schumacher, B. Bickel, J. Rys, S. Marschner, C. Daraio, and M. Gross. "Microstructures to control elasticity in 3D printing". In: *ACM Transactions on Graphics* 34.4 (2015). ISSN: 0730-0301. DOI: 10.1145/2766926.

[14] J. G. Tanenbaum, A. M. Williams, A. Desjardins, and K. Tanenbaum. "Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice". In: *Proceedings of CHI'13*. 2013, pages 2603–2612. ISBN: 978-1-4503-1899-0. DOI: 10.1145/2470654.2481360.

[15] C. Weichel, M. Lau, D. Kim, N. Villar, and H. W. Gellersen. "MixFab: A Mixed-reality Environment for Personal Fabrication". In: *Proceedings of CHI'14*. 2014, pages 3855–3864. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557090.

[16] K. Willis, E. Brockmeyer, S. Hudson, and I. Poupyrev. "Printed optics: 3D printing of embedded optical elements for interactive devices". In: *Proceedings of UIST'12*. 2012, pages 589–598. ISBN: 978-1-4503-1580-7. DOI: 10.1145/2380116.2380190.

Profiling the Web of Data

Anja Jentzsch

Information Systems Group
Hasso-Plattner-Institut
anja.jentzsch@hpi.uni-potsdam.de

The Web of Data contains a large number of openly-available datasets covering a wide variety of topics. In order to benefit from this massive amount of open data such external datasets must be analyzed and understood already at the basic level of data types, constraints, value patterns, etc.

For Linked Datasets such meta information is currently very limited or not available at all. Data profiling techniques are needed to compute respective statistics and meta information. However, current state of the art approaches can either not be applied to Linked Data, or exhibit considerable performance problems. This paper presents my doctoral research which tackles these problems.

1 Problem Statement

Over the past years, an increasingly large number of data sources has been published as part of the Web of Data. This trend, together with the inherent heterogeneity of Linked Datasets and their schemata, makes it increasingly time-consuming to find and understand datasets that are relevant for integration. The true value of Linked Data becomes apparent when datasets are analyzed and understood already at the basic level of data types, constraints, value patterns etc. For Linked Datasets and other Web data meta information is currently quite limited or not available at all. Such *data profiling* is especially challenging for RDF data, the underlying data model on the Web of Data. In comparison to other data models, e.g., the relational model, RDF often lacks explicit schema information that precisely defines the types of entities and their attributes.

Existing work on data profiling often can not be applied to Linked Datasets due to their different nature. To overcome this gap we introduce a comprehensive list of data profiling tasks which compute the most important statistical properties along different groupings.

Finding information about Linked Datasets is an open issue on the constantly growing Web of Data. While most of the Linked Datasets are listed in registries as for instance at the Data Hub (`datahub.io`), these registries usually are manually curated. Existing means and standards for describing datasets are often limited in their depth of information. We present approaches and challenges for cataloging Linked Datasets and retrieving basic metadata.

Data profiling often exhibits considerable performance problems. We introduce three common techniques for improving performance, and present an approach that relies on parallelization and adapts multi-query optimization for relational data to optimize execution plans of Linked Data profiling tasks [5].

As Linked Datasets are usually sparsely populated, key candidates often consist of either multiple low-density properties or cannot be found at all. We present two approaches for key discovery, a traditional unique column combination adaption and an approach that tackles the sparsity on the Web of Data by combining the uniqueness and density of properties [8]. Furthermore, since ontologies are topically clustered by their underlying ontologies, we analyze how to retrieve key candidates per topic cluster.

As Linked Datasets grow on the web, their entities and links among them form intricate graphs, and intrinsic patterns emerge. While graph patterns and pattern mining are known concepts with many methods, we propose an explorative and visual approach to engage in and understand the semantics of Linked Datasets. To this end, we formally define a set of frequent patterns based on our initial observations. We then propose an extensible algorithm to efficiently extract such patterns and their variations from very large Linked Datasets [7]. Interestingly, we observe many re-occurring motifs across various heterogeneous datasets, suggesting an underlying regularity of how data accretes.

All presented approaches are evaluated thoroughly on real-world datasets, and are implemented in the interactive Linked Data profiling suite ProLod++ [1].

2 Related Work

While many general tools and algorithms already exist for data profiling, most of them cannot be used for graph datasets, because they assume a relational data structure, a well-defined schema, or simply cannot deal with very large datasets. Nonetheless, some Linked Data profiling tools already exist. Most of them focus on solving specific use cases instead of data profiling in general.

One relevant use case is schema induction, because the lack of a fixed and well-defined schema is a common problem with Linked Datasets. One example for this field of research is the ExpLOD tool [10]. ExpLOD creates summaries for RDF graphs based on class and property usage as well as statistics on the interlinking between datasets based on owl:sameAs links.

Li describes a tool that can induce the actual schema of an RDF dataset [12]. It gathers schema-relevant statistics like cardinalities for class and property usage, and presents the induced schema in a UML-based visualization. Its implementation is based on the execution of SPARQL queries against a local database. Like ExpLOD, the approach is not parallelized. Both solutions still take approximately 10h to process a 10 million triples dataset with 13 classes and 90 properties. These results illustrate that performance is a common problem with large Linked Datasets.

An example for the query optimization use-case is presented in [11]. The authors present RDFStats, which uses Jena's SPARQL processor to collect statistics on Linked Datasets. These statistics include histograms for subjects (URIs, blank nodes) and histograms for properties and associated ranges.

Others have worked more generally on generating statistics that describe datasets on the Web of Data and thereby help understanding them. LODStats computes

statistical information for datasets from the Data Hub [2]. It calculates 32 simple statistical criteria, e.g. cardinalities for different schema elements and types of literal values (e.g. languages, value data types).

In [3] the authors automatically create VoID descriptions for large datasets using MapReduce. They manage to profile the BTC2010 dataset in about an hour on Amazon's EC2 cloud, showing that parallelization can be an effective approach to improve runtime when profiling large amounts of data.

3 Large Scale Data Profiling

The process of running data profiling tasks for large Linked Datasets can take hours to days, depending on the complexity of task and the size of the respective datasets. Data set characteristics highly influence the profiling task runtime. As an example, our *Property Cooccurrence by Resource* script runs 16 hours for only 1 million triples of the Web Data Commons RDFa dataset in contrast to 5 min on Freebase and 9 min on DBpedia.

We have compiled a list of 56 data profiling tasks implemented in Apache Pig to be executed on Hadoop. At this point Apache Pig only applies some basic logical optimization rules, like removing unused statements [6]. We present Lodop, a framework for executing, optimizing, and benchmarking such a set of profiling tasks, highlight reasons for poor performance when executing the scripts sequentially, and develop a number of optimization techniques. In particular, we developed and evaluated three multi-script optimization rules for combining logical operators in the execution plans of profiling scripts.

3.1 Multi-query optimization for Apache Pig

A prevalent goal for relational database optimization is to reduce the amount of required full table scans, which for file-based database systems effectively means reducing the amount of disk operations. Sellis introduces Multi-Query Optimization for relational databases as the process of optimizing a set of queries which may share common data [13]. The goal is to execute these queries together and reduce the overall effort by executing similar parts only once. The optimization process consists of two parts: identifying shared parts in multiple queries and finding a globally optimal execution plan that avoids superfluous computation.

Apache Pig[1] is a platform for performing large-scale data transformations on top of Hadoop clusters. It provides a high-level language (called *Pig Latin*) for specifying data transformations, e.g. selections, projections, joins, aggregations and sorting on datasets. Pig Latin scripts are compiled into a series of MapReduce tasks and executed on a cluster.

[1] http://pig.apache.org/ (last accessed 2016-10-20).

The main goals for our multi-query optimization rules for Pig are the following two: First, we attempt to minimize the dataflow between operators. In our evaluation we identified the dataflow between MapReduce jobs as a reasonable indicator for the performance of Pig scripts, as it is closely related to the amount of required HDFS operations. Second, we try to avoid performing identical or similar operations multiple times. The idea behind this is to free up cluster resources for other tasks. All optimization rules presented in this section, are based on optimizing the logical plans of Pig scripts.

Three optimization rules have been implemented: Rule 1 merges identical operators in logical plans of different scripts, Rule 2 combines `FILTER` operators, and Rule 3 combines aggregations, i.e. `FOREACH` operators. Rule 1 is a prerequisite for the other two rules, which work on pairs of siblings operators, i.e. operators that have the same parent operator in a respective logical plan. For all optimization rules, it was important to make sure that their usage does not affect the intended output of scripts.

Rule 1 – Merge identical operators In order to better utilize cluster resources, it makes sense to submit jobs to Hadoop in parallel. Lodop supports this by merging logical plans of different scripts into a single large plan. In our experiments, executing scripts in parallel as part of one large plan cuts execution time down to 25 % to 30 % of the time required to execute scripts sequentially. Once all plans have been merged together, it's possible to also merge identical operators. For 52 of our Pig scripts, this reduces the number of operators from 365 to 267.

Rule 2 – Combine filters `FILTER` operators reduce the amount of data that needs to be processed in later steps of the execution pipeline. This optimization rule aims to avoid iterating over large sets multiple times. From our selection of profiling scripts, 25 scripts perform filtering operations on the full initial dataset.

First, we identify all suitable sibling filters, i.e. all `FILTER` operators that have the same parent operator. Second, a combined filter is created and we attach it to the same parent operator. This combined filter contains all boolean expressions of existing filters concatenated via `OR`. The expression of the combined filter is cleaned up by transforming it into disjunctive normal form. Finally, we re-arrange all previous filters and move them after the combined filter.

Rule 3 – Combine aggregations `FOREACH` operators can be used for projections and aggregations. Some instances perform identical aggregations, but project different properties. This can happen, e.g. if the aggregation itself is only a preprocessing step to another aggregation. These operators are not exactly identical, so the rule for merging identical operators will not be able to merge them. However, these cases can be optimized by separating the aggregation from the projection, i.e. performing the aggregation only once with all projected columns, and then projecting the exact columns afterwards. For our set of scripts, this rule can be applied in seven different cases and combines varying numbers of `FOREACH` operators from the minimum of two to a maximum of eleven siblings operators.

While our goal is to optimize the performance of profiling tasks, the optimization rules can be applied on any Pig script.

3.2 Evaluation

The number of MapReduce jobs and the amount of dataflow in the operator pipeline are good indicators for the performance of Apache Pig scripts. Our evaluation shows that improving only on these factors does not necessarily improve overall performance. Merging identical operators reduces both the total number of operators and the number of MapReduce jobs. It comes at the cost of less parallelism. Combining filter operators was shown to reduce the execution time of map/reduce functions (i.e. CPU time). Combining aggregations can reduce the amount of HDFS I/O, and improves overall execution time for certain combinations of scripts and datasets. Figure 1 shows execution times when optimizations are applied for all scripts. Overall in our experiments, executing scripts in parallel and applying all optimization rules cuts execution time down to 25 % to 30 % of the time required to execute scripts sequentially.

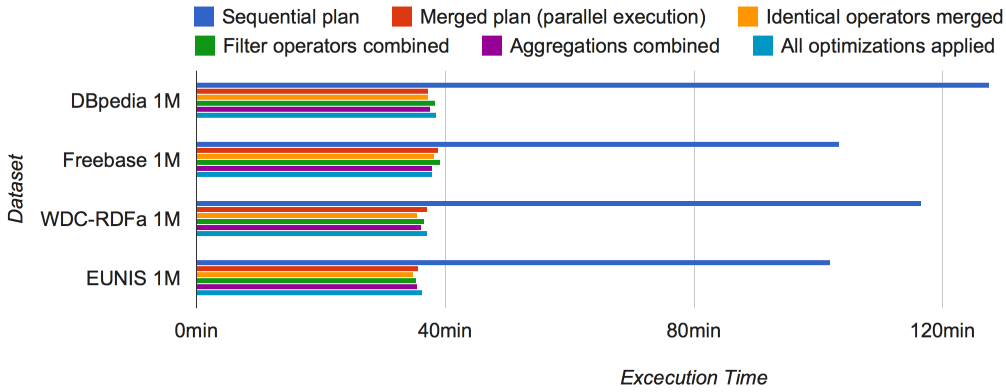


Figure 1: Execution time for all optimizations (52 scripts).

4 Uniqueness, Density, and Keyness of Data

As Linked Datasets are usually sparsely populated, minimal unique property combinations (key candidates) often consist of either multiple low-density properties or cannot be found at all. Novel property attributes, such as the uniqueness, density, and keyness of a property are needed to discover the set of properties that likely identifies an entity, the key candidates. Furthermore, since ontologies are topically clustered by their underlying ontologies, these attributes can be determined per clus-

ter and give some detailed insights into the properties that serve as key candidates per topic.

A Linked Dataset's class hierarchy is the taxonomy defined by its ontology and therein the rdfs:subClassOf relations between the classes. A cluster C_c for a class c consists of all the entities e that are of rdf:type c, which includes all subclasses of c.

$$C_c = \{e | e \xrightarrow{rdf:type} c\}$$

Clusters can contain entities e that are not in any of its subclusters d. We cluster these entities separately and call the resulting clusters *unspecialized clusters*, denoted as C'_c.

$$C'_c = C_c \setminus \{e \mid e \xrightarrow{rdf:type} d,\ d \xrightarrow{rdfs:subClassOf} c\}$$

We omit the c subscript where it is irrelevant in the context. As an additional complication, properties on the Web of Data can have multiple property values. E.g., in the DBpedia dataset we find the following four values for the property dbpedia:birthPlace for the entity of Albert Einstein:

```
dbpedia:Albert_Einstein dbpedia:birthPlace dbpedia:Ulm,
        dbpedia:Kingdom_of_Wuerttemberg,
        dbpedia:German_Empire
        dbpedia:Baden-Wuerttemberg .
```

We denote the set of property values of an entity e and property p as $V(e, p)$. To count the number of entities in a cluster C that have at least one value for p, we define $V(C, p) = \{e \mid |V(e, p)| > 0, e \in C\}$. Property values of a property p and two entities $e1$ and $e2$ are equal if $V(e1, p) = V(e2, p)$, i.e., if the two sets are identical. With this definition we further define the *set of unique value sets* as $V_{uq}(C, p) = \{V(e, p) \mid e \in C\}$.

We are now ready to define the three attributes, uniqueness, density, and keyness, of a property. The *uniqueness uq* of a property p for a cluster C is the number of unique value sets $V_{uq}(C, p)$ per number of total value sets $V(C, p)$ for the given property.

$$\textbf{Uniqueness:} \qquad uq(C, p) = \frac{|V_{uq}(C, p)|}{|V(C, p)|} \tag{1}$$

The *density d* of a property p for a cluster C is the ratio of entities in C that have p to the overall number of entities in C.

$$\textbf{Density:} \qquad d(C, p) = \frac{|V(C, p)|}{|C|} \tag{2}$$

We call a property *full key candidate* if its density and uniqueness are both 1. For cases where they are not both 1 we define its keyness as a useful attribute. The *keyness k* of a property p for a cluster C is the harmonic mean of its uniqueness and density. The harmonic mean emphasizes that *both* parameters must be high to achieve an overall high keyness:

$$\textbf{Keyness:} \qquad k(C, p) = \frac{2 \cdot uq(C, p) \cdot d(C, p)}{uq(C, p) + d(C, p)} \tag{3}$$

We call a property *key candidate* if its keyness is above some threshold.

We investigate the three attributes of an RDF property, uniqueness, density, and keyness, for the given cluster types C, and C'. Determining uniqueness, density, and keyness for a property p in a cluster C_c requires analyzing *all* property value sets for *all* entities in the given cluster. We observe all kinds of specificities of properties for clusters and their subclusters that allow for a fine-grained, cluster-based retrieval of key candidates.

Our evaluation shows that the property keyness can help discovering key candidates for Linked Datasets. It also highlights the advantages of analyzing the class hierarchy in order to observe property behavior for classes along it and make better choices when identifying key candidates for specific classes.

5 Graph Structures in Linked Datasets

Graph patterns are of interest to many communities, e.g. to understand protein structures, to analyse network traffic, to support crime detection, to model object-oriented data, and to query RDF data.

We propose GraphLOD, a system for general-purpose explorative and visual research on Linked Dataset graphs to find frequent graph patterns, common graph patterns for specific schema classes, and we give a uniform definition of interesting patterns for Linked Datasets.

Frequent pattern mining is the essence of graph mining. The objective is to extract all the frequent subgraphs (patterns, motifs) in a given graph, whose occurrence counts are above a specified threshold. If there are common graph patterns amongst multiple and many Linked Datasets, we can define a core set of graph patterns for the Web of Data. Furthermore, we are interested in common patterns for specific classes and class combinations.

Having these basic graph patterns at hand allows for various data management tasks, namely data amendment, data cleansing, neighborhood prediction, and data integration.

A number of frequent subgraph mining algorithms has been introduced since the 1990s, the most popular ones being gSpan [14] and GraMi [4]. While these are useful in certain application domains, such as the life sciences, the graph patterns we mined from Linked Datasets with those methods proved to be neither representative nor interesting as our evaluation shows later. We discuss both algorithms in more detail as related work in the next section. Jiang et al. already note the absence of a pattern mining approach for "compact and meaningful set of frequent subgraphs instead of a complete set of frequent subgraphs [...] There is no clear understanding of what kind of frequent subgraphs are the most compact and representative for any given application" [9]. In fact, for our use case of analyzing Linked Datasets, we observe re-occurring motifs across the datasets that often show a high similarity but are not necessarily isomorphic. While there are e.g. 13,576 windmill graphs in DBpedia, their size ranges between 5 and 453.

Due to the different nature of Linked Datasets and the incompatibility of existing pattern mining algorithms, we are defining a core set of patterns based on patterns that we originally observed in the easy-to-visualize satellite components of the evaluated datasets.

To not only mine the data but to also allow users to interactively explore the results, we have significantly extended our prototype ProLod++, which features many basic as well as specific profiling tasks for a given Rdf dataset, and allows easy extension by further techniques. It is available at http://prolod.org. We implemented and added the GraphLod library, which provides the following functionality:

- Basic graph statistics, such as the number of connected components and strongly connected components, their corresponding diameter, chromatic number, and node degree distribution.
- Visualization of connected components and their isomorphic groups.
- Three graph pattern mining algorithms.
- Visualization of frequent graph patterns with class coloring and their color-isomorphic groups.
- Interactive graph structure exploration in a faceted fashion.

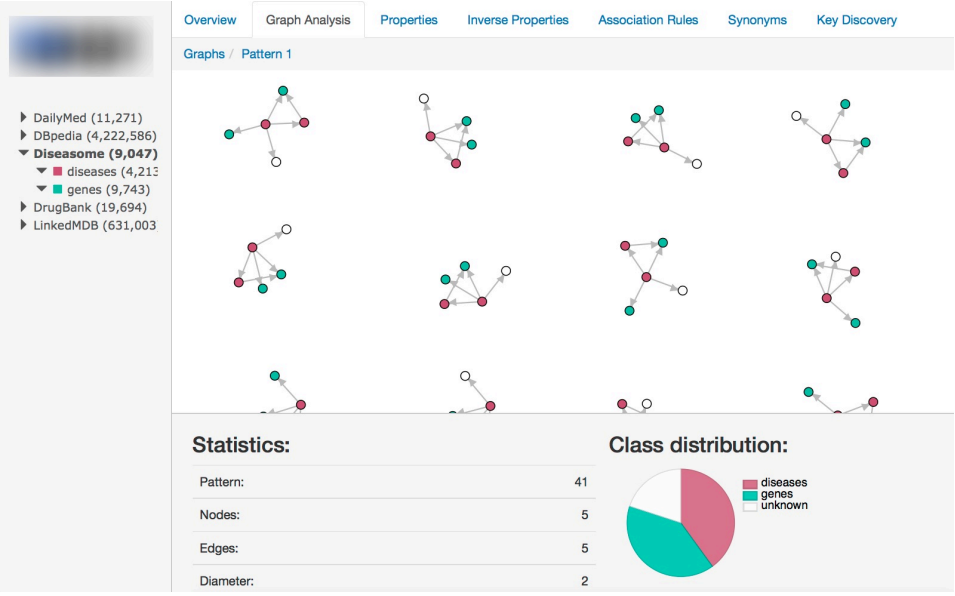


Figure 2: Class-colored instances of a pattern (antenna) in Diseasome visualized by ProLod++.

Figure 2 is a screenshot of ProLod++ showing all occurrences of a selected pattern and their class distribution along with some statistical information.

Our approach to first define the compact pattern and then mine for the exact as well as larger representations takes into account the variability and repetition that occur in Linked Datasets. Grouping the patterns by their class combinations allows for a more fine-granular analysis of the underlying patterns in the data and patterns that are specific for certain class combinations.

Our evaluation on five representative Linked Datasets reveals a high number of pattern occurrences among all of them. We show how graph patterns can support various data management tasks. Furthermore, we discuss how graph pattern mining can highlight the benefits of dataset integration.

6 Reflections and Conclusion

The main difference in my approach with existing work on Linked Data profiling is to address the shortcomings mentioned in section 2, in particular gathering comprehensive metadata in an efficient way. Within my research I am building on existing profiling techniques for relational data and adapting them according to the different nature of Linked Datasets.

This paper has presented the outline and preliminary results of my doctoral research, in which I am focussing on profiling the Web of Data.

We have specified and implemented a comprehensive set of Linked Data profiling tasks and illustrated the Web of Data's diversity with the results for four different Linked Datasets. Furthermore we introduced three common techniques for improving performance of Linked Data profiling and implemented three multi-query optimization rules, reducing profiling task runtimes by 70 %.

We have introduced the concept of keyness (and therein uniqueness and density) of a property to address the sparsity on the Web of Data and thus create the possibility to find key candidates where traditional approaches fail. Our approach has been implemented in ProLod++ and provides users with the uniqueness, density, and keyness for all properties. Having these profiling results at hand helps users in finding key candidates and analyzing the relevance of properties along class hierarchies in Linked Datasets.

We presented the GraphLod extension for ProLod++, which offers Rdf *graph analysis* features. It allows for interactively exploring the graphical structures of Linked Datasets by visualizing the connected components and the graph patterns mined from them. Furthermore it offers basic graph statistics as node degree distribution, pattern diameter, and more. Furthermore we defined a set of graph patterns that can be considered the core of most Linked Datasets.

References

[1] Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. "Mining and Profiling RDF Data with ProLOD++". In: *Proceedings of the International Conference on Data Engineering (ICDE)*. Demo. 2014.

[2] S. Auer, J. Demter, M. Martin, and J. Lehmann. "LODStats – an extensible framework for high-performance dataset analytics". In: *Proceedings of the Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*. 2012.

[3] C. Böhm, J. Lorey, and F. Naumann. "Creating VoiD Descriptions for Web-scale Data". In: *Journal of Web Semantics* 9.3 (2011), pages 339–345.

[4] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. "GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph". In: *PVLDB* 7.7 (2014), pages 517–528. ISSN: 2150-8097.

[5] B. Forchhammer, A. Jentzsch, and F. Naumann. "LODOP – Multi-Query Optimization for Linked Data Profiling Queries". In: *ESWC Workshop on Profiling & Federated Search for Linked Data (PROFILES)*. 2014.

[6] A. Gates, J. Dai, and T. Nair. "Apache Pig's Optimizer". In: *IEEE Data Engineering Bulletin* 35.1 (2013), pages 34–45.

[7] A. Jentzsch, C. Dullweber, P. Troiano, and F. Naumann. "Exploring Linked Data Graph Structures". In: *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015*. 2015.

[8] A. Jentzsch, H. Mühleisen, and F. Naumann. "Uniqueness, Density, and Keyness: Exploring Class Hierarchies". In: *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD 2015)*. 2015.

[9] C. Jiang, F. Coenen, and M. Zito. "A survey of frequent subgraph mining algorithms". In: *The Knowledge Engineering Review* 28.1 (2013), pages 75–105. ISSN: 1877-0509. DOI: 10.1017/S0269888912000331.

[10] S. Khatchadourian and M. P. Consens. "ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud". In: *Proceedings of the Extended Semantic Web Conference (ESWC)*. Heraklion, Greece, 2010. ISBN: 978-3-642-13489-0.

[11] A. Langegger and W. Wöß. "RDFStats – An Extensible RDF Statistics Generator and Library". In: *Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA)*. Los Alamitos, CA, USA, 2009, pages 79–83.

[12] H. Li. "Data Profiling for Semantic Web Data". In: *Proceedings of the International Conference on Web Information Systems and Mining (WISM)*. 2012. ISBN: 978-3-642-33469-6.

[13] T. K. Sellis. "Multiple-query optimization". In: *ACM Transactions on Database Systems (TODS)* 13.1 (1988), pages 23–52. ISSN: 0362-5915.

[14] X. Yan and J. Han. "gSpan: Graph-Based Substructure Pattern Mining". In: *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. 2002, pages 721–724.

Creating Structurally Sound Truss Structures on Desktop 3D Printers

Robert Kovacs

Human Computer Interaction group
Hasso-Plattner-Institut
robert.kovacs@hpi.uni-potsdam.de

We present TrussFab, an integrated end-to-end system that allows users to fabricate large scale structures that are sturdy enough to carry human weight. TrussFab achieves the large scale by complementing 3D print with plastic bottles. It does not use these bottles as "bricks" though, but as beams that form structurally sound node-link structures, also known as trusses, allowing it to handle the forces resulting from scale and load. TrussFab embodies the required engineering knowledge, allowing non-engineers to design such structures and to validate their design using integrated structural analysis. We have used TrussFab to design and fabricate tables and chairs, a 2.5 m long bridge strong enough to carry a human, a functional boat that seats two, and a 5 m diameter dome.

1 Introduction

Personal fabrication tools, such as 3D printers have become popular in HCI, where they have been used for fast prototyping [16] as well as to fabricate interactive objects [10], optical elements [26], or kinematic characters [4]. Since 3D printers today are available in a desktop form factor, they have been able to spread to the maker community [23] and are now increasingly reaching the consumer market [20].

In contrast, the fabrication of large objects still has remained a privilege of industry, which has access to specialized equipment, such as concrete printers that allow making houses [12] or robotic arms capable of 3D printing [17]. The owners of the widespread desktop devices, in contrast, cannot participate in this evolution, because the underlying technology does not scale. Even techniques that break down large models into printer sized parts [14] ultimately do not scale, as large models consume material and time proportional to their size, which quickly renders 3D printing and related techniques intractable for larger-than-desktop-scale models.

As an alternative approach, fabrication enthusiasts have created large objects by combining 3D print with ready-made objects, such as plastic bottles [9]. In their simplest form, such objects wrapped in 3D print can serve as 3D voxel collages that approximate the volume of an object [28].

Going larger, however, is not only about scale and print volume. For large objects, the main design objective is typically to withstand large forces, as forces grow cubed with the size of the object. Also, large objects afford substantial external loads; furniture, bridges, and vehicles, for example, all must be engineered to hold the weight of a human. Designing for large forces, however, requires substantial engineering

skill [6] from envisioning appropriate structures in the first place to verifying their structural integrity.

In this paper, we present TrussFab, an integrated end-to-end system that allows users to design large structures that are sturdy enough to carry human weight (Figure 1). TrussFab achieves this by taking a different perspective on bottles. Unlike previous systems that stacked bottles as if they were "bricks", TrussFab considers them as beams and uses them to form structurally sound node link structures based on closed triangles, also known as trusses. TrussFab embodies the required engineering knowledge, allowing non-engineers to design such structures. TrussFab also allows users to validate their designs using integrated structural analysis (Figure 4). Our main contribution is this end-to-end system.

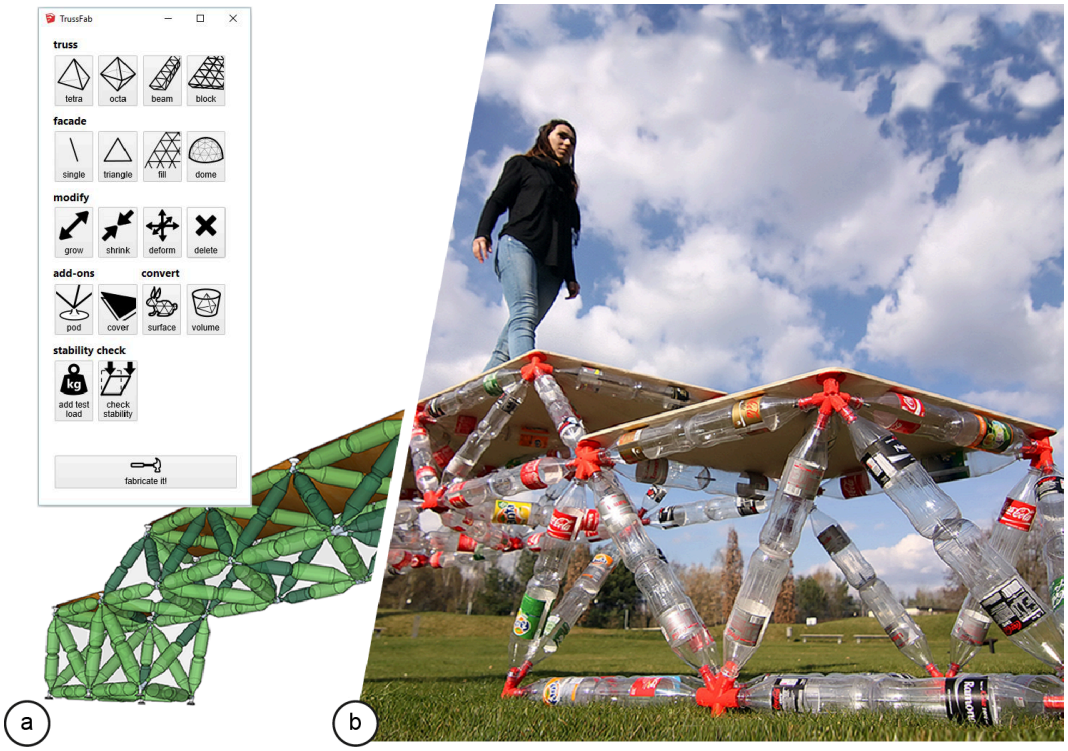


Figure 1: (a) TrussFab's editor allows users to fabricate large structures sturdy enough to carry human weight. (b) TrussFab considers bottles as beams that form structurally sound node-link structures also known as trusses, allowing it to handle the forces resulting from scale and load.

2 Related Work

TrussFab builds on previous efforts in the following branches: large-scale personal fabrication, design with existing objects and construction kits, and tools for creating structurally sound objects.

2.1 Large-scale personal fabrication

Architects and engineers have made efforts to scale up the additive manufacturing process for constructing large-scale structures, such as houses or sculptures. These efforts mostly involve a scaled-up version of the common machinery, like concrete printers [12], or breaking down the objects into smaller parts to print on desktop machines [14]. Another approach to fabricating architectural-scale objects is the use of mobile printing robots, which move on the ground [11] or fly [27] around the printed object. Yet another approach is to use human assisted devices, such as Protopiper [1]. Similarly, Yoshida et al. [29] proposed a computer-assisted fabrication method for large-scale architecture that combines a handheld chopstick dispenser with a projector-based guiding system.

2.2 Construction kits

Construction kits are popular for fast prototyping and fabrication. They offer a repertoire of prefabricated elements which can be combined in various ways. Henrik and Kobbelt [30] developed a system to accurately approximate complex shapes using the Zometool mathematical modeling kit. Skouras et al. [21] created an interactive editor to computationally combine interlocking elements into a desired shape.

2.3 Designing with ready-made objects

MixFab [25], and Encore [3] allow users to integrate existing objects into their design. For creating objects enclosing electronic components Ashbrook et al. [2] developed an augmented fabrication system. Devendorf and Ryokai [5] proposed a human-assisted fabrication system that helps users incorporate everyday objects into 3D print. Gellért assembled wooden boards combined with 3D printed connectors in node-link structure [8]. Combining carbon tubes with 3D printed metal has been proposed for creating functional cars [6]. Skilled individuals have stacked or tied plastic bottles in order to make art pieces, furniture, rafts or houses [9]. Yamada et al. [29] proposed a system for arranging ready-made objects into 3D shapes using 3D printed connectors.

2.4 Tools for creating structurally sound objects

Smith et al. [22] developed a system that automatically generates truss structures using nonlinear optimization. Makris et al. [15] proposed a design tool that generates parametrically defined, semi-automatically analyzed, and visualized structures.

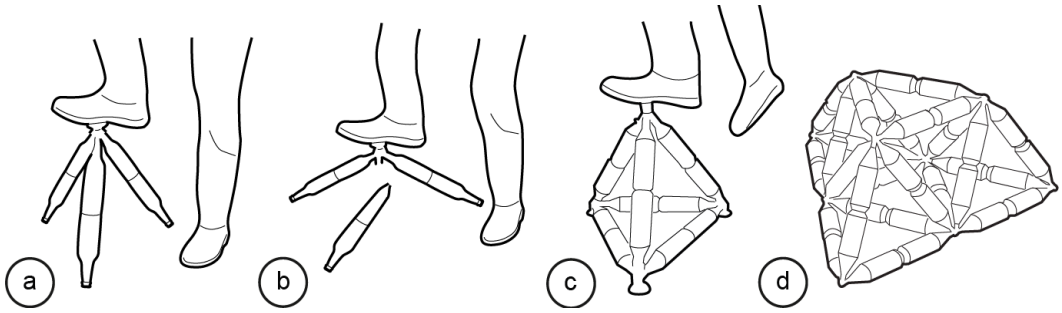


Figure 2: (a) Large objects involve large levers, causing them (b) to break under load. (c) TrussFab instead affords structures based on closed triangles, here forming a tetrahedron. Such structures are particularly sturdy. (d) TrussFab extends this concept to tetrahedron-octahedron trusses of arbitrary size.

Wang et al. [24] developed an automated method that minimizes material cost by converting solid 3D models into a skin-frame structure. SketchChair [19] is an interactive chair design system that allows users to validate the structural integrity of their design by subjecting it to the weight of a human rag doll.

3 Creating structurally sound structures using TrussFab

The key ideas behind TrussFab are (1) to employ bottles in their structurally most sturdy way, i.e., as beams from bottom to bottleneck and (2) to afford sturdy "closed frame structures", also known as trusses [13]. While freestanding bottles tend to break easily (Figure 2a/b), truss structures essentially consist of triangles. In such an arrangement, it is the structure that prevents deformation, not the individual bottle. The main strength of trusses is that they turn lateral forces (aka bending moments) into tension and compression forces along the length of the edges (aka members). Bottles make great members: while they buckle easily when pushed from the side, they are very strong when pushed or pulled along their main axis. (c) The resulting structures, such as this tetrahedron, are strong enough to bear the weight of one or more humans. (d) TrussFab affords building trusses by combining tetrahedra and octahedra into so-called tetrahedral honeycomb structures.

The main design rationale behind the TrussFab editor is to afford this kind of stable structures. We achieve this by starting any design with primitives that already are miniature trusses, i.e., tetrahedra and octahedra; additional functions then allow users to extend and tweak the structure while maintaining the truss property at all times. Once the main truss structure has been created, users may add facades and decorative details.

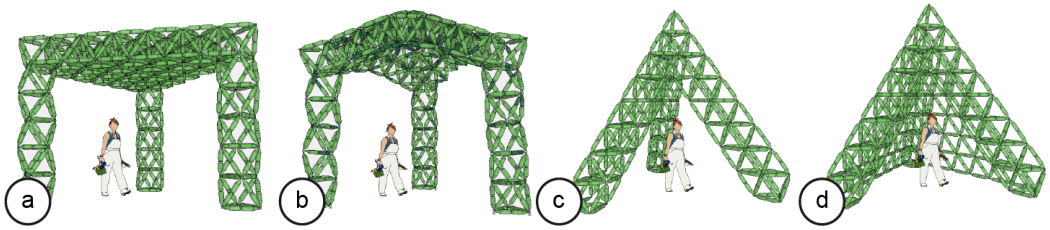


Figure 3: (a) Pavilion created using the beam and block tools. (b) The roof is freely deformed by pulling upwards using the deform tool. (c) The load-bearing structure of the tipi is best made from trusses; (d) its sides can be filled using single layer facades.

3.1 Editing truss structures efficiently

To create larger objects, TrussFab offers a number of tools that create larger trusses in a single interaction, thus resulting in a more efficient design process. The beam tool creates entire beams in one go. The legs of the pavilion in Figure 3a were created this way. The block tool creates a tetra-octa plane in one go. We used it to create the roof of the pavilion in Figure 3a. It can also serve, for example, as a stage.

The deform tool allows users to deform trusses. In Figure 3b we applied this tool in order to obtain a curved roof. Using the tool, we grabbed a hub located in the middle of the roof and dragged it upwards. The tool accommodates this by growing and shrinking members throughout the truss (see section "Implementation").

To support building with layered architecture, TrussFab complements its truss tools with specialized facade tools. The editor offers facade tools for filling large opening with facades. Figure 3c/d illustrates this. Users start by creating a load-bearing structure in the form of trusses (Figure 3c). Users then fill in the non-load-bearing sides as facades (Figure 3d). The benefit of this two-stage process is that facades are particularly efficient. First, they are single-layer, thus require fewer bottles. Second, the hubs that form a facade are flat; this allows TrussFab to fabricate such hubs using a laser cutter, which is very fast (40× faster than 3D print).

3.2 Verifying stability

To demonstrate how TrussFab verifies the stability of the created objects, we use the example of a chair design (Figure 4). First, TrussFab checks if the created structures are structurally sound, TrussFab is checking if there are parts that are not completely locked in place by other members and are subject to shearing forces. If found, the software suggests placing additional stabilizing members. Our chair, however, is rigid, so there are no warnings. For detailed description of the algorithm, see section "Rigidity check".

Second, to verify if the created structures can withstand the desired load, we select the add weight tool and place virtual load on the impact points. Figure 4 illustrates this on the example of a chair. We add 25 kg weight on each of the three corners of

the sitting plate. A click at the backrest adds another 10 kg pushing load into the backrest. Clicking the check stability icon causes TrussFab to compute the effect of these weights onto the structure using finite element analysis. As show in Figure 4a, TrussFab shades all members accordingly. The six vertical members of the octahedron now appear in shades of red, suggesting that these are experiencing compression. So does the chair's "backbone". All other members are tinted blue, suggesting that these are subject to tension. TrussFab compares these forces with the maximal load members and hubs can hold. It warns the user if the limits are exceeded. This is not the case here, so we now know that our chair model is structurally sound.

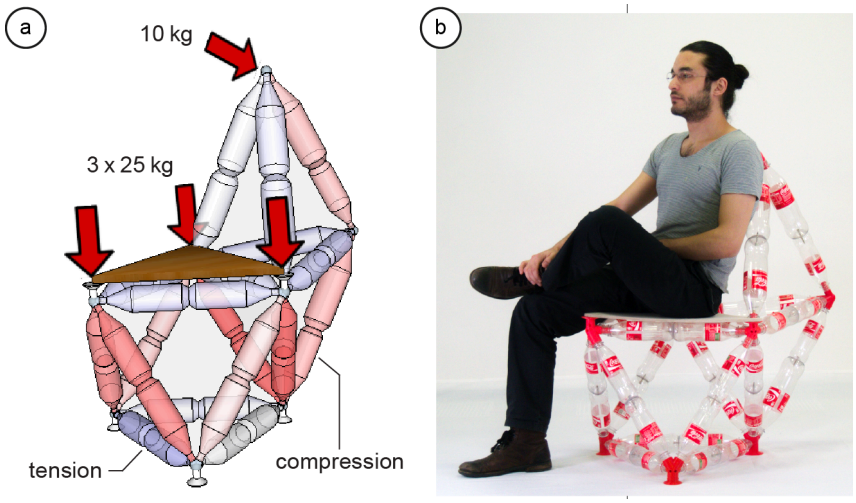


Figure 4: (a) To verify the chair's structural stability we add the load forces expected during use. The system now calculates internal compression and tension forces. Here, no forces are exceeding the limit; thus no warnings are displayed. (b) As predicted, the fabricated chair holds the human weight.

4 Implementation

4.1 TrussFab editor

We implemented TrussFab as an extension to the 3D editor SketchUp. It is written in Ruby and JavaScript. It allows users to create 3D models, verify stability, and to trigger the TrussFab Hub Generator. The grow and shrink tools affect the lengths of members and consequently the angles between members. TrussFab restores the consistency of the 3D model by running a dynamic relaxation algorithm, i.e., neighboring members start to push-pull each other until they find the position that accommodates the change. TrussFab iterates up to 10,000 cycles or until 0.1 mm accuracy has been reached.

4.2 Finite element analysis

TrussFab uses karamba3D[1] as its finite element engine [7]. This method models each edge as a spring of particular stiffness and calculates the displacement of the nodes under the given force. TrussFab treats all hubs as ball joints, allowing for deformations without breaking. The bottle members are modelled as filled cylinders, which are rigid in shear. The pods touching the ground are considered anchor points. TrussFab sends the geometry of the model together with the specified load forces to Karamba3D in JSON format, which returns the resulting compression and tension forces for each member.

4.3 Rigidity check

To check rigidity [18], TrussFab represents the 3D model as a node-link diagram. From this graph G TrussFab forms a rigidity matrix. If the rank of this matrix is equal $3n - 6$ where n is the number of vertices in G TrussFab considers the structure rigid. To shortly explain this, consider a movement of the vertices given by specifying a velocity $_i(t)$ for each vertex v_i at every point in time t. Let p_i be the initial position of v_i. Then the movement preserves the length of an edge v_iv_j, if and only if

$$(_i(t) - _j(t))(p_i - p_j) = 0$$

holds for every point in time. Thus, to check G for rigidity, we can instead test whether velocities satisfying this equation for every edge exist. As each equation is linear, we obtain a system of linear equations. This system can be written as $A = 0$ where μ is the vector of all velocities and each row of the matrix A corresponds to one equation. The matrix A is the above mentioned rigidity matrix. Note that has dimension $3n$ as we have one velocity for each vertex and each velocity is 3-dimensional. Thus, if $rank(A) = 3n - 6$ then the solution space of $A = 0$ is 6-dimensional, which covers exactly the trivial movements of rotating (in three dimensions) and translating (in three dimensions) the whole graph. Hence, if $rank(A) = 3n - 6$, no other edge-length preserving movement can exist, i.e. G is rigid.

5 Conclusion

TrussFab is an integrated end-to-end system that allows users to fabricate large structures that are sturdy enough to carry human weight on desktop 3D printers. Unlike previous systems that built on up-cycled plastic bottles combined with 3D print, TrussFab considers bottles not as “bricks”, but as beams that form structurally sound node link structures also known as trusses, allowing users to handle the forces resulting from scale and load. TrussFab embodies the required engineering knowledge, allowing non-engineers to design such structures and allows users to

[1] http://karamba3d.com (last accessed 2016-10-20).

validate their designs using integrated structural analysis. In the future, we plan on creating kinematic machines based on TrussFab structures.

References

[1] H. Agrawal, U. Umapathi, R. Kovacs, J. Frohnhofen, H.-T. Chen, S. Mueller, and P. Baudisch. "Protopiper: Physically Sketching Room-Sized Objects at Actual Scale". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST '15. Daegu, Kyungpook, Republic of Korea: ACM, 2015, pages 427–436. DOI: 10.1145/2807442.2807505.

[2] D. Ashbrook, S. S. Guo, and A. Lambie. "Towards Augmented Fabrication: Combining Fabricated and Existing Objects". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. Santa Clara, California, USA: ACM, 2016, pages 1510–1518. DOI: 10.1145/2851581.2892509.

[3] X. '. Chen, S. Coros, J. Mankoff, and S. E. Hudson. "Encore: 3D Printed Augmentation of Everyday Objects with Printed-Over, Affixed and Interlocked Attachments". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. UIST '15. Daegu, Kyungpook, Republic of Korea: ACM, 2015, pages 73–82. DOI: 10.1145/2807442.2807498.

[4] S. Coros, B. Thomaszewski, G. Noris, S. Sueda, M. Forberg, R. W. Sumner, W. Matusik, and B. Bickel. "Computational Design of Mechanical Characters". In: *ACM Trans. Graph.* 32.4 (July 2013), 83:1–83:12. DOI: 10.1145/2461912.2461953.

[5] L. Devendorf and K. Ryokai. "Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: ACM, 2015, pages 2477–2486. ISBN: 978-1-4503-3145-6. DOI: 10.1145/2702123.2702547.

[6] *Divergent3D: The World First 3D Printed Super Car.* URL: http://www.divergent3d.com/ (last accessed 2016-10-01).

[7] J. Fish and T. Belytschko. *A first course in finite elements*. John Wiley & Sons, 2007. ISBN: 0-470-03580-3.

[8] O. Gellért. *Print To Build, 3D printed joint collection.* URL: https://www.behance.net/gallery/27812109/Print-To-Build-3D-printed-joint-collection (last accessed 2016-10-01).

[9] *Homes Made from Plastic Bottles.* URL: http://www.inspirationgreen.com/plastic-bottle-homes (last accessed 2016-10-01).

[10] S. E. Hudson. "Printing teddy bears: a technique for 3D printing of soft interactive objects". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2014, pages 459–468. DOI: 10.1145/2556288.2557338.

[11] S. Jokic, P. Novikov, S. Maggs, D. Sadan, S. Jin, and C. Nan. "Robotic positioning device for three-dimensional printing". In: *arXiv preprint arXiv:1406.3400* (2014).

[12] B. Khoshnevis. "Automated construction by contour crafting-related robotics and information technologies". In: *Automation in construction* 13.1 (2004), pages 5–19. DOI: 10.1016/j.autcon.2003.08.012.

[13] T. T. Lan. "Space frame structures". In: *Handbook of Structural Engineering, CRC Press, Boca Raton, FL* (2005), pages 24–1.

[14] M. Lau, A. Ohgawara, J. Mitani, and T. Igarashi. "Converting 3D furniture models to fabricatable parts and connectors". In: *ACM Transactions on Graphics (TOG)*. Volume 30. 4. ACM. 2011, page 85.

[15] M. Makris, D. Gerber, A. Carlson, and D. Noble. "Informing Design through Parametric Integrated Structural Simulation: Iterative structural feedback for design decision support of complex trusses". In: *eCAADe 2013: Computation and Performance – Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe, Delft, The Netherlands, September 18-20, 2013*. Faculty of Architecture, Delft University of Technology; eCAADe (Education and research in Computer Aided Architectural Design in Europe). 2013.

[16] S. Mueller, S. Im, S. Gurevich, A. Teibrich, L. Pfisterer, F. Guimbretière, and P. Baudisch. "WirePrint: 3D printed previews for fast prototyping". In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM. 2014, pages 273–280. DOI: 10.1145/2642918.2647359.

[17] P. Novikov and S. Jokić. *Mataerial.* URL: http://mataerial.com (last accessed 2016-10-01).

[18] B. Roth. "Rigid and flexible frameworks". In: *The American Mathematical Monthly* 88.1 (1981), pages 6–21.

[19] G. Saul, M. Lau, J. Mitani, and T. Igarashi. "SketchChair: An All-in-one Chair Design System for End Users". In: *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*. TEI '11. Funchal, Portugal: ACM, 2011, pages 73–80. DOI: 10.1145/1935701.1935717.

[20] R. Shewbridge, A. Hurst, and S. K. Kane. "Everyday Making: Identifying Future Uses for 3D Printing in the Home". In: *Proceedings of the 2014 Conference on Designing Interactive Systems*. DIS '14. Vancouver, BC, Canada: ACM, 2014, pages 815–824. DOI: 10.1145/2598510.2598544.

[21] M. Skouras, S. Coros, E. Grinspun, and B. Thomaszewski. "Interactive Surface Design with Interlocking Elements". In: *ACM Trans. Graph.* 34.6 (Oct. 2015), 224:1–224:7. DOI: 10.1145/2816795.2818128.

[22] J. Smith, J. Hodgins, I. Oppenheim, and A. Witkin. "Creating Models of Truss Structures with Optimization". In: *ACM Trans. Graph.* 21.3 (July 2002), pages 295–301. DOI: 10.1145/566654.566580.

[23] J. G. Tanenbaum, A. M. Williams, A. Desjardins, and K. Tanenbaum. "Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2013, pages 2603–2612. ISBN: 978-1-4503-1899-0.

[24] W. Wang, T. Y. Wang, Z. Yang, L. Liu, X. Tong, W. Tong, J. Deng, F. Chen, and X. Liu. "Cost-effective Printing of 3D Objects with Skin-frame Structures". In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 177:1–177:10. DOI: 10.1145/2508363.2508382.

[25] C. Weichel, M. Lau, D. Kim, N. Villar, and H. W. Gellersen. "MixFab: A Mixed-reality Environment for Personal Fabrication". In: *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*. CHI '14. Toronto, Ontario, Canada: ACM, 2014, pages 3855–3864. DOI: 10.1145/2556288.2557090.

[26] K. Willis, E. Brockmeyer, S. Hudson, and I. Poupyrev. "Printed Optics: 3D Printing of Embedded Optical Elements for Interactive Devices". In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. UIST '12. Cambridge, Massachusetts, USA: ACM, 2012, pages 589–598. DOI: 10.1145/2380116.2380190.

[27] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler. "Aerial robotic construction towards a new field of architectural research". In: *International journal of architectural computing* 10.3 (2012), pages 439–459. DOI: 10.1260/1478-0771.10.3.439.

[28] S. Yamada, H. Morishige, H. Nozaki, M. Ogawa, T. Yonezawa, and H. Tokuda. "ReFabricator: Integrating Everyday Objects for Digital Fabrication". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. Santa Clara, California, USA: ACM, 2016, pages 3804–3807. DOI: 10.1145/2851581.2890237.

[29] H. Yoshida, T. Igarashi, Y. Obuchi, Y. Takami, J. Sato, M. Araki, M. Miki, K. Nagata, K. Sakai, and S. Igarashi. "Architecture-scale Human-assisted Additive Manufacturing". In: *ACM Trans. Graph.* 34.4 (July 2015), 88:1–88:8. DOI: 10.1145/2766951.

[30] H. Zimmer, F. Lafarge, P. Alliez, and L. Kobbelt. "Zometool shape approximation". In: *Graphical Models* 76.5 (2014), pages 390–401. ISSN: 1524-0703.

Theoretical Analyses of Evolutionary Algorithms with a Focus on Estimation of Distribution Algorithms

Martin Krejca

Algorithm Engineering
Hasso-Plattner-Institut
martin.krejca@hpi.de

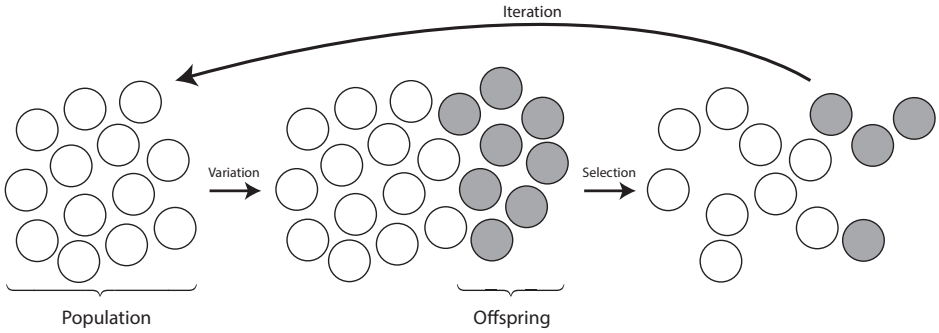
Evolutionary Algorithms (EAs) are randomized search heuristics and general-purpose solvers that mimic behavior seen in natural evolution, such as mutation or crossover. They are most often used, due to their flexibility, when there is little problem-specific knowledge or when there is too little budget to implement or set up highly specific solvers.

Our focus lies on theoretically analyzing EAs to gain insights into their key features, with the aim to answer important questions, such as how to overcome noise or when crossover is beneficial. We especially consider so-called Estimation of Distribution Algorithms (EDAs), which are EAs in a broader sense. EDAs keep a probability distribution over the space of all possible solutions and modify said distribution by an update scheme specific to the algorithm. For this purpose, we introduced a general framework that subsumes many EDAs of interest, trying to state general results – that is, not only run time results – about properties that make these algorithms succeed or fail.

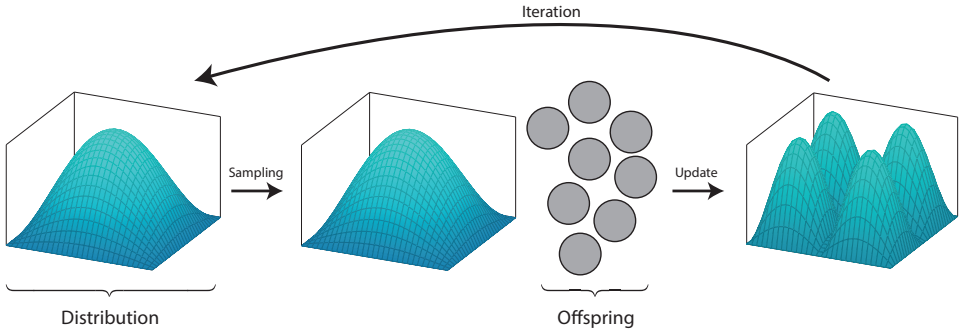
1 Introduction

When facing real-world optimization problems, it is sensible to use state-of-the-art solvers. Due to the NP-hard nature of many of these problems, one does not necessarily expect to get optimal results but a fair approximation. However, if the problem cannot (easily) be transformed into the form that the solver accepts, if there is no good solver for the problem, or if one is interested in a fast (and hopefully good) preliminary solution, one may consider using alternative approaches like randomized search heuristics. One class of such heuristics are Evolutionary Algorithms (EAs) [2], which can produce good solutions in little time. A recent result even shows that an EA can outperform two popular solvers for integer linear programs on an industrial problem containing over 50 000 variables [8].

In the traditional sense, an EA is an algorithm that maintains a (multi-)set (the *population*) of candidate solutions (so-called individuals) over the space of all potential solutions of the problem it should optimize. In an iterative fashion, the EA creates new individuals by modifying the existing ones and then discarding bad individuals, creating a new population. Operations that only modify a single individual are called mutation, and operations that use more than one individual are called crossover (or recombination). The act of choosing the best individuals for the next iteration is referred to as selection. This scheme is also sketched in Figure 1a.



(a) The schematic view of a classical EA. The population first increases through variation (i.e., mutation or crossover) and then gets reduced by selection.



(b) The schematic view of an EDA. The algorithm samples offspring from a distribution and then performs an update.

Figure 1: A comparison of the main differences between classical EAs and EDAs. An EA (Figure 1a) works with an explicit population, whereas an EDA (Figure 1b) uses a probability distribution instead. Both algorithms create offspring from their population and update it afterward.

Now, all that is needed for an EA to work is a problem-specific quality measure that says how good or bad each individual is, i.e., an objective function, also called fitness function in this context. All of the other operators (mutation, crossover, and selection) only depend on the encoding of an individual.[1] This makes EAs vastly applicable, as they are basically problem-agnostic.

In theoretical computer science, when modeling algorithms that have no information about the problem they are optimizing besides the results of the objective function, a black-box model is used. Theoretical analyses then follow black-box complexity, that means, only the number of function evaluations – the supposedly most costly operation – is considered. In this model, it is well known that NP-hard problems can be solved in polynomial time by expensive offline computations that do

[1]Of course, the fitness function can also be problem-independent, but in that case, optimization does not make any sense.

not count toward the complexity [32], even by EAs [9]. In such a case, the function evaluations are not necessarily the most expensive operations. To avoid unrealistic scenarios like these, we always assume that the offline computations of an EA only involve cheap mutation, crossover, and selection operators and nothing else.

One of the fundamental questions in the field of EAs that has been stated since they emerged is how these algorithms benefit from crossover more than just from mutation [19]. Since every algorithm is as bad as any other when looking at a uniform distribution over all problems possible (known as the no free lunch theorem [21]), it makes sense to focus on certain problem classes that appear in real-world scenarios. On the other hand, these classes should be general enough to get widely applicable statements on EAs. This proves to be hard, as the theoretical analysis of EAs is rather challenging. Thus, compromises have to be made.

Usually, in the theoretical analysis of EAs, the solution space is modeled as $\{0,1\}^n$, that is, all bit strings of length n [21]. This way, there exist 2^n different solutions, and a brute force or uninformed search approach would end up with an exponential run time (in expectation). The most common mutation operator is standard bit flip mutation, and the most common crossover operator is uniform crossover. The former takes one individual and flips each bit independently with a probability of $1/n$, that is, it flips 1 bit in expectation. The latter takes two individuals and creates a new one by choosing each bit uniformly at random from one of the input's bit at each position.

The most common functions that are analyzed in theory are OneMax (Equation 1), which represents a simple hill climbing task by yielding the number of 1s in a bit string, and LeadingOnes (Equation 2), which represents the search for a random permutation by yielding the number of leading 1s in a bit string:

$$\text{OneMax}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i \,, \tag{1}$$

$$\text{LeadingOnes}(\boldsymbol{x}) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j \,. \tag{2}$$

Both of these functions have been heavily analyzed in different scenarios (see, e.g., [11, 18, 23] or [3, 10, 12], respectively), and investigating them allows for insights into fundamental properties of EAs that are, for instance, necessary to climb a hill or the cope with dependencies.

Looking at the general EA scheme mentioned earlier, one can abstract and speak of EAs in a wider sense when considering algorithms that store solutions in an arbitrary way, create samples, and then update their storage. When we exchange the population (i.e., the explicit set of solutions) of a classical EA with a probability distribution over $\{0,1\}^n$, we arrive at a class of algorithms called Estimation of Distribution Algorithms (EDAs) (see [20] for a nice survey). The general EDA scheme is depicted in Figure 1b, and the comparison to classical EAs is explained in Figure 1. Our main focus is on theoretically analyzing EDAs, because there are only few results; but there has been some recent interest in the theoretical analysis of EDAs [4, 6, 33].

Since storing an arbitrary probability distribution over $\{0,1\}^n$ means storing up to 2^n different values, allowing for arbitrary distributions in EDAs would potentially result in exponential memory space. Thus, many EDAs only store a vector $\boldsymbol{p} \in \mathbf{R}^n$ of n real numbers, where each component represents the probability to sample a 1 at said position (called incremental, univariate EDAs in [20]). Mathematically speaking, these algorithms maintain a Poisson binomial distributions.

We now proceed by stating some notation in Section 2 that we need to explain our results presented in Section 5. We conclude with an overview on future work in Section 4.

2 Preliminaries

We start by presenting the $(\mu+1)$-GA[2] (Algorithm 1): one of the simplest classical EAs using crossover. This algorithm has a population size of $\mu \in \mathbf{N}^+$ and creates every iteration a single new individual by either mutation or crossover (followed by mutation). Its parameter p_c – the crossover probability – determines which action of the former two to choose when creating a new individual.

The crossover operator used in Algorithm 1 is the uniform crossover, but the name $(\mu+1)$-GA is not exclusive for that crossover operator. In one of our papers [15], we actually consider another kind of crossover, called majority vote crossover, which actually uses three instead of the usual two parents to create a new individual. The offspring takes the bit according to the majority of its parents' bits at each position.

In another paper of ours [16], we consider the $(\mu+1)$-EA, which is the $(\mu+1)$-GA without crossover, i.e., with a crossover probability $p_c = 0$.

As can been seen in line 6 of Algorithm 1, the termination criterion is not specified. In run time analysis, we are interested in the time an algorithm needs to find an optimum for the first time. This means that the termination criterion could also say *until optimum found*. However, since this is a piece of information the algorithm does not have, we assume that it keeps iterating forever, and we just stop all further considerations, once the optimum is found. Formally, we are interested in the expected first hitting time of the algorithm finding an optimum. For that, let $O \subseteq \{0,1\}^n$ denote the set of all optimal solutions for the fitness function the algorithm is optimizing, and let

$$T = \inf\{t \mid P^{(t)} \cap O \neq \emptyset\} ,$$

i.e., the first time that the population contains an individual that is optimal.

Note that T – the first hitting time – itself is a random variable because the $P^{(t)}$ are random variables, due to the random nature of the algorithm. The most valuable piece of information about a random variable is of course its distribution. However, determining the exact distribution of such a complex random variable like T is basically impossible. Therefore, the standard approach is to calculate its expectation, i.e., $\mathrm{E}[T]$. We also call this value the expected run time of a randomized algorithm.

[2] GA stands for Genetic Algorithm, where *genetic* implies the use of crossover.

Algorithm 1: The $(\mu+1)$-GA with crossover probability p_c, maximizing f, using standard bit flip mutation as *mutate*, uniform crossover as *crossover*

```
1  t ← 0
2  P^(t) ← ∅
3  for i ∈ {1, ..., μ} do
4      P^(t) ← P^(t) ∪ {x}, where x ∈ {0,1}^n is chosen uniformly at random (u.a.r.)
5  end
6  while termination criterion not met do
7      p ∈ [0,1] chosen u.a.r.
8      if p < p_c then
9          z ← mutate(crossover(x, y)), where x, y ∈ P^(t) are chosen u.a.r.
10     else
11         z ← mutate(x), where x ∈ P^(t) is chosen u.a.r.
12     end
13     P^(t+1) ← (P^(t) ∪ {z}) ∖ {r}, where r ∈ arg min_{o ∈ P^(t) ∪ {z}} f(o)
14     t ← t + 1
15 end
```

If possible, we extend results to concentration bounds, that is, we bound the probabilities of T not getting too large (or too small) with high probability, where *with high probability* means with a probability of $1 - o(1)$. These bounds serve as a fair approximation of the real distribution of T.

For the EDAs, we introduced a general framework in [14] that captures incremental, univariate EDAs, as already discussed in Section 1. We call this framework the n-Bernoulli-λ-EDA (Algorithm 2). It maintains a Poisson binomial distribution via a so-called frequency vector $\boldsymbol{p}$. Every iteration, it samples λ offspring according to $\boldsymbol{p}$ and then updates its frequency vector with respect to an algorithm-specific update scheme φ. This update scheme is at the core of the framework, as it can be an arbitrary function (with the interface seen in line 9). The idea behind φ is that it takes the old frequency vector as well as all of the sampled offspring and their respective fitness. It then yields a new frequency vector. Our main interest lies in analyzing this framework, that is, we want to understand which properties of φ lead to which behavior of the n-Bernoulli-λ-EDA. For this, we are trying to be as general as possible.

3 Results

We are not going to discuss our results on noise [34, 35] that we already presented in great detail in last year's report [30], but we do have published another paper in that context [16]. In contrast to our previous papers on noise, this paper exclusively considers the $(\mu+1)$-EA.

Algorithm 2: n-Bernoulli-λ-EDA with a given update scheme φ, optimizing f

$t \leftarrow 0$
foreach $i \in \{1, \dots, n\}$ **do** $p_i^{(t)} \leftarrow \frac{1}{2}$
repeat
  $D \leftarrow \emptyset$
  for $j \in \{1, \dots, \lambda\}$ **do**
    $x \leftarrow$ offspring sampled with respect to $p^{(t)}$
    $D \leftarrow D \cup \{x\}$
  end
  $p^{(t+1)} \leftarrow \varphi\big(p^{(t)}, \big(x, f(x)\big)_{x \in D}\big)$
  $t \leftarrow t + 1$
until termination criterion met

There are already some results on population-based EAs under noise, e.g., [17, 31], but none of them aim at showing differences between noise settings. Our paper considers two general noise models: a more general version of a uniform distribution and a distribution that declines exponentially from its mean. The idea is that the former noise model is lower-bounded (it basically has an infinitely steep tail), whereas the latter has a tail that is exponentially steep. We show that this difference of steepness in the tail determines whether the algorithm is successful or not, i.e., whether its expected run time is polynomial or superpolynomial with high probability. The $(\mu + 1)$-EA is able to optimize OneMax under the first noise model if the population size is large enough. It can do so in polynomial time if the variance of the noise is at most a polynomial (but maybe far larger than the actual dimension size) and without knowing the variance! The algorithm, however, fails under the second noise model, once the variance is in orders of $\omega(n^2)$, i.e., once the noise dominates the signal from the fitness. Unfortunately, our results only hold for OneMax so far. Hence, a generalization would desirable.

Our Results on classical EAs [5, 7, 15] all involve crossover but are mainly about population diversity. In [5], we look at the $(\mu + 1)$-GA, but we add different so-called diversity mechanisms to line 13 in Algorithm 1. These mechanisms do not delete a *random* individual with worst fitness but one with criteria specific to the mechanism, e.g., we prioritize deleting individuals that occur multiple times in the population. Our function of interest is a function similar to OneMax but with a plateau of local optima in-between: Jump [22]. We do consider the expected run time of the $(\mu + 1)$-GA using different diversity mechanisms. Of course, there are already some results on the $(\mu + 1)$-GA on Jump, e.g., [22, 24], but they use a very low crossover probability p_c, leaving all of the diversity work to the mutation operator. Our work is the first in that area that uses a realistic crossover probability of a constant less than 1 for many different diversity operators. This way, we show what kind of speed-up can be achieved using different approaches to selecting individuals when stuck in local optima.

Our setting in [7] is similar to the one before, but we use the $(\mu + 1)$-GA as presented in Algorithm 1 this time, that is, we use no explicit diversity operator. Hence, this setting is even closer to [22, 24]. The main difference is that we show how diversity emerges naturally through an interplay of crossover and mutation. As mentioned in the previous paragraph, prior analyzes always chose a very small crossover probability to let mutation do all the work. In our paper, we do not have this restriction and actually benefit from crossover occurring quite often, that is, with constant probability. This approach is a novelty in this kind of setting. Further, we also consider different mutation rates that only differ by a constant factor. For lower mutation rates, this is harmful with respect to the optimization time, whereas higher mutation rates can actually even compete with variants of the $(\mu + 1)$-GA that use explicit diversity mechanisms (as analyzed in [5]). This means that the interplay of crossover and mutation with well-chosen crossover probability and mutation rate leads to the same sufficient diversity like when using explicit external routines to force diversity!

In [15], we also look at the $(\mu + 1)$-GA on Jump, but we use mutation operators that only flip a single bit every time, and we consider the majority vote crossover instead of the uniform crossover; the former has not been theoretically analyzed before. We extend our analyzes to specific VertexCover instances, as they are considered in [13, 26], where the majority vote crossover leads to a significant speed-up when compared to the uniform crossover, because the majority vote crossover easily fixes wrong bit values if they are distributed uniformly at random, i.e., if the majority is more likely to be correct, the result as a whole will be as well. We exploit this observation by providing a general scheme (following probability amplification) for Monte Carlo algorithms with a failure probability of less than $1/2$ that makes use of the majority vote crossover and decreases the failure probability drastically.

The last work we discuss in this report is our paper on EDAs [14]. As already mentioned in Section 2, we introduce the theory-driven n-Bernoulli-λ-EDA framework (Algorithm 2) in this paper. This is not the first EDA framework proposed (other frameworks can be seen, e.g., in [27, 28]), but this is the only one that subsumes all of the (and only the) incremental, univariate EDAs, which are described in [20]. Note that our algorithms considered in our prior noise papers [34, 35] also fall into this framework. In [14], we look at a fundamental property of the n-Bernoulli-λ-EDA: whether the update scheme changes a single probability of the frequency vector $\boldsymbol{p}$ in expectation if there is no signal from the fitness function. Many EDAs (also the ones we considered in our noise papers) do not change such a probability in expectation if there is no clear input. However, we could show that this behavior implies that the probability's variance will increase over time if possible. This leads to the undesired behavior that certain frequencies that do not get a fitness signal for a long time will end up at one of the extreme points, making it hard or even impossible to recover from that. We suggested an alternative approach that was able to overcome this problem, and we proposed an algorithm that optimizes LeadingOnes in $O(n \log n)$ in expectation, whereas many common randomized search heuristics need a time of $\Theta(n^2)$ [1]. Unfortunately, this alternative approach seems to be highly specific. Thus, further research is necessary to investigate the true limits of our results. Also, the

main result is not as general as it can be. We do have a more general version (that applies to random processes in general!), but we did not publish it yet.

We also contributed to two other papers, which are, at present, not published. One of them considers the SSWM algorithm [29] – an algorithm that also accepts worse solutions with a certain probability – in dynamic environments of changing threats. The other paper proves a lower bound of a specific EDA that falls into our framework on OneMax. Unfortunately, this result is not very general, but the proof methods used have just been proposed [33], hence, we hope that we can learn from specific results to end up with a generalization eventually.

4 Future Work

We already mentioned some improvements to our results in Section 5. Our latest paper on noise [16] is very specific up to now. However, there are only a few places that are that specific. Generalizing these would generalize all of the results immediately. This seems to be a reasonable next step.

Regarding our work on crossover, we did run some experiments as well, and we could see that our theoretical bounds are not very tight. Since our proof ideas are novel in that area, this is not surprising. However, improving the theoretical results to match the experiments is of course an ambition we have.

We plan to dedicate most of our efforts to investigating n-Bernoulli-λ-EDAs. Right now, we are working on unbiasedness (as investigated in [25]), which has not been formally defined in the context of these EDAs yet. The idea is to show which properties are necessary and sufficient for such an EDA to not value 1s over 0s or the other way around when looking at its update scheme. Unbiasedness implies invariance of the optimization process with respect to inverting or permuting the bits in the fitness function. That means that the results on unbiased EDAs can be generalized to far greater function classes at no extra cost. Another goal of ours is to get more lower bounds, since that is an area that has not been considered besides [33]. At last, general results on EDAs to noise also seem reasonable. Our prior work [34, 35] hints at some similarities of the different algorithms considered. Finding and exploiting these could lead to a general statement about robustness of EDAs to noise.

References

[1] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. Larsen, and K. Mehlhorn. "The Query Complexity of Finding a Hidden Permutation". In: *Space-Efficient Data Structures, Streams, and Algorithms*. 2013, pages 1–11. ISBN: 978-3-642-40273-9.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. 1996. ISBN: 978-0-19-509971-3.

[3] S. Böttcher, B. Doerr, and F. Neumann. "Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem". In: *Proc. of PPSN*. 2010, pages 1–10. ISBN: 978-3-642-15844-5.

[4] T. Chen, K. Tang, G. Chen, and X. Yao. "Analysis of computational time of simple estimation of distribution algorithms". In: *IEEE TEVC* (2010). ISSN: 1089-778X.

[5] D.-C. Dang, T. Friedrich, M. S. Krejca, T. Kötzing, P. K. Lehre, P. S. Oliveto, D. Sudholt, and A. M. Sutton. "Escaping Local Optima with Diversity Mechanisms and Crossover". In: *Proc. of GECCO*. 2016, pages 645–652. ISBN: 978-1-4503-4206-3.

[6] D.-C. Dang and P. K. Lehre. "Simplified Runtime Analysis of Estimation of Distribution Algorithms". In: *Proc. of GECCO*. 2015, pages 513–518. ISBN: 978-1-4503-3472-3.

[7] D.-C. Dang, P. K. Lehre, T. Friedrich, T. Kötzing, M. S. Krejca, P. S. Oliveto, D. Sudholt, and A. M. Sutton. "Emergence of Diversity and its Benefits for Crossover in Genetic Algorithms". In: *Proc. of PPSN*. 2016, pages 890–900. ISBN: 978-3-319-45823-6.

[8] K. Deb and C. Myburgh. "Breaking the Billion-Variable Barrier in Real-World Optimization Using a Customized Evolutionary Algorithm". In: *Proc. of GECCO*. 2016, pages 653–660. ISBN: 978-1-4503-4206-3.

[9] B. Doerr, C. Doerr, and T. Kötzing. "The unbiased black-box complexity of partition is polynomial". In: *Artificial Intelligence* (2014). ISSN: 0004-3702.

[10] B. Doerr and C. Winzen. "Black-Box Complexity: Breaking the $O(n \log n)$ Barrier of LeadingOnes". In: *Computing Research Repository* (2012).

[11] B. Doerr and C. Winzen. "Memory-restricted black-box complexity of OneMax". In: *Information Processing Letters* (2012). ISSN: 0020-0190.

[12] C. Doerr and J. Lengler. "The (1+1) Elitist Black-Box Complexity of LeadingOnes". In: *Proc. of GECCO*. 2016, pages 1131–1138. ISBN: 978-1-4503-4206-3.

[13] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. "Approximating Covering Problems by Randomized Search Heuristics Using Multi-objective Models". In: *Evolutionary Computation* (2010).

[14] T. Friedrich, T. Kötzing, and M. S. Krejca. "EDAs cannot be Balanced and Stable". In: *Proc. of GECCO*. 2016, pages 1139–1146. ISBN: 978-1-4503-4206-3.

[15] T. Friedrich, T. Kötzing, M. S. Krejca, S. Nallaperuma, F. Neumann, and M. Schirneck. "Fast Building Block Assembly by Majority Vote Crossover". In: *Proc. of GECCO*. 2016, pages 661–668. ISBN: 978-1-4503-4206-3.

[16] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. "Graceful Scaling on Uniform Versus Steep-Tailed Noise". In: *Proc. of PPSN*. 2016, pages 761–770. ISBN: 978-3-319-45823-6.

[17] C. Gießen and T. Kötzing. "Robustness of Populations in Stochastic Environments". In: *Proc. of GECCO*. 2014, pages 1383–1390. ISBN: 978-1-4503-3472-3.

[18] C. Gießen and C. Witt. "Optimal Mutation Rates for the (1+λ) EA on OneMax". In: *Proc. of GECCO*. 2016, pages 1147–1154. ISBN: 978-1-4503-4206-3.

[19] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1989. ISBN: 978-0-201-15767-3.

[20] M. Hauschild and M. Pelikan. "An Introduction and Survey of Estimation of Distribution Algorithms". In: *Swarm and Evolutionary Computation* (2011). ISSN: 2210-6502.

[21] T. Jansen. *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. 2013. ISBN: 978-3-642-17338-7.

[22] T. Jansen and I. Wegener. "The Analysis of Evolutionary Algorithms – A Proof That Crossover Really Can Help". In: *Algorithmica* (2002). ISSN: 1432-0541.

[23] T. Kötzing, A. Lissovoi, and C. Witt. "(1+1) EA on Generalized Dynamic OneMax". In: *Proc. of FOGA*. 2015, pages 40–51. ISBN: 978-1-4503-3434-1.

[24] T. Kötzing, D. Sudholt, and M. Theile. "How Crossover Helps in Pseudo-Boolean Optimization". In: *Proc. of GECCO*. 2011, pages 989–996. ISBN: 978-1-4503-0557-0.

[25] P. K. Lehre and C. Witt. "Black-Box Search by Unbiased Variation". In: *Algorithmica* (2012). ISSN: 1432-0541.

[26] F. Neumann, P. S. Oliveto, G. Rudolph, and D. Sudholt. "On the Effectiveness of Crossover for Migration in Parallel Evolutionary Algorithms". In: *Proc. of GECCO*. 2011, pages 1587–1594. ISBN: 978-1-4503-0557-0.

[27] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen. *Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles*. 2013. arXiv: 1106.3708v3.

[28] T. Paixão, G. Badkobeh, N. H. Barton, D. Çörüs, D.-C. Dang, T. Friedrich, P. K. Lehre, D. Sudholt, A. Sutton, and B. Trubenová. "Toward a unifying framework for evolutionary processes". In: *Journal of Theoretical Biology* (2015), pages 28–43. ISSN: 0022-5193.

[29] T. Paixao, J. Pérez Heredia, D. Sudholt, and B. Trubenova. "First Steps Towards a Runtime Comparison of Natural and Artificial Evolution". In: *Proc. of GECCO*. 2015, pages 1455–1462. ISBN: 978-1-4503-3472-3.

[30] *Proceedings of the 9th Ph.D. retreat of the HPI Research School on service-oriented systems engineering*. 2015. ISBN: 978-3-86956-345-9.

[31] A. Prügel-Bennett, J. Rowe, and J. Shapiro. "Run-Time Analysis of Population-Based Evolutionary Algorithm in Noisy Environments". In: *Proc. of FOGA*. 2015, pages 69–75. ISBN: 978-1-4503-3434-1.

[32] R. Pruim and I. Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. 2005. ISBN: 978-3-540-21045-0.

[33] D. Sudholt and C. Witt. "Update Strength in EDAs and ACO: How to Avoid Genetic Drift". In: *Proc. of GECCO*. 2016, pages 61–68. ISBN: 978-1-4503-4206-3.

[34] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. "Robustness of Ant Colony Optimization to Noise". In: *Proc. of GECCO*. 2015, pages 17–24.

[35] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. "The Benefit of Recombination in Noisy Evolutionary Search". In: *Proc. of ISAAC*. 2015, pages 140–150. ISBN: 978-3-662-48971-0.

Understanding "Bad Code" Using Qualitative Methods

Kateryna Kuksenok

Software Architecture Group
Hasso-Plattner-Institut
Kateryna.Kuksenok@hpi.uni-potsdam.de

In what situations is code that is recognized as "bad", and code production mechanisms that are recognized as producing bad code, kept without improvement, and why? This report describes a study of such code that uses qualitative methods. The concept of "bad code" itself is described and motivated based on a conceptual framework of deliberate individual change. The particular qualitative approach of this study is related to empirical methods in general, and qualitative methods in particular, in software engineering research. An initial characterization connects to literature on *code smells* and *technical debt*, as well as to research in mining code repositories.

1 Introduction

Software development involves many different tools depending on the problem, and spans different social structures depending on the organization. Additionally, the act of writing code is inescapably situated in time. Time is a resource that is pervasively and overwhelmingly lacking; the passage of time reveals new tools and possibilities not visible previously. Literature on debugging, fault localization, and testing aims to improve the means for programmers to write and maintain good code, and to identify and address bad code.

This report argues that there are situations in which code is recognized as "bad" but allowed to persist. Depending on the context of the program and who is making the judgment, "bad" can mean inelegant or inefficient implementation, as well as faulty behavior, wrong output, or security holes. The judgment and reality may not align; something may be considered merely ugly, but in the end result in errors. The aesthetic claims of beauty and ugliness are connected to material judgments through familiarity and expertise [12], and though an expert may decide for good reasons that ugly code is permissible, not everyone who permits ugly code is an expert. Time pressures in real-world projects mean that self-described bad code is inevitable.

Are there kinds of understood bad code that can be made useful? Is there a pattern of programmer behavior that produces bad code easy to identify? Beck and Fowler's 1999 list of "bad smells" in code, for example, is intended to help navigate the difficult task of deciding how and when to refactor code [1], and have since been studied as a means to help programmers write more maintainable and less error-prone code. Some of the code smells "intrinsically characterized by how code elements change over time" so using change history in addition to structural information improves the identification some of the smells with 60 % to 80 % accuracy [15]. An approach for analyzing the evolution of code smells, and their impact and fre-

quency, has also been attempted on a case-study basis [13]. These approaches focus on the code and its history; but can the reflective moment of calling something "bad" be identified and, for example, used as a point from which to prompt to articulate tacit information?

Less visible than bad smells in code are the bad practices that precipitate in such code: actions that are *actively recognized as bad* but not remedied. Section 2 describes the particular case of scientific software development, which was studied in depth in prior dissertation work [10]. That study used a qualitative, ethnographic approach to understand how researchers not formally trained in software engineering were adopting and adapting programming tools. The outcome was a conceptual framework for deliberate individual change, which articulates the antecedents for deciding to implement a better practice, or try a better tool.

Section 3 argues that this qualitative approach, which has considerable precedent in the landscape of empirical research in software engineering, provides an important capability central to the novelty of this study. In particular, this approach supports maintaining both the *emic* (internal) terminology of the informant population and the *etic* (outsider-looking-in) terminology of software engineering. Section 4 introduces a preliminary categorization of "bad code" that is situated in programming as a sociotechnical practice in time, and outlines the next steps. The preliminary categorization uses qualitative methods, and the next steps articulate how to relate this approach to existing work, particularly in mining code repositories for patterns.

2 Deliberate Individual Change

Increasingly, programming and coding is done by individuals who learned the relevant skills piecewise, in a "workshop" or "hackathon" model. In the natural and social sciences, Software Carpentry Workshops are one very popular example of such a workshop, and aim explicitly to teach "best practices" for a growing body of computational tools and skills within those domain disciplines [19]. Much of what is written about scientific software practice emphasizes that the reluctance to adopt better practices results in software that is slower and more error-prone, and with less of the desired capability. Heaton and Carver provide a 2005 systematic literature review of claims about software engineering practices [5].

Less normative perspectives on scientific programming include Kelly's knowledge acquisition model [7]. This model challenges the view of scientific programmers as "end users" of complex tools. Instead, Kelly views the programmer in this context not as following a series of tasks from a software engineering method paradigm, but rather "interacts with the software to fill in gaps in his or her knowledge" and these interactions are driven by the knowledge acquisition needs of the individual or group. This allows Kelly to explain why in scientific teams, individuals have roles tied to their area of scientific interest or expertise, rather than software task, reflecting "cradle to grave" kind of ownership in scientific software. Furthermore, the model explains how a "risk-averse" scientist continually verifies the software through inspection, review, and repeated purposeful testing. In the climate science

context, the use of visualization as a means for consistent testing is described by Easterbrook and Johns as adapting some agile development processes [2].

The following conceptual framework of **deliberate individual change** in programming tools and skills aims to reconcile these three views of the scientific programmer as an enthusiastic learner and advocate for new computational techniques; a programmer with processes that work against the production of better code; and risk-averse and centered on the scientific problem.

- **The Working Environment** is the set of resource available to an individual, composed of components belonging to one or more of the following categories:
  - The **technical** corresponds to those things that the individual "sets up" immediately after getting a new workstation, or starting a major new project. This decision intentionally includes not only the essential tools needed for a task, but also the arguably aesthetic modifications.
  - **Personal** components which are, likewise, able to accommodate all prior experiences that suddenly become relevant in a task, and which are wholly peculiar to individuals, and are enabled by the personalization of the technical components.
  - **Social** components which include not only the resources available through formal or project-oriented means, but past colleagues, significant others, friends, roommates, and so forth.
- The working environment is subject to (1) evaluation and (2) change relative to the **Perfect World**.
- Deliberate change is possible in many small **Moments of Flux**. Four things are necessary for an individual to choose to pursue a change, in the following order:
  - awareness — "I have heard of it and know what it is."
  - normative framing — "I should probably be using it."
  - momentum — "If only I already knew it, I would use it!"
  - opportunity — "A clear course of action is available to begin to use it."

All of the above are preconditions for change, which is associated with **uncertainty** of cost: "how much time, energy, and money will I, or my team, spend before deciding this was the wrong choice?" Rather than a scarcity, there is an embarrassment of riches in terms of available tools or skills, which all have different complex costs and benefits associated with them. As a result, the orientation to change — which is sometimes characterized as taking a plunge, or going down a rabbit hole – is, by default, relatively conservative. On the other hand, popularity of workshops on teaching scientists to program in the span of days, weeks, or months becomes unsurprising if we recognize how these create *opportunity* for change by limiting the uncertainty of commitment using time limits and structure.

3 Qualitative Methods in Software Engineering Research

The previous section summarized a conceptual framework grounded in an 18-month qualitative study of programming in science, and particularly oceanography, where code work is done largely by people with minimal or no shared or formal programming background [10]. The study included four very different groups of scientists who were working with code. Some researchers were beginning to use R or Python scripts instead of Excel operations. Others not only used a self-made data analysis pipeline, but also contributed to an externally-facing visualization dashboard. Different forms of version control (including ad-hoc and file-based, as well as formalized and multi-contributor open-source) were used. Some of the conducting analyses that required working with relatively recent "big data" from instrumentations, which was sometimes made available from APIs, databases, or simply text files. Some of the analyses included the transformation and comparison of such data relative to the output of internationally-used models written in FORTRAN, demanding not only learning new programming skills but maintaining continuity with historical practices. Edwards provides an overview of this history in the area of climate modeling [4].

The use of qualitative methods provides a valuable complement to quantitative methods [16] because they enable answering *how* and *why* questions which are too broad to be addressed by experimentation in absence of comprehensive theory [8]. Though controlled experiments can produce robust knowledge about cause-effect relationships, there are many threats to internal and external validity in practice [17, 18].

The challenge in the exploitation of the full potential of quantitative methods is the relative lack of a broad, coherent theory of human behavior with regard to software engineering as a sociotechnical practice. Theories serve three functions in software engineering: (1) to define terms enough to identify them in the wild, (2) to explain the choices that people make around the subject of study, and, to the extent plausible in context, (3) predict qualities of what people do based on certain factors identified as relevant by the theory; qualitative methods provide a systematic approach to theory-building, in turn enabling the hypothesis testing demanded by empirical studies [3].

Easterbrook *et al.* [3] and Wohlin *et al.* [20] provide in-depth advice for the costs and benefits of different empirical methods. Qualitative methods appear in some form in a variety of SE research methods. Though detailed and useful, **case studies** (exploratory or confirmatory) are not generalizable, but allow only for claims of theoretical replication: that other cases are expected to be the same, or they are expected to differ in predictable ways described by the theory [3]. These data collection methods can be used also in **post-mortem analyses**, which are distinct from case studies in that they are done regarding a project (or part of a project) that has been completed, and therefore capture chiefly reflections [20]. Similar methods are also useful in **ethnographies** which (unlike case studies) involve a longer-term exposure to study subjects or informants and are focused on understanding the internal meanings of a community's cultural practices in order to build local theories [3].

Finally, in **action research,** the researcher is an intentional agent of change, unlike in an ethnography, but may undertake similar methods during design process for potentially an extended period of time [3].

In the study of oceanographers described in the beginning of this section, the choice of the word "code," rather than "software," "program," or "script" is intentionally inclusive. "Code" refers to the programmatic instructions written in the course of software production (and sometimes software use), programming, and scripting. The primary context of this work is code that is experimental, in development, or "in permanent beta," rather than infrastructural or essential code, using the distinction articulated by Kelly [7]. The strength of the qualitative methods here was the focus on participants' own terminology (*emic* terms), in addition to interpreting their actions through the lens of software engineering or computer science terminology (*etic* terms). The use of etic ("outsider looking in") terminology allows the researcher to see parallels in practice – e.g., to identify when certain development or debugging practices appear, despite not being recognized or named as such by study participants. The use and interpretation of emic terms, however, is crucial to the construction of the kind of conceptual framework presented in section 2.

The terms "bug" and "debugging", for example, were practically non-existent; most participants said that their code "is not working" or is "being weird." Rather than "debugging code," they would re-examine the code as well as look over the data (if any) in plain-text or in an overview visualization, or re-do the math (if any) with pencil-and-paper or on a whiteboard. In most cases, the bug had to do with assumptions in parsing and formatting of dates; unit conversion; or a divide-by-zero error arising from the current coder making assumptions different from the previous coder on the range of some variable(s). In other words, the bug turned out to be a code bug (rather than a conceptual or mathematical error), despite not being referred to as such. The non-use of the terminology is meaningful in that it reflects a common claim about software engineering in science: that the implementation is understood and expected to be so close to the concept that even in more complex programs, design, debugging, and testing are not distinguishable processes with respect to the code itself [5]. It is possible to identify aspects of agile practice (to take one example) in observing the programming practices, or to ask about "testing" directly, but this obscures an essential aspect of how the programmer understands code and coding.

4 Bad Code

The qualitative study which was used to construct the conceptual framework in section 2 involved observation of indidivuals coding: the observer was present for several hours (between 4 hours and the full day) in the office or lab, and worked alongside the informant. When the informant came to natural stops – like leaning back and sighing deeply, or exclaiming "it works!" – the observer asked questions about what was problematic or exciting; how unexpected behavior came to be exposed; and what the next steps were. Analyzing these observations again, in light of

the deliberate change framework from section 2, revealed two kinds of **"bad" code that persists**:

- Type 1, recognized at production: "I know it's bad, but I'm doing it anyway."
- Type 2, recognized in retrospect: "It's ugly, but it has historical reasons, and I am not changing it."

A common example of Type 1 bad code in scientific practice observed was copying and pasting. This is the Number 1 "code smell" that suggests something is bad [1], as well as a top target for constructive criticism among scientific programmers themselves [19]. Nevertheless, in practice, this happened most often and obviously with code used to generate figures, either in short scripts or in interactive environments like Jupyter Notebook. In this case, the main thing that makes copy and past bad in the first place (duplicated a functionality which then becomes inconsistent over time, undermining anything that depends on it) *does not really apply* because figure code is:

- not library code but rather "kleenex" [5] code, and even if re-used, does not in this case constitute a dangerous dependency problem,
- used more often as a reference: indeed, the code for a figure acts itself as a kind of documentation for that figure, and having quick access to it is useful.

The example of "documenting" figures with copy-pasted code is particular to the scientific domain. However, Type 1 code can serve a social role in the maintenance of the codebase: as material embodiments of a scaffolding necessary for the intermediate and experimental stages of development [14]. Returning to the role of social resources in development, we can also see that "developers spend vast amounts of time gathering precious, demonstrably useful information, but rarely record it for future developers" [11]. So when does the Type 1 code, accompanied by a comment like "TODO: fix this...", conceal legitimate expert judgments and prioritisations? How often are these fixed and/or documented in practice, and how often they are neither fixed, not documented, but have to be re-explained?

A common example of Type 2 bad code was the use of libraries or tools that provided continuity within the field or within the group, such as having to do with how data was stored and formatted. In this example, too, the code is not "bad" so much as it is the result of an unsatisfying balancing of real constraints, both historical and current, for which no other compromise is imagined that is so much better as to justify a complete overhaul of everything dependent on it. When Type 1 code is intended as a "temporary hack" but end up being less temporary than initially hoped, it becomes the kind of Type 2 hack that constitutes "technical debt" [9].

The above examples include bad code which: starts out as Type 1 but will not become Type 2; ends up as Type 2 but does not start out as Type 1. This code may be not perfect, but it is benign relative to more significant defects. Furthermore, rather than being benign, some Type 2 bad code may conceal knowledge about prioritization that had been lost. The problem is that it is unclear which kinds of

bad code occur with what frequency, and how many of them are benignly inelegant; secretly informative; accumulating technical debt; or potentially catastrophic.

How does this anthropocentric typology of "bad" code relate to actual software quality? To answer this question, the collections of scripts and data analysis pipelines used by two of the four teams studied were inspected. These teams had made their code available freely online, along with version control information. The preliminary analysis was done wholly manually, for three reasons: there was a great deal of variation in how the version control tool was used over time by different individuals so automation would have been more challenging; there was not a prohibitive quantity of activity; the existing familiarity with the projects allowed the interpretation of some of the more obscure comments or messages. Two patterns emerged:

- The phrase "cleaning up …" in the commit message mapped onto relatively substantial fixes of Type 2 bad code. This, incidentally, looks a lot like the "Shotgun Surgery" code smell, which includes "mak[ing] a lot of different changes to a lot of different classes" [1]. Interestingly, "clean up" seemed to span both aesthetic improvements, documentation, major simplifications of loops used in analysis, and in some cases, identifying and removing a source of faulty behavior.
- Commits that claimed to fix some particular bug or address a particular issue routinely had a small *hanger-on* fix, often a minor readability improvement.

What patterns would emerge from other open-source projects, that have a less exploratory character in terms of software practice than the above? What are the advantages and drawbacks of the framework described in this report, relative to other typologies used in the literature (e.g., code smells)? The objective is to identify some available projects for analysis: first by sampling and qualitatively annotating in order to identify characteristic patterns, and if possible and appropriate, supplement systematic manual inspection with automation.

Mining code is difficult because digital traces of software development are not only incomplete, but incomplete in different ways for different groups of programmers. For example, Kalliamvakou *et al.* list eight potential dangers of using GitHub data [6], which can be explained by the deliberate individual change framework in the following way:

- Working environments are unique to individuals; they embody and make use of personal resources in ways that may mean not only adopting, but adapting the tools at hand.
  Matching perils from Kalliamvakou et al.*:*
  - I "A repository is not necessarily a project"
  - II "Most projects have very few commits"
  - III "Most projects are inactive"
  - IV "A large portion of repositories are not for software development"
  - V "Two thirds of projects (71.6 % of repositories) are personal"

- Digital traces, constitute a partial view of a particular social/technical resource of a developer or project's working environment; other possible co-existing social/technical resources are not represented.
  Matching perils from Kalliamvakou et al.*:*

  VI "Only a fraction of projects use pull requests. And of those that use them, their use is very skewed"

  VII "If the commits in a pull-request are reworked (in response to comments) GitHub records only the commits that are the result of the peer-review, not the original commits"

  VIII "Most pull requests appear as non-merged even if they are actually merged"

  IX "Many active projects do not conduct all their software development in GitHub"

In other words, under this framework, there is no assumption that any two groups would tooling in the same way. The notion of automated or semi-automated detection of bad code is not without precedent, and not without major challenges. A proof of concept would be either a fully automated approach, or an automated approach that has particular requirements on data made available to it, such as manually-prepared examples or test data. However, an initial qualitative-first approach can help iterate on the typology, and determine whether it is valuable relative to other existing typologies. Whereas existing approaches in identifying "code smells" have considered the analysis of the technical components of the working environment, this approach is distinct by including the social, both in identification of "bad code" and in the opportunities for productive intervention.

5 Summary

The objective is to design and pilot a mixed-methods approach for investigating code that is identified as somehow deficient, but which nevertheless persists. This report describes a preliminary theory of bad code, describing this phenomenon and making it discernible in the real world software projects as well as helping to explain the choices made by the developers. This theory has so far been developed using qualitative interview and observation methods within the particular domain of scientific software development, particularly oceanography. Though the qualitative approach has notable strengths, the next steps emphasize identifying other code bases and developing a productive combination of qualitative and automated methods for further refining and interrogating this explanatory conceptual framework.

References

[1] K. Beck and M. Fowler. *Refactoring: Improving the design of existing code*. Edited by M. Fowler, K. Beck, J. Brant, and W. Opdyke. Addison-Wesley, 1999. Chapter Bad smells in code, pages 75–88. ISBN: 978-0-201-48567-7.

[2] S. M. Easterbrook and T. C. Johns. "Engineering the software for understanding climate change". In: *Computing in Science & Engineering* 11.6 (2009), pages 64–74. DOI: 10.1109/mcse.2009.193.

[3] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. "Selecting empirical methods for software engineering research". In: *Guide to advanced empirical software engineering*. Springer, 2008, pages 285–311. DOI: 10.1007/978-1-84800-044-5_11.

[4] P. N. Edwards. *A Vast Machine: Computer models, climate data, and the politics of global warming*. MIT Press, 2010. ISBN: 978-0-262-01392-5.

[5] D. Heaton and J. C. Carver. "Claims about the use of software engineering practices in science: A systematic literature review". In: *Information and Software Technology* 67 (2015), pages 207–219. DOI: 10.1016/j.infsof.2015.07.011.

[6] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. "The promises and perils of mining GitHub". In: *Proceedings of the 11th working conference on mining software repositories*. ACM. 2014, pages 92–101. DOI: 10.1145/2597073.2597074.

[7] D. Kelly. "Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software". In: *Journal of Systems and Software* 109 (2015), pages 50–61. DOI: 10.1016/j.jss.2015.07.027.

[8] A. Ko. *Making Software: What Really Works, and Why We Believe It*. O'Reilly, 2010. Chapter Understanding software engineering through qualitative methods, pages 55–64. ISBN: 978-0-596-80832-7.

[9] P. Kruchten, R. L. Nord, and I. Ozkaya. "Technical debt: from metaphor to theory and practice". In: *IEEE Software* 6 (2012), pages 18–21. DOI: 10.1109/ms.2012.167.

[10] K. Kuksenok. "Influence apart from Adoption: How Interaction between Programming and Scientific Practices Shapes Modes of Inquiry in Four Oceanography Teams". PhD thesis. University of Washington, 2016.

[11] T. D. LaToza, G. Venolia, and R. DeLine. "Maintaining mental models: a study of developer work habits". In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pages 492–501. DOI: 10.1145/1134285.1134355.

[12] J. Leach, D. Nafus, and B. Krieger. "Freedom imagined: morality and aesthetics in open source software design". In: *Ethnos* 74.1 (2009), pages 51–71. DOI: 10.1080/00141840902751188.

[13] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka. "The evolution and impact of code smells: A case study of two open source systems". In: *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement*. 2009, pages 390–400. DOI: 10.1109/esem.2009.5314231.

[14] W. J. Orlikowski. "Material knowing: the scaffolding of human knowledgeability". In: *European Journal of Information Systems* 15.5 (2006), page 460. DOI: 10.1057/palgrave.ejis.3000639.

[15] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. "Detecting bad smells in source code using change history information". In: *Automated software engineering (ASE), 2013 IEEE/ACM 28th international conference on*. IEEE. 2013, pages 268–278. DOI: 10.1109/ase.2013.6693086.

[16] C. B. Seaman. "Qualitative methods in empirical studies of software engineering". In: *IEEE Transactions on software engineering* 25.4 (1999), pages 557–572. DOI: 10.1109/32.799955.

[17] D. I. Sjøberg, T. Dyba, and M. Jorgensen. "The future of empirical methods in software engineering research". In: *2007 Future of Software Engineering*. IEEE Computer Society. 2007, pages 358–378. DOI: 10.1109/fose.2007.30.

[18] D. I. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. "A survey of controlled experiments in software engineering". In: *IEEE transactions on software engineering* 31.9 (2005), pages 733–753. DOI: 10.1109/tse.2005.97.

[19] G. Wilson, D. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. Haddock, K. Huff, I. M. Mitchell, M. D. Plumbley, et al. "Best practices for scientific computing". In: *PLoS biology* 12.1 (2014), e1001745. DOI: 10.1371/journal.pbio.1001745.

[20] C. Wohlin, M. Höst, and K. Henningsson. "Empirical research methods in software engineering". In: *Empirical methods and studies in software engineering*. Springer, 2003, pages 7–23. DOI: 10.1007/978-3-540-45143-3_2.

Event Subscription

Sankalita Mandal

Business Process Technology
Hasso-Plattner-Institut
Sankalita.Mandal@hpi.uni-potsdam.de

Today, business process management is a key approach to organize work, and many companies represent their operations in business process models. These business processes are executable using process engines. The process engines can produce and consume events for the completion of the processes. However, to receive the external events, we must first subscribe to them. BPMN does not specify concrete guideline about when the subscription should be made. Lack of proper subscription and unsubscription can cause a process to get stuck or behave incorrectly. In practice, the subscription for the events are generally done at process deployment, although this is not always a smart solution. The work presented here explores different phases a process goes through and evaluates the earliest and latest subscription point for each event kind.

1 Introduction

These days, business processes are at the core of all operations in many organizations [6]. The process engines like Camunda[1] or Activiti[2] support the deployment and execution of process models. Also, in the recent years, we have seen that the information that we can gather from external events can be very helpful to make our processes more flexible and efficient in term of reacting towards the environmental happenings [3]. These external events are mapped to BPMN events [5] in the process models. The process engine receives these external events either directly from the event sources (e.g., an event of thunderstorm from a weather API) or from an event processing platform (e.g., Unicorn[3] built in Business Process Technology Group) [4].

To receive the events, the process engines first need to subscribe to specific event(s) [1]. This is generally done by registering an event query to the event platform. On occurrence of the events, if the event platform finds a matching query, then the subscriber is notified about the event(s). Since there exist different types of events in a BPMN process (e.g., start, intermediate, boundary), the question lies when to subscribe for which event. The common practice is to register the event queries at the time of deployment of the process model. But this is not efficient enough as all the events are not relevant for the whole duration of the model being deployed [2]. Therefore, we propose specific subscription points for each type of BPMN events that

[1] http://camunda.de (last accessed 2016-10-20).
[2] http://activiti.org (last accessed 2016-10-20).
[3] http://bpt.hpi.uni-potsdam.de/UNICORN (last accessed 2016-10-20).

describe when to subscribe to which events during process deployment, instantiation and execution. The points of unsubscription are next on our research agenda.

2 Motivation

This section provides the rationale behind the work done. Let us consider the example shown in Figure 1. The example describes the process of offering a seminar for a semester in our institute. Before start of the semester, the student office sends an email or publishes an website notification for the research groups saying it is time to offer the seminar for next semester. The groups then prepares and publishes the seminar topic. If an interested student applies for the seminar then it takes place. Otherwise the seminar is dropped after the deadline for application is gone.

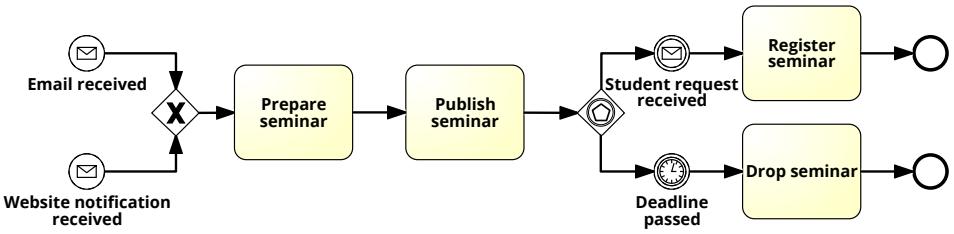


Figure 1: Motivating example for the need of proper (un)subscription.

2.1 Benefits of the Approach

Now, to get this information that the semester is approaching, we need to subscribe to the mailing list of student office and also to the website notification. Otherwise, we miss the information and cannot offer any seminar, i.e., the process is never initiated due to lack of subscription. On the contrary, if we think about the scenario that we have already started preparing the seminar after receiving the email, but have not unsubscribed to the website notification yet, then the website notification can start another process instance, which is not expected. If we look further in the process, after the deadline for application is past, we should not consider any student request anymore for the proper completion of the process. Therefore, for the sake of correct behavior of the process, we need to know at which point of time we need to subscribe and unsubscribe to certain events.

Apart from the correct behavior, there is another very important aspect for which the proper subscription is necessary, i.e. the performance enhancement of the process engine as well as the event platform. If we can skip the subscription queries which are not relevant at specific point in time then that decreases the overhead on the event platform. In case the events are buffered in the event platform for further complex

event processing, then proper subscription and unsubscription makes the storage of events just as per need and makes the event platform work more efficiently. Also, if we unsubscribe when the event is not expected any more, the process engine is not bothered with receiving unnecessary event stream.

Therefore, to summarize, proper event subscription and unsubscription provide us with the following benefits:

1. **Correct behavior**
   - process is not stuck due to missing subscription
   - unwanted instances are not started due to delayed unsubscription
   - process instance can be completed properly
2. **Performance enhancement**
   - events are not stored for indefinite time
   - subscription query/notification overload is avoided
   - irrelevant event streams are not processed

In the next section, we talk about the state transitions a process goes through. Further, we specify the state transition for a process instance as well. Later, we introduce the points of subscription based on them.

2.2 Lifecycle of Process Model and Instance

A process model can go through three phases. Once the process is modeled, it can be deployed and then eventually can be undeployed as well. On the other hand, the lifecycle of a process instance can only start if the process model is in deployed state. Once the start event occurs, it is instantiated. Then it waits for further execution of activities. The moment the next activity is started, the instance enters the running state. When the end event is triggered or the instance is aborted otherwise, it reaches the state terminated.

Based on the above mentioned stages a process or a process instance has to go through, we define few points at which event subscriptions can be done. According to the position and behavior, BPMN classifies the events in three categories: Start Events, Intermediate Events and End Events. As we are concerned about the events for which we need subscriptions, it is obvious that we consider only catching events in the scope of our paper. Therefore, End Events and Intermediate Throwing Events are out of scope. For the rest of the event kinds, we recommend the points of subscription suitable for the kind. Furthermore, for each event kind, earliest and latest subscription points are discussed. Based on the process context, engine and event platform capacity and required process efficiency, the decision for earliest or latest point of subscription for each kind of event should be considered.

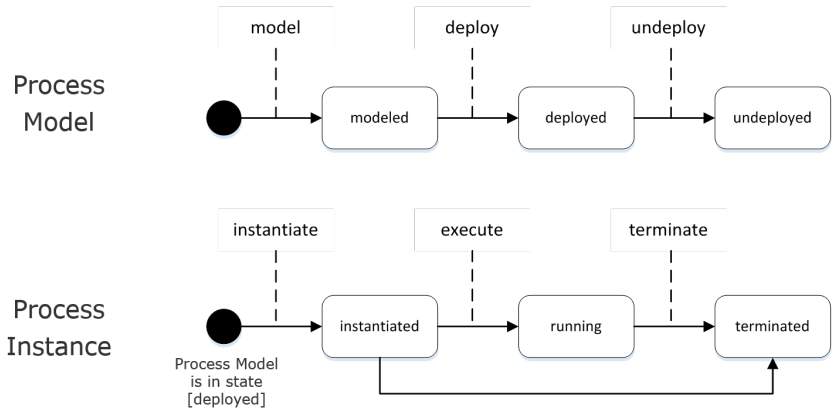


Figure 2: State transition diagram of process model and process instance.

2.3 Points of Subscription

As already mentioned, the events can be subscribed to at various points in time during process deployment, instantiation and execution. Figure 3 gives an overview of the possibilities of subscription. All the points of subscription are explained using another example process which shows the process of preparing for a conference. We discuss the variations of the process activities when it deems fit.

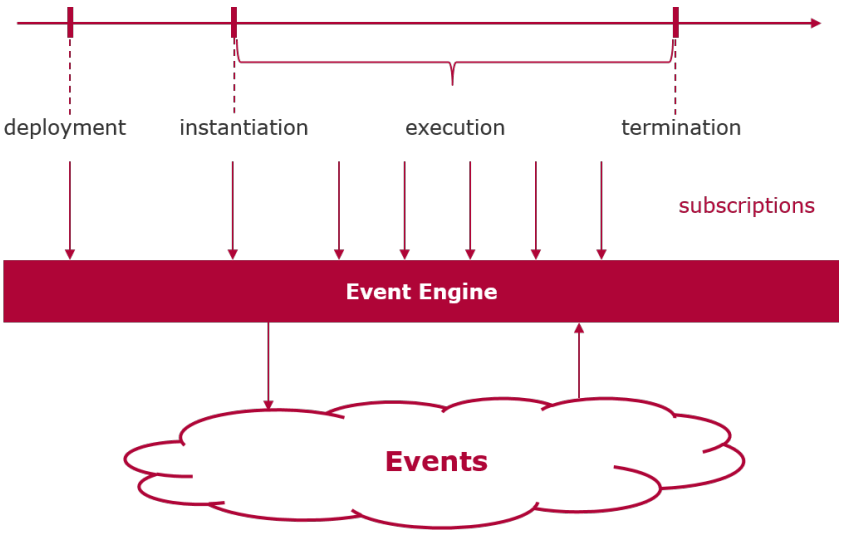


Figure 3: Subscription points during different phases of process.

1. **At process deployment.** Subscription is created after the model is deployed. If we consider the start events, then we see that they are needed for process instantiation. Therefore, they must be subscribed before process instantiation,

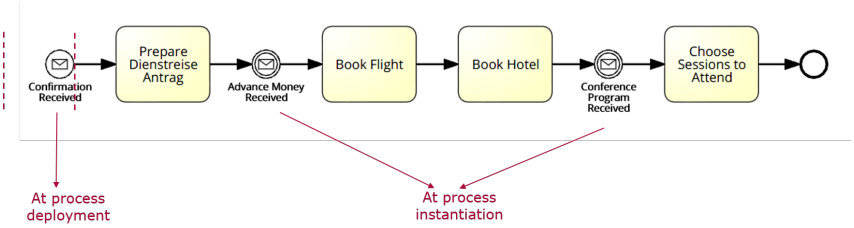


Figure 4: Point of Subscription: At process deployment and instantiation.

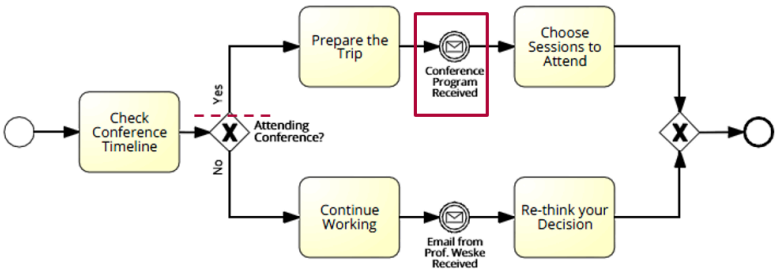


Figure 5: Point of Subscription: All in one.

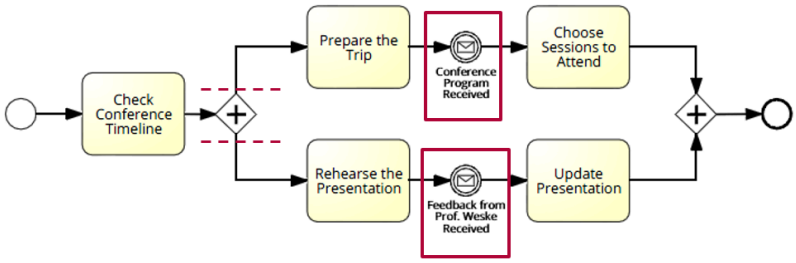


Figure 6: Point of Subscription: All in all.

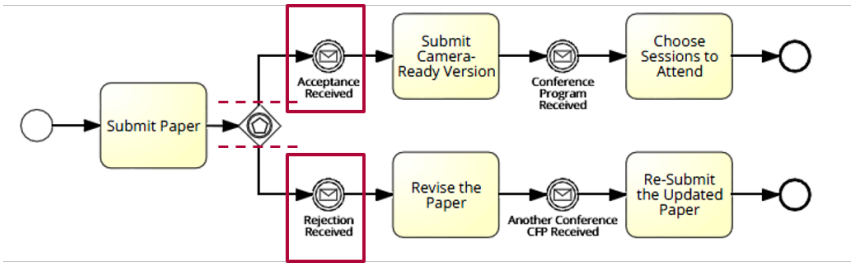


Figure 7: Point of Subscription: Only immediate.

otherwise the process is never started. In such cases, the subscription point should be at deployment.

In Figure 4, the process starts once the conference confirmation is received, we need to prepare `Dienstreiseantrag` or application for a business trip. Then we wait to receive some money in advance. Once we receive the money we can book the flight and the hotel with it. However, we still wait for the conference program to choose the sessions we want to attend. Now, we need to subscribe to the event `Confirmation Received` to get notification about the conference. This might be done by subscribing to the mailing list or the Tweets or may be the newsletter. The first dotted line shows the point of time when the process is deployed and the second one shows when the process is instantiated.

2. **At process instantiation.** Subscription is created at instantiation. As discussed in Section 2.2, a process model is instantiated at the point of occurrence of the start event or, in case of multiple start events, the occurrence of the first one of its start events. Thus, the point of subscription lies after the start event has occurred. The events that lie in the path of sequence flow and have to occur for the completion of the instance should be subscribed at instantiation. Again, if we look at Figure 4, the events `Advance Money Received` and `Conference Program received` can be subscribed at this point.

3. **At gateway activation.** Events are subscribed to after a specific gateway is terminated and the token is passed to the control flow of one or more branches following the gateway. Since the gateways determine the relevance of some events, it is more efficient not to subscribe to all the events at deployment or instantiation. Depending on the gateway type, there can be three ways of subscription as described below.

    - **All in one.** If it is a data-based XOR split gateway, then only one branch of the alternatives are chosen after the gateway is activated. In this case, only the events situated in the chosen path are subscribed to. The events that lie in the disabled paths are not subscribed during the course of execution of this particular process instance. In our conference preparation process, if we consider from a little before, then we might first check the conference timeline and decide whether to attend it or not. Based on the decision we either prepare for the trip or continue regular working. Now, only if we choose to go, we are interested in the conference program. Therefore, we subscribe to the event `Conference Program received` only when the gateway is activated and control flow is routed to the branch *yes*, as shown in Figure 5.
    - **All in all.** The parallel split gateway, on the other hand, has different semantics. If a parallel gateway is terminated, then all the outgoing branches are activated. Therefore, subscriptions are created for all the events in all the branches after the gateway. In a similar process like before, if we also need to present a paper in the conference, then we might initiate the activities required for the preparation of the talk in parallel with the conference

organization. In Figure 6, we need to subscribe to both the events `Conference Program Received` and `Feedback from Prof. Weske` once the parallel gateway is activated.

- **Only immediate.** This applies for the event-based gateways. In case of an event-based gateway, when the gateway is terminated, subscriptions should be made for all the events which are immediate successors of the gateway. Because we do not know at this point which path will be followed, we do not subscribe to the events situated further in the outgoing branches. For example, if we submit paper in a conference, then we either receive an acceptance or a rejection. Based on the event we receive we choose the next activities. So at this point we only subscribe to the events `Acceptance Received` or `Rejection received` in the example shown in Figure 7.

4. **At activity termination.** Here, subscription for an event is made after the preceding activity is terminated. This is consistent with the semantic of the flow of token. When an activity terminates, the token is passed through the outgoing edge and the following event is enabled. So, we subscribe to the event and the state changes to enabled. Then the event occurs at some point and the token is passed to the next node according to the sequence flow. In the example process in Figure 8, the event `Advance Money Received` can be done after the previous activity terminates (shown by red dotted line).

5. **At activity start.** Subscription is created when certain activity is started. This is a special case needed for specific kind of events, i.e., the boundary events. They are only relevant once the adjacent activity is started. Thus, we subscribe to them only when the associated activity starts. Figure 8 also depicts this. The event `Super Saver Offer Received` is relevant only when the activity `Book Hotel` is started, again shown by red dotted line.

6. **At event occurrence.** Subscription is created after certain event has occurred. This applies when the occurrence of the event determines the next path to follow. For example, after an event based gateway, one of the expected events occur and the branch following this event is taken thereafter. The events in other branches are not relevant anymore. In such scenarios, we create the subscription after the occurrence of the event that determines the course of the process. In Figure 9, once we know that the paper is accepeted, we subscribe to the event `Conference Program Received` as it becomes relevant only after we get the acceptance.

2.4 Application on Event Kinds

So far, we have discussed different points when subscriptions can be created. In the following we will see for each event kind, which should be the suitable point of subscription. There can be an earliest and a latest point for each event subscription. However, for specific events, the earliest and latest subscription points overlap.

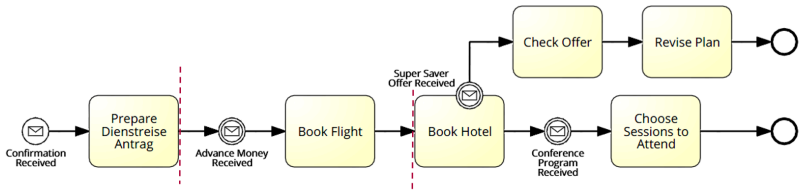


Figure 8: Point of Subscription: At activity start and termination.

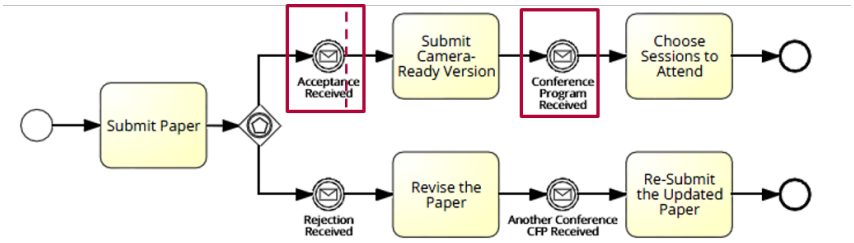


Figure 9: Point of Subscription: At event occurrence.

- **Start Events.** Start events are required to instantiate the process. Therefore, they must be subscribed at the time of deployment. In case the process has multiple start events, if any of them happens then the process is instantiated. The later required start events, if any, are correlated to the existing instance. in this case, the earliest and latest point of subscription are same.
- **Intermediate Events.** BPMN intermediate events can be further classified depending on their position in the process path. They are described below.
  - **Events outside gateway branch.** These are the events that lie on the sequence flow before a split gateway or after a join gateway. The occurrence of these events are anyways required for the process completion, i.e., they must occur once the process is instantiated. Hence, the earliest point of subscription for these events are at instantiation. On the other hand, subscriptions for these events can also be created once the preceding activity terminates. Therefore, the latest point of subscription for intermediate events outside gateway branch is at activity termination.
  - **Events inside gateway branch.** These events are placed in one of the branches of a gateway. Depending on the type of the gateway, they may or may not be required for the process instance. This can only be known once the gateway is terminated and the token is routed to one or more outgoing edges. Thus, they should use the earliest subscription point at gateway activation depending on the semantic of corresponding gateway. Again, the latest point of subscription can be at activity termination for these events.

One exception should be pointed out here. The events that are the immediate successors of an event based gateway also fall in this category. But for them, there is no preceding activity. For them, both the earliest and latest point of subscription are at gateway termination, only immediate to be specific.

– **Events attached to boundary.** The boundary events are relevant only in the scope of the activity it is attached to. Thus, it does not make sense to subscribe to it before the activity starts. So we use the subscription point at activity start here. Also, the earliest and latest points of subscription are same for boundary events.

– **Events in exceptional path.** These are the events situated in the outgoing branch of the boundary event. Be it an interrupting or non-interrupting boundary event, the events in exceptional path are required only if the boundary event is fired. Naturally, it is more efficient to subscribe to them after the boundary event has occurred. Thus, the earliest point of subscription for them is at event occurrence whereas the latest point remains at activity termination.

The summary of the event kinds and the earliest and latest points of subscription for them are shown in Figure 10.

Event Kind	Earliest POS	Latest POS
Start events	At process deployment	At process deployment
Events outside gateway branch	At process instantiation	At activity termination
Events on XOR gateway branch	At gateway activation	At activity termination
Events on AND gateway branch	At process instantiation	At activity termination
Events immediately after Event-based gateway branch	At process instantiation	At gateway activation
Events in exceptional path / On event based gateway branch	At event occurrence	At activity termination
Boundary events	At process instantiation	At activity start

Figure 10: Earliest and latest Points of Subscription.

References

[1] A. Barros, G. Decker, and A. Grosskopf. "Complex Events in Business Processes". In: *Business Information Systems*. Springer, 2007.

[2] G. Decker and J. Mendling. "Process instantiation". In: *Data Knowl. Eng.* 68.9 (2009), pages 777–792. DOI: 10.1016/j.datak.2009.02.013.

[3] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Co., 2010. ISBN: 978-1-935182-21-4.

[4] N. Herzberg, A. Meyer, and M. Weske. "An Event Processing Platform for Business Process Management". In: *EDOC*. IEEE, 2013.

[5] OMG. *Business Process Model and Notation (BPMN), Version 2.0*. Jan. 2011. URL: http://www.omg.org/spec/BPMN/2.0/ (last accessed 2016-10-01).

[6] M. Weske. *Business Process Management – Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012. ISBN: 978-3-642-28615-5.

Relying on Development Data for Software Development Processes

Christoph Matthies

Enterprise Platform and Integration Concepts
Hasso-Plattner-Institut
christoph.matthies@hpi.de

The way that developers work together, manage tasks and organize themselves impacts the software that is produced. There is thus the need to practice and uphold an effective software development process. When it comes to improving or finding weaknesses in the executed process, teams often rely on self-improvement sessions or mentoring by a knowledgeable third party. These methods rely on subjective evaluations and their results are hard to quantify. After changes are implemented it is difficult to assess their impact and whether an identified issue was fully resolved. In order to tackle these challenges we propose an approach relying on analysing data created by software development teams. Software developers create more than just code during regular development activities. Development artifacts, such as commits, tickets, wiki pages, code coverage statistics and build logs, contain information on how a team worked together. By aggregating, linking and analysing this already present data, insights into the development process of teams can be generated.

1 Overview

Specialised software analysis tools are integral parts of modern software engineering. They allow insights into the created products where manual analyses would be complex and costly. Examples for such programs include linters as well as code coverage or formal verification analyses. Most of these common tools rely on static program analysis, scrutinizing the produced source code and giving recommendations for code improvements. Data on the specifics of how the code was created is not needed and is not evaluated. This is of benefit, as no additional overhead for collecting development data is introduced, e.g. by surveying developers or making them document their efforts.

We argue that existing development data can be used to gain insights into the employed software development process. Adding administrative work is avoided, as only artifacts created during regular development activity are considered. This is enabled by the way that software engineers work: In contrast to many other engineering disciplines, software developers "self-document", i.e. they frequently produce artifacts that allow following their progress. For example, by committing code into a common software repository and updating the corresponding issue, a software engineer not only shares exactly what was done, but a variety of other information:

- **Commit metadata**, such as the commit author's email address as well as the date and time that the changes were made.

- **Commit message**, comprising the goal of the change, as well as the sentiment of the developer while she was writing it. Furthermore, the commit message can include structured information, such as an issue number (e.g. "Ref #123").
- **Issue information**, including the motivation for the change, possibly in a semi-structured "as a <user> I want to <goal> so that <benefit>" user story pattern.
- **Issue metadata**, including information on which developers were assigned or reassigned to this ticket over time, when the ticket was created and by whom, which priority it has, as well as other labels attached to it.
- **Issue comments**, made by other developers, clarifying uncertainties, requesting changes or discussing the merits of the issue's goal or associated commits.

In most development teams, the created development data is shared openly. This is especially true for open-source development. In most cases even data on the problems that occurred is readily available, through discussions in issues, wiki pages or logs of broken builds. With this wealth of available information, we can build tools that can help developers gain an overview of the *implicit* data they and their team produce and allow them to identify possible problems in the executed process.

2 Approach

In order to develop effective data-driven tools for supporting development processes access to software development teams is vital. Only with observations on how teams interact, what existing development tools are used and what data is produced, can tools be built that can attempt to be relevant.

2.1 University Courses for Data Mining

Software engineering II is a final year undergraduate software engineering course in which multiple student teams, employing the Scrum methodology, develop a single software system together. It provides a recurring opportunity to gather realistic data and develop appropriate metrics for real development teams facing real-world challenges. Especially analysis on frequent "beginner mistakes" are pertinent. Teams with little experience applying agile methodologies are likely to make mistakes, which can be quickly rectified by providing feedback or pointing to alternative practices. The course blends a hands-on development project, coaching by tutors as well as minimal lectures on agile methodologies. Figure 1 depicts the Scrum process that is being followed, including adaptations that account for the fact that students cannot devote their entire work time to the project.

In order to be as close to real-world scenarios as possible, students are consciously allowed to make mistakes and learn from them. One of the areas that is focused on are the values of the Agile Manifesto, such as "individuals and interactions over processes and tools". Students are given authority and responsibility for decisions

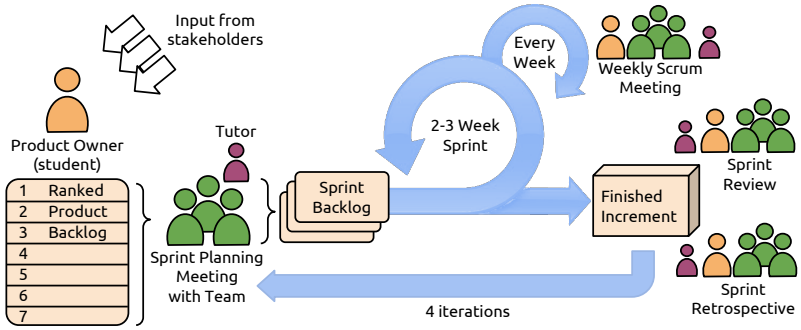


Figure 1: The modified Scrum process followed in *Software Engineering II*.

and role assignment and self-organize in their teams. These *self-organizing teams* are one of the base concepts of agile methodologies [3] and have been identified as a critical success factor for projects[1].

In order to ascertain what impact the focus on self-organizing teams that create real development data has on the satisfaction with Scrum as well as the overall course design, surveys were devised and analysed [8]. As the same survey as in a previous study by Mahnic [7] was used, the results could be compared to those of Mahnic, who employed a much stricter, more controlled course setup, featuring no explicit self-organization by students. Figure 1 lists the topics and questions of the main survey.

Both courses received consistently positive scores over all sprints for questions relating to satisfaction with the performed work as well as the Scrum methodology. These findings are in line with more extensive studies [10] on the subject.

However, the survey question directly relating to students' satisfaction and experience with letting teams self-organize only received slightly above neutral average results. Answers featured standard deviations of more than one point, a disparity that was also evident in feedback by students. While some students looked fondly on attending only a few lectures and focussing on self-organization, others would have preferred more guided tutorials and stricter organization. Reconciling these two sides in a software engineering course is an ongoing challenge. Even so, the focus of the course on self-organizing teams did not hinder the perceived learning results of students.

2.2 Data Collection

Once it is clear, which teams should be analysed and what their context is (e.g. what tools they are using), as a first step, the development artifacts that they produce have to be collected.

Modern development teams are supported by a variety of tools. These include programs which have become more or less standard, such as issue trackers and version control systems, but also a variety of other solutions, e.g. continuous integration or static code analysis.

Table 1: Main sprint survey adapted from Mahnic [7], which was filled out by students at the end of every of the four sprints of the project. Questions could be answered on a 5-point scale of "strong no" to "strong yes".

#	Question
1	**Clarity of requirements in the Product Backlog** Were the user stories in the backlog clear enough? Did the descriptions suffice to understand what the Product Owner really wanted?
2	**Effort estimation** Were the estimates of required work (story points/man-hours) of user stories adequate/realistic?
3	**Maintenance of the Sprint Backlog** Was it clear how to handle user stories? How to log performed work?
4	**Administrative workload** Does the administrative work called for by Scrum (meetings, planning, reviews, maintenance of backlog, etc.) represent a significant additional workload?
5	**Cooperation with the Scrum Master** Was the cooperation with the Scrum Master adequate/satisfactory? Did the Scrum Master contribute to the team's success?
6	**Cooperation with the Product Owner** Was the cooperation with the Product Owner adequate/satisfactory?
7	**Cooperation with other Team members** Was the cooperation with other Team members satisfactory/adequate? Did the Scrum process encourage cooperation?
8	**Workload** Was the amount of work required for the project adequate?
9	**Satisfaction with work** Are you satisfied with the work/the results of this sprint?
10	**Satisfaction with Scrum** Is the methodology useful? Would you recommend Scrum to other software developers?

In order to perform analyses on the data within these systems an *ETL process* is necessary. Data needs to be extracted, transformed into a unified data scheme and loaded into a suitable database.

A major challenge in implementing this process is keeping up to date with the development of new systems and the evolution of existing tools. Most current systems that exist for collecting development data, such as SonarQube [2] rely on connection plugins for each data source, which need to be updated when the source changes. Furthermore, they define a rigid data scheme that custom data (e.g. extra fields on a ticket) do not necessarily fit into. Lastly, it is of benefit to allow a selection of target databases that the mined and connected data can be written to, depending on the analysis use case. For example, if the social graph of developers is to be analyzed, a graph database which includes graph analysis algorithms is best. However, when text analysis is required, a document store might be superior.

DataRover [6] tackles these issues by reducing the implementation effort for ETL workflows. The querying of the data source APIs and data storage are separated. Figure 2 describes the architecture of the system. *Explorers* are each responsible for querying a single data source and represent a minimalistic connector implementation.

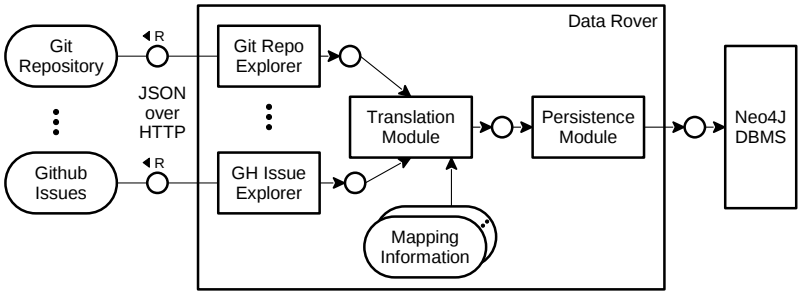


Figure 2: FMC block diagram of the architecture of the data rover.

The transformation step into a property graph, which has no previously defined data scheme, is guided by user-defined mappings. These are created through a graphical front end based on sample API JSON responses. For example, every commit could be linked to its corresponding continuous integration build by its SHA hash. A Neo4J graph database serves as an intermediate data sink that stores the connected graph of development data. The graph can then be exported to a variety of different databases to allow further analysis and the use of other tools.

Using DataRover, it is possible to create minimalistic data sets tailored to specific use cases, by using a graphical front end and only having to write querying code in case of API changes. The system's linking mechanisms further allow adding additional data sources when needed and extending existing analyses.

2.3 Development Data Analysis

Once the data of all the tools that development teams use is collected, linked and stored, it can be analysed. Our main use case, is to gain insights into the development process of student teams, to find out how well Scrum and agile best practices are followed in a team and allow targeted counteractive measures. To this end, we propose a set of nine objective, automated conformance metrics which can perform this assessment [9], complementing proven, battle-tested techniques, such as assessments by tutors or exams. We have split the developed metrics into three categories:

- **XP Practices**: Metrics measuring violations of Extreme Programming development practices.
- **Backlog Maintenance**: Metrics measuring violations concerning entries in both the product and sprint backlogs.

- **Developer Productivity**: Metrics dealing with topics such as the workload of developers, how work is structured and how it is assigned.

Conformance metrics attempt to extract patterns in the collected data that do not comply with the defined practices. In practice, these are processes recommended by agile methodologies such as Scrum or XP. Furthermore, we included metrics which students frequently had issues with in past instalments of the Software Engineering II course, that diminish process adoption and student engagement, and are supported in the literature. For example, a user story that is overly long and does not fit on an index card and should be split, can be identified. These violations can reveal problem areas in the executed process, that need special attention. Metrics follow the iterative model described in Figure 3, adapted from Zazworka et al. [12]. First, conformance metrics are defined using a template, containing the minimum of information that is needed in order to execute the metric and interpret results. The template is given in Figure 2. In a second step, metrics are executed and violations are detected, down to the artifact level, i.e. the actual offending artifact is extracted. Third, the context of the detected violations is researched, i.e. the how, what and why questions surrounding the artifact are discussed with the team or relevant stakeholders.

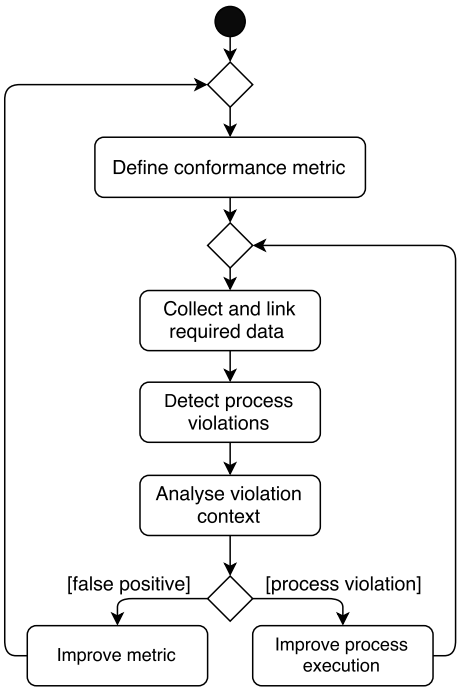


Figure 3: Activity diagram of the iterative lifecycle of conformance metrics.

Lastly, measures are taken to prevent future violations. In the case of false positives, i.e. artifacts that were detected as violations but are not considered problematic, the metric needs to be adapted. In the case of a correctly identified process violation that is considered severe enough to tackle, steps to ensure that the defined process is followed more closely in the future, e.g. by additional training or additional software tools, can be taken.

Using the proposed conformance metrics, violations for all teams in all sprints could be detected in the development data of the 2014/15 instalment of Software engineering II with 38 participants. As the specific artifact that caused a violations can be extracted by the metrics, additional research can take place to uncover the root cause of an issue. These root causes can then be tackled with additional tutor intervention. In some cases, severe violations were found that were missed by tutors, who took part in meetings. For example, a very complex, wrongly prioritized user story, that had been in the sprint backlog of all sprints. Being one user story among hundreds in the issue tracker that was being used it was missed by manual analyses.

Table 2: Conformance metric template. This template is filled for every metric that is to be measured. It contains all the information needed for an executable metric with interpretable results.

Name	The unique, descriptive identifier of the metric.
Synopsis	A short description of the type of violations the metric measures, e.g. "commits without tests".
Description	Overview of the expected process, i.e. the practice which should be followed, and its advantages, with references to literature. A description of what constitutes a violation of this process should be included.
Data sources	A list of data sources the metric requires and which the *query* is based on, e.g. code repositories or issue trackers.
Query	Steps needed to extract violations from the *data sources*. Ideally, these steps can be automated, e.g. as a database query.
Rating function	Function that maps detected violations into a numerical score, indicating the degree of mismatch between the executed process and the one detailed in the *description*.
Pitfalls	Description of what the metric does *not* measure, e.g. limitations or possible misconceptions about the results of the metric.
Categories	Topics in the domain of agile software development the metric attempts at measuring, e.g. "XP practices."
Effort	How much effort collecting violations and calculating a score requires. Either low, medium or high, e.g. using an automated process on existing data sources is "low" effort. Low effort metrics should be implemented first.
Severity	Importance in the context of the project's agile development process. How severe violations found by this metric are. Either informational, very low, low, normal or high.

As such, we see this data-driven approach using the developed metrics as an effective addition to the usually employed assessment techniques of agile teams.

3 Related Work

Due to the effort involved with analysing development data manually, multiple automatic tools, targeting specific areas, have been proposed. Examples include work by Johnson et al.[4], who proposed *Zorro*, a tool for finding violations of Test-Driven-Development practices or as well as tools for analysing development artifacts in respect to the PSP (Personal Software Process) [5]. However, with the rise of Software-as-a-Service solutions and a focus on web technologies, recent analysis tools, even in research, have moved away from native desktop applications. A good example

of this, as well as a good example of how process metrics are gaining traction is *Gitlab*,[1] a collaborative git repository hosting service on the web. Over time, Gitlab has included a full range of services, from issue tracking, code reviews, to continuous integration and continuous delivery. This gives them access to a wealth of development data for each process, along the deployment pipeline of a project. They used this data to develop process metrics called *Cycle Analytics*.[2] These measure the time it takes for an idea to pass through the different stages of a project, i.e. issue, plan, code, test, review, review and staging, starting with an issue, until code that implements the idea is deployed. These analysis are possible, as Gitlab has access to a variety of development artifacts. Without this diversity and the knowledge to link them, e.g. knowing that "Fixes#123" in a commit message relates to issue number 123, no times relevant for projects could be measured. A different approach is taken by *Commit Guru* by Rosen et al. [11]. The tool relies solely on version control information, taking a git repository on GitHub as input. Every commit is assigned a rating, placing it either in the "risky" or "not risky" categories. A risky commit is likely to contain bugs and should be reviewed. Whether a commit is determined to be risky or not is based on thirteen metrics that are calculated for each commit. These include very simple ones, such as the amount of deleted lines (higher is considered riskier) or the amount of commits the developer has recently made (less is riskier). However, some more complex metrics, such as subsystem developer experience, measuring the number of commits the developer made in the past to the subsystems touched by current commit (higher is less risky), show that meaningful statistics can already be generated from a single data source.

4 Future Work

Future work in the area of measuring process conformance by analysing development artifacts includes iterating and refining the employed metrics. As every development team is different and software development methodologies are meant to be adapted to a team's context, finding the differences in needed metrics is an interesting field of study. The related work can give very good starting points for own measurements, as these represent metrics that other teams deem relevant to their processes. Furthermore, future work will focus on how developers can be notified of possible violations. Providing a read-only web interface that needs to be regularly visited for updates, might not be the ideal solution. Providing notifications in a prompt fashion could allow new ways to interact with analysis tools, leading to greater engagement with developers.

[1] https://gitlab.com/ (last accessed 2016-10-20).
[2] https://about.gitlab.com/solutions/cycle-analytics/ (last accessed 2016-10-20).

5 Publications

Three papers were accepted in conferences in 2016:

- C. Matthies, T. Kowark, and M. Uflacker. "Teaching Agile the Agile Way – Employing Self-Organizing Teams in a University Software Engineering Course". In: *American Society for Engineering Education International Forum*. New Orleans, Louisiana: ASEE, 2016 [8].
- T. Kowark, C. Matthies, M. Uflacker, and H. Plattner. "Lightweight Collection and Storage of Software Repository Data with DataRover". In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE 2016. New York, NY, USA: ACM, 2016, pages 810–815. ISBN: 978-1-4503-3845-5. DOI: 10.1145/2970276.2970286 [6].
- C. Matthies, T. Kowark, M. Uflacker, and H. Plattner. "Agile Metrics for a University Software Engineering Course". In: *46th Annual Frontiers in Education Conference*. Erie, PA: FIE, 2016 [9].

References

[1] S. Augustine. *Managing agile projects*. Prentice Hall PTR, 2005.

[2] G. Campbell and P. P. Papapetrou. *SonarQube in Action*. Manning Publications Co., 2013.

[3] T. Chow and D.-B. Cao. "A survey study of critical success factors in agile software projects". In: *Journal of Systems and Software* 81.6 (2008), pages 961–971.

[4] P. M. Johnson and H. Kou. "Automated Recognition of Test-Driven Development with Zorro." In: *AGILE*. Volume 7. Citeseer. 2007, pages 15–25.

[5] P. M. Johnson, H. Kou, J. M. Agustin, Q. Zhang, A. Kagawa, and T. Yamashita. "Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH". In: *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*. Aug. 2004, pages 136–144. DOI: 10.1109/ISESE.2004.1334901.

[6] T. Kowark, C. Matthies, M. Uflacker, and H. Plattner. "Lightweight Collection and Storage of Software Repository Data with DataRover". In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE 2016. New York, NY, USA: ACM, 2016, pages 810–815. ISBN: 978-1-4503-3845-5. DOI: 10.1145/2970276.2970286.

[7] V. Mahnic. "Teaching Scrum through Team-Project Work: Students' Perceptions and Teacher's Observations". In: *International Journal of Engineering Education* 26 (2010), pages 96–110.

[8] C. Matthies, T. Kowark, and M. Uflacker. "Teaching Agile the Agile Way – Employing Self-Organizing Teams in a University Software Engineering Course". In: *American Society for Engineering Education International Forum*. New Orleans, Louisiana: ASEE, 2016.

[9] C. Matthies, T. Kowark, M. Uflacker, and H. Plattner. "Agile Metrics for a University Software Engineering Course". In: *46th Annual Frontiers in Education Conference*. Erie, PA: FIE, 2016.

[10] G. Melnik and F. Maurer. "A cross-program investigation of students' perceptions of agile methods". In: *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* May 2005, pages 481–488. DOI: 10.1109/ICSE.2005.1553593.

[11] C. Rosen, B. Grawi, and E. Shihab. "Commit Guru: Analytics and Risk Prediction of Software Commits". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pages 966–969. ISBN: 978-1-4503-3675-8. DOI: 10.1145/2786805.2803183.

[12] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider. "Are Developers Complying with the Process: An XP Study". In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM. 2010, page 14.

Supporting Program Comprehension Through Semantic Code Models

Toni Mattis

Software Architecture Group
Hasso-Plattner-Institut
toni.mattis@hpi.uni-potsdam.de

The source code of programs is written with a conceptual model in mind. To understand and modify code, a developer needs to recover this model, which can be a time-consuming task. We explore the use of recent data mining techniques that provide high-level insights guiding the developer. In this report, we summarize our current progress on using and improving probabilistic models for this task and explore future directions of research.

1 Introduction

According to Naur's notion of programming [13], programmers build a "theory of how certain affairs of the world will be handled by, or supported by, a computer program". To review, use, or modify source code, another programmer needs to re-enact the reasoning that lead to a particular implementation.

Programmers have several means to communicate their theory to other developers, especially their future selves. The most powerful and obvious means are *names*. By giving the name *user* to certain objects in the system, programmers simultaneously group the entirety of attributes (e.g. name, email address, profile picture) into a conceivable module (e.g. a class) and create an analogy to the real world, stating that actual people using the system are represented by the *user* class in the program. At the same time, naming is closely linked to the process of abstraction: Once a compound data structure has been called *rectangle* and given an attribute *width*, the program code can refer to *rectangle.width* rather than accessing the coordinates inside the underlying data structure and computing the difference of their x-values.

In code, concepts and abstractions emerge as usage patterns of certain names: A concept can be identified by co-occurring names (*rectangle, width, height, origin, ...*), while abstractions tend to use vocabulary from one concept (e.g. *origin*) in their interface and internally refer to concepts used for implementing the abstraction (*point, x, y, ...*). We will exploit these phenomena in section 2 and the following sections, where we use and adapt models from natural language processing. Since the meaning of code is not solely encoded in names, but also in the structure which relates these names to each other, we will look into techniques to recover meaningful structural relations and correlate them with concepts in section 2.3. The effectiveness of this approach does not only rely on the underlying model, but also how we communicate the recovered concepts to the programmer to evoke insight or guide attention. We

discuss possible solutions in section 2.4. Finally, we discuss a number of research questions for future work.

2 Code as Result of a Random Process

The observable data, such as names and structure in code and related artifacts, is the result of a series of random decisions. Our aim is to find models of random processes that generate the observed code with a high probability. If this is the case, our model is said to "explain" an observation. Unexplained variability is considered to be noise.

We first focus on a process which generates names. Assume there is a finite set of available names, i.e. the corpus of a language. A trivial random process would draw each word from that corpus with uniform probability, however, our observed data has no such distribution and therefore is unlikely under such model. Code exhibits local variation in how names and identifiers are distributed; it is clear that this variation is not just noise. It makes sense to explain this variation as the result of a statistical factor, which we call *concept* for now. Every concept has its own distribution of names, e.g. the *geometry* concept is likely to use "point", "rectangle", or "line", but less likely to use "password". The concept of *authentication* however is very likely to use "password". Each module can be seen as mixture of such concept distributions, a higher proportion of one concept biases the distribution of names for variables, methods, classes, attributes, etc. towards this particular concept's distribution.

A random process involving concepts could look like this:

- Generate C concepts, each concept ϕ_c being a vector of proportions over all possible names n with $\sum_n \phi_{cn} = 1$, i.e. ϕ_c is a *multinomial* distribution.
- For each code module m_i:
  - For each concept c, sample a proportion θ_{ic} that states how much this module is concerned with the respective concept. The vector θ_i is *multinomial* again.
  - For each position of a possible name n_{ij} in the module:
    * Sample a random concept indicator $c_{ij} = z$ with probability θ_{iz}.
    * Sample a random name $n_{ij} = m$ with probability ϕ_{zm}, i.e. the name distribution of concept z.

In literature, the framework according to which this process is being constructed is known as *Latent Dirichlet Allocation (LDA)* [7], as the involved multinomial distributions can itself be regarded as random samples from a Dirichlet distribution. A graphical representation of the full LDA model including the Dirichlet priors is given in Figure 2a.

From only observing n_{ij} we cannot directly compute the values of c_{ij}, ϕ and θ. We must treat them as *hidden variables* and our objective is to optimize them in a way that jointly maximizes the probability of our current observations n_{ij}.

The process described above makes some assumptions:

Assumption 1 *(Distributional Hypothesis)*: If a name occurs close to another name (e.g. in the same method), they are likely part of the same concept. This is expressed in step 2.a as the fact, that proportions of concepts θ_i are shared among all names inside a single module.

Assumption 2 *(Conditional Independence)*: Names are conditionally independent given their concept. This is a strong but useful simplification of how code actually works. We will relax this assumption later.

2.1 Mining Concepts

An example algorithm assigning each name a concept and thereby representing each module m_i of the code as mixture of concepts θ_i is given as follows. In this example, we consider a module being a single method in an object-oriented environment or function in a functional environment.

Let n_{ij} be the j-th name or identifier in the i-th method m_i, and $1 \leq c_{ij} \leq C$ be the concept assigned to the j-th identifier in method m_i. With $|n \rightarrow c|$ we denote the overall number of times name n has been assigned to concept c, and $|m \leftarrow c|$ the number of times concept c has been assigned to any position in method m. Consequently, $|\cdot \rightarrow c|$ denotes how often c has been selected overall, and $|m|$ is the number of identifiers in method m, N the total number of distinct names.

We start the algorithm with a fully random assignment and improve each assignment regarding the criteria and assumptions stated above. For each improvement, we randomly sample a new concept c_{ij} from the available range of concepts according to the following probability distribution over concepts. This is a so called Monte-Carlo algorithm:

$$P(c_{ij} = z) \propto \frac{|m_i \leftarrow z| + \alpha}{|m_i| + C\alpha} \cdot \frac{|n_{ij} \rightarrow z| + \beta}{|\cdot \rightarrow z| + N\beta} \tag{1}$$

The first factor represents the relative proportion of the concept within a method, θ_i. In accordance with assumption 1, more frequent concepts near a name are more likely to be selected as a concept. The second factor is the relative representation of the current name within each concept, thereby preferring concepts which have been associated with the name more frequently. This reflects the fact that concept-name-distributions ϕ_c are shared among all modules. The model parameters α and β smooth the distribution, avoid division by zero and assign non-zero probabilities for concepts that have never been seen.

This sampling for each n_{ij} should be repeated multiple times. After a few iterations, the per-module proportions of each concept $\theta_i = |m_i \leftarrow c|/|m_i|$ converge.

A simple extension to this model allows the number of concepts C to grow to best fit the code base. A practical implementation would allow a $(C+1)$-th concept to be sampled with non-zero probability, e.g. by discounting existing concepts by factor $|\cdot \rightarrow c|/(K+\gamma)$ and weighting the new concept $\gamma/(K+\gamma)$, with $K = \sum_i |m_i|$. The parameter γ controls the rate at which new concepts are introduced and thus inversely affects the number of identifiers assigned to each concept.

Relation to LDA and Dirichlet Processes Equation 1 can equivalently be described in terms of a Gibbs sampler [10] on the LDA model,[1] however we opted for a more intuitive description here. A sound, probabilistic interpretation of a dynamically growing number of concepts is given by a *Dirichlet process (DP)*, which is a distribution over infinitely many concepts of exponentially decreasing likelihood. A full description of the DP-based model and training algorithm is beyond the scope of this work, but similar schemes have been implemented as extensions to LDA-like topic models [6, 19].

Feature Calibration In contrast to natural language, identifiers in code are simpler features, since they are rarely inflected. However, they often appear in plural form, e.g. *user* may refer to a single user object, while a variable *users* might hold a list of user objects. In practice, removing the plural form has not had a significant impact on model performance, since both singular and plural form often co-occur. Most programming languages, however, form compound identifiers using camel-casing (`checkPassword`) or underscores (`check_password`) in mainstream languages, hyphens (`check-password`) in LISP-like languages, or selector parts (`checkPassword:ifInvalid:`) in Smalltalk. It is desirable to split these into atomic words.

Natural-language Context Often, code appears in the context of natural language, e.g. in documentation, *StackOverflow* posts, or by being accompanied by code comments. We can apply standard methods of NLP to obtain word features from this context, such as stemming and stop-word elimination, and treat them like names occurring in the code.

Temporal vs. Spatial Context We can easily change our definition of context from co-occurrence within the same module to co-modification within the same time frame. In this case, the above algorithm works on names that have been added or removed within a coherent modification of the code base. Such modifications typically have the form of patches or commits in a version control system like *Git* or *Mercurial*. We can run the above algorithm on these commits rather than modules from the latest version, and propagate hidden variables forwards if we can track the identity of a name across several versions, see Figure 1.

Using temporal over spatial context is intuitively more effective when commits have a finer granularity than just the modules in the latest version. Large commits can be split into their respective modules, so that a balance between spatial and temporal locality can be maintained. In any case, adding historical information increases statistical support for the concepts while raising the time to analyze a legacy code base significantly.

[1] For a more accurate estimate, the currently re-assigned concept should not be included in the aggregate counts, hence the term $|m_i|$ should also be corrected to $|m_i| - 1$.

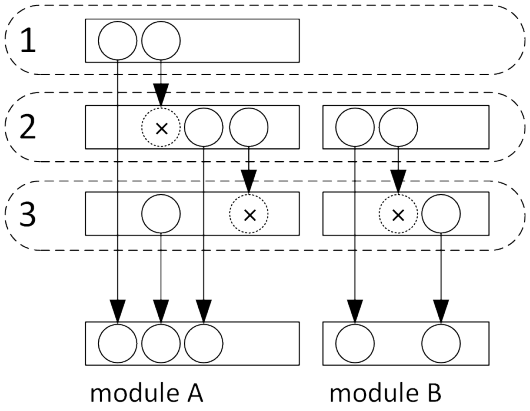


Figure 1: Two modules created over the course of three commits. Names introduced in one commit retain their identity and hidden variables until they are removed again. The hidden concept variables for each module are taken from the surviving names of earlier commits.

2.2 Abstraction-aware Concept Allocation

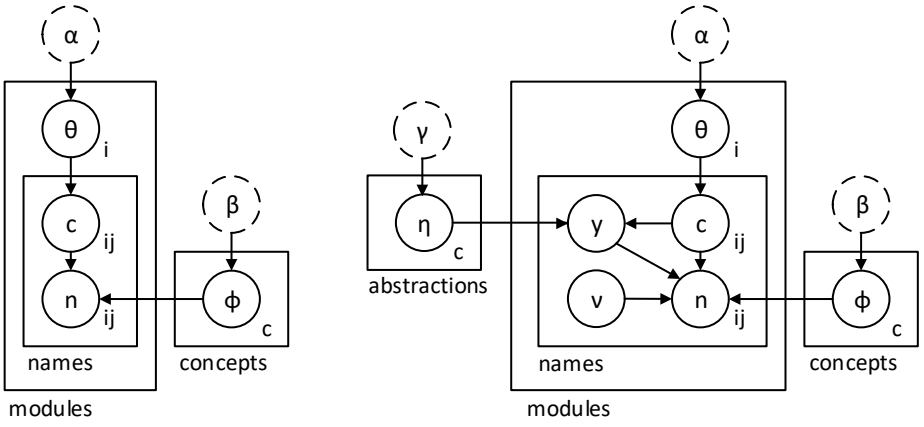
Concerning the above model, we reproduced and validated existing research on the LDA-framework and code. The simplicity of the underlying model does, however, only explain the distribution of names across the code base and yield a very simple notion of a *concept*. In ongoing research, we are extending the model to capture *abstraction*.

Abstractions are usually in place to separate higher-level concepts from their lower-level implementation. Externally, they manifest as interfaces that expose a domain-specific concept (e.g. finding a route between two locations on a map) and use names specific to this domain ("map", "road", "city"). Internally, they express parts of the protocol in terms of different concepts, e.g. invoking a graph library and find a shortest path, thereby using graph-specific names ("path", "edge", "vertex").

Implementation code usually mixes both sides of the abstraction as it maps domain behavior and state onto implementation concepts. We introduced a new binary factor $\nu_{ij} \in \{0, 1\}$ that states whether n_{ij} is an implementation detail (1) or just the abstraction (0), and a translation matrix η of size $C \times C$ which captures for each abstract concept the proportions of its implementing concepts, i.e. η_{cy} is the probability that concept c uses concept y in its implementation.

Our random process is modified as follows:

- Generate ϕ and θ as in the LDA model.
- For each position of a possible name n_{ij} in the module:
  - Sample a random concept indicator $c_{ij} = z$ with probability θ_{iz}.
  - Sample a Bernoulli-distributed indicator $\nu_{ij} \in \{0, 1\}$.
  - If $\nu_{ij} = 0$ (Name is from the abstraction)



(a) LDA-based concept model from literature

(b) Proposed abstraction-aware concept model

Figure 2: The generative models in plate notation. Arrows indicate dependencies between random variables, boxes indicate multiplicity, i.e. replicate the variables for each name, module or concept respectively. Dashed circles represent Bayesian priors.

    * Sample a random name $n_{ij} = m$ with probability ϕ_{zm} as in the LDA model.
- If $\nu_{ij} = 1$ (Name is from the implementation)
    * Sample a random implementation concept y with probability η_{zy}.
    * Sample a random name $n_{ij} = m$ with probability ϕ_{ym}.

Fitting the hidden variables of this model results in interesting insights, because the optimal allocation of the abstraction-implementation-matrix η now captures how concepts relate to each other. Additionally, the concepts itself are "cleaned up" compared to the LDA-like approach, since co-occurrence does not automatically imply relatedness within the same concept, but also across abstraction barriers.

A limitation of our optimization procedure is that it does not optimize the ν variables yet. We assume $\nu = 0$ for any name in the public interface of a class and method arguments, $\nu = 1$ otherwise. A consequence is that η has a strong diagonal, i.e. concepts often refer to themselves as their implementation.

Example Fitting the above model on the Smalltalk tool-building framework and IDE *Vivide* [18], we can obtain the results outlined in Table 1, two noisy concepts are omitted from the table. Some important connections have been properly detected, e.g. that the event system often refers to boxes (UI elements), which represent drop targets but at the same time can be drawn on a canvas. Also, the source code editing concept makes use of an underlying text and font rendering concept.

Table 1: Abstraction-aware concepts fitted on the Vivide tool-building framework. $C = 10, \alpha = 1.0, \beta = 1.0, \gamma = 1.0, \approx$ 60,000 LOC. Implementation concepts with $\eta_{cy} < 0.08$ omitted.

Concept	Top 5 names	Impl. concepts
1	`color canvas bounds draw width`	
2	`update box target emit drop`	1
3	`event mouse selector signal sut`	2
4	`form world context active vivide`	
5	`text string edit label font`	
6	`pane script source code service`	5
7	`next result size first all`	
8	`icon method icons add toolbar`	4

2.3 Code Structure beyond Names

A logical extension to the name-based approach is to consider structure of the underlying code. Typically, programs can be represented as an *abstract syntax tree (AST)*. Our previous models can be interpreted as explaining a subset of leaf nodes in an AST, e.g. variable and method-name nodes. In order to represent more than just leaf nodes, we require a set of rules which generate structure. Such a rule set is given by a *context-free grammar* of our language. If we attach probabilities to each alternative rule, the resulting probabilistic context-free grammar *(pCFG)* is a generative process for describing fully structured code.

However, a simple pCFG cannot express structures larger than a single rule application. In natural language processing and pattern mining, this deficiency is addressed by adding redundant rules to the pCFG which resolve to tree fragments [3, 17]. Such a construction is called *probabilistic Tree Substitution Grammar (pTSG)*. An example

E	$\rightarrow$	'(' E A ')'	$p = 0.3$
E	$\rightarrow$	Literal	$p = 0.7$
A	$\rightarrow$	E A	$p = 0.3$
A	$\rightarrow$	ϵ	$p = 0.7$

(a) A pCFG for LISP-like expressions

E	$\rightarrow$	'(' E A ')'	$p = 0.3$
E	$\rightarrow$	Literal	$p = 0.6$
*E	$\rightarrow$	'foo'	$p = 0.1$*
A	$\rightarrow$	E A	$p = 0.2$
A	$\rightarrow$	ϵ	$p = 0.3$
*A	$\rightarrow$	E	$p = 0.5$*

(b) A pTSG including two redundant substitutions (*) with separate probabilities.

Figure 3: A pCFG and a derived pTSG. *E* represents an expression, *A* a recursively defined list of arguments.

can be seen in Figure 3. The decision which tree fragments should be substituted in a single rule is equivalent to mining "interesting" subtrees in a training set of trees. Such a process differs from *frequent subtree mining*, as it maximizes the probability of the observed trees under the pTSG rather than just collecting frequently repeating patterns, which tend to be rather small.

Finding subtrees Post and Gildea [17] have derived an algorithmic approach to isolate probability-maximizing fragments from a set of trees. The idea is to place a random *split tokens* $z_n \in 0, 1$ at every node n, which decide where the tree should be split into fragments.

The algorithm tends to split trees and fragments into smaller fragments if the isolated fragments are more probable or have been observed more often than the joined tree. Otherwise, fragments get joined again. This is achieved by iteratively visiting each node n and randomly re-sampling the split token z_n according to the following probability distribution:

$$P(z_n = 0) = \frac{P(T_{join})}{P(T_{join}) + P(T_s)P(T_t)} \tag{2}$$

where T_{join} is the tree that results if the parent tree fragment and the fragment originating at n are joined, while T_t is just the parent fragment including n, and T_s is the child fragment rooted at n. The probability over a tree fragment is given by:

$$P(T) = \frac{\#T + \lambda P_0(T)}{\#R_T + \lambda} \tag{3}$$

Where $\#T$ is the number of times fragment T has been seen over all trees, and $\#R_T$ how often the root node of the fragment has been encountered. P_0 is a prior distribution, which usually decreases for larger tree fragments, and λ determines how much this prior is valued during sampling. An illustration of the involved components can be seen in Figure 4.

Technically, this algorithm can iterate forever, generating an asymptotically growing amount of tree fragments. However, as the algorithm runs, we can observe certain fragments repeating more frequently than others, these are candidates to augment our pTSG. Their relative frequency gives an estimate for the probability their respective rule would have assigned.

Concept-specific Subtrees The above mechanism only finds global *idioms*. When we introduce our hidden variable for concepts again, we can explain local variation in the frequency of certain tree fragments as mixture of concepts, each of them bearing an own distribution of tree substitutions. Most variation is observed in the rules for generating leaf nodes, especially those for method and variable names, but certain concept-specific idioms also emerge.

This is a novel extension to LDA-like models as well as pTSG-based models. The generalization of the abstraction-aware model to trees is yet unclear, as well as the integration of natural language context.

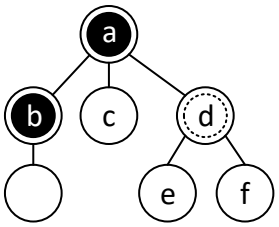


Figure 4: State of the pTSG mining algorithm, black nodes have $z = 1$, white nodes $z = 0$. For the choice whether z_d should split the tree, we consider frequencies of $T_t = \{a, b, c, d\}, T_s = \{d, e, f\}$ and $T_{join} = \{a, b, c, d, e, f\}$.

2.4 Applications

Models capable of explaining code in terms of concepts and their relations can support program comprehension in a multitude of scenarios.

Navigation and Recommendation Traditional code manipulation environments support navigation by senders or implementors of a method or message, class hierarchy and namespaces or packages. Concept-aware environments can provide additional navigation directions to locate code that is related through a specific concept, or code that is similar as it shares the same distribution of concepts.

General information needs, e.g. code completion and examples of how to use certain API, can be answered with relevant results scoped to the current task of the developer.

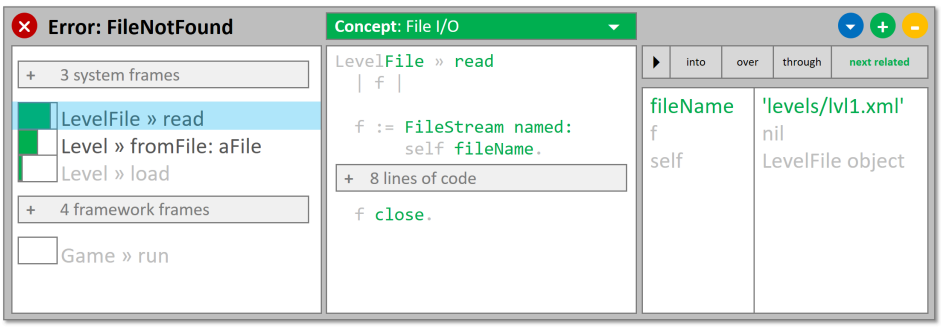


Figure 5: A debugger in Squeak/Smalltalk fixed to a single concept. Relatedness is displayed as green bars and highlighting, unrelated code is hidden from the developer's view until explicitly navigated. Debugging can step through related code, e.g. halting at the *close* statement after opening the file.

Focus of Developer Attention Knowing which concept some code belongs to enables tools to focus on exactly this concept, e.g. a debugger responding to a *FileNotFound* exception can highlight file-related code while hiding code and stack frames that have nothing to do with the file concept. Boilerplate code not relevant for understanding certain core concerns can be grayed out to reduce distraction. A prototypical example of a debugger that can be scoped to certain concepts can be seen in Figure 5.

3 Related Work

A number of non-probabilistic techniques have been proposed to identify concepts, mostly from the domain of *aspect mining*. These include clustering algorithms with varying similarity measures [15], random walks [21], run-time data [9, 20], version history [8], or formal concept analysis [16].

More recently, language models have been used for source code mining, notably graph-based approaches [14] and n-gram models at large scale [2]. They are successful in explaining fine-grained structures in source code in terms of the surrounding context.

Probabilistic code models have been constructed for a variety of tasks, such as finding cross-project concepts on a large scale using LDA [5, 11], summarizing source code using a hierarchical topic model [12], modularity assessment and refactoring recommendations using the relational topic model (RTM) [4], learning coding conventions using a log-bilinear model [1], and mining language idioms using the pTSG model [3].

4 Future Work

UI Integration So far, most work has been data-oriented and theoretical. The next steps should be to explore how concepts can be linked to every-day tools and new tools used by programmers. As a starting point, we currently investigate the workflow of reviewing student's projects, where the reviewer is not familiar with the code base but required to quickly identify the core concerns and their implementation from scratch. For projects based on a legacy code base, a concept-level overview of changes compared to a baseline is desirable. Relating authors to concepts hints at the knowledge distribution among team members.

Expert Feedback A concept model still contains some noise and may not always guess the correct allocation of concepts. Hence, we should explore ways to correct the model's propositions if the developer recognizes an error in the model. This also raises the question of sharing the same model across different authors in a way that inexperienced developers can immediately benefit from expert input.

Model Evaluation Code and concept models can be evaluated in two different stages: The model itself yields a probability distribution over the code base, which gives an indicator of how well the model explains code. We can see that this probability increases for previously unknown code when we add "meaningful" variables, such as the differentiation between abstraction and implementation. However, we also need to consider whether the resulting concepts benefit the user, which requires both a well-designed UI, a set of developer tasks and metrics and a user study set-up.

Scaling We currently have access to approximately 10 TiB of source code, including history and process artifacts. Being in a phase of trial-and-error experimentation, we are working with subsets of several thousands LOC up to a few millions, and history of not more than 70,000 commits. We are facing a trade-off: The insightful tree-based and abstraction-based models scale poorly because parsing and tree-based operations are inherently expensive, while LDA-like models are easier to scale. Both directions hold interesting research questions. A middle way worth investigating is bootstrapping using a simple background model mined at scale, but using the complex model for fine-grained analysis in a project-specific context.

5 Conclusion

Probabilistic models derived from either the LDA framework or the pTSG approach are a good starting point for recovering concepts from source code. The structure of these models allows them to be mined using iterative, at any time interruptible Monte-Carlo algorithms. By addressing specific structural properties of code, we can improve and combine both types of models. Apart from practical challenges, like scaling issues, the impact on the developer workflow and the design space for tools is yet to be explored.

References

[1] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. "Learning Natural Coding Conventions". In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. New York, NY, USA: ACM, 2014, pages 281–293. ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635883.

[2] M. Allamanis and C. Sutton. "Mining Source Code Repositories at Massive Scale Using Language Modeling". In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pages 207–216. ISBN: 978-1-4673-2936-1.

[3] M. Allamanis and C. Sutton. "Mining Idioms from Source Code". In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. New York, NY, USA: ACM, 2014, pages 472–483. ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635901.

[4] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. de Lucia. "Improving Software Modularization via Automated Analysis of Latent Topics and Dependencies". In: *ACM Trans. Softw. Eng. Methodol.* 23.1 (Feb. 2014), 4:1–4:33. ISSN: 1049-331X. DOI: 10.1145/2559935.

[5] D. Binkley, D. Heinz, D. Lawrie, and J. Overfelt. "Understanding LDA in Source Code Analysis". In: *Proceedings of the 22Nd International Conference on Program Comprehension*. ICPC 2014. New York, NY, USA: ACM, 2014, pages 26–36. ISBN: 978-1-4503-2879-1. DOI: 10.1145/2597008.2597150.

[6] D. M. Blei, T. L. Griffiths, and M. I. Jordan. "The Nested Chinese Restaurant Process and Bayesian Nonparametric Inference of Topic Hierarchies". In: *J. ACM* 57.2 (Feb. 2010), 7:1–7:30. ISSN: 0004-5411. DOI: 10.1145/1667053.1667056.

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent Dirichlet Allocation". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pages 993–1022. ISSN: 1532-4435.

[8] S. Breu and T. Zimmermann. "Mining Aspects from Version History". In: *21st IEEE/ACM International Conference on Automated Software Engineering, 2006. ASE '06.* Sept. 2006, pages 221–230. DOI: 10.1109/ASE.2006.50.

[9] S. Breu and J. Krinke. "Aspect Mining Using Event Traces". In: *Proceedings of the 19th IEEE International Conference on Automated Software Engineering*. ASE '04. Washington, DC, USA: IEEE Computer Society, 2004, pages 310–315. ISBN: 978-0-7695-2131-2. DOI: 10.1109/ASE.2004.12.

[10] T. Griffiths. *Gibbs Sampling in the Generative Model of Latent Dirichlet Allocation*. 2011. URL: https://people.cs.umass.edu/~wallach/courses/s11/cmpsci791ss/readings/griffiths02gibbs.pdf (last accessed 2016-10-01).

[11] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. "Mining Concepts from Code with Probabilistic Topic Models". In: *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*. ASE '07. New York, NY, USA: ACM, 2007, pages 461–464. ISBN: 978-1-59593-882-4. DOI: 10.1145/1321631.1321709.

[12] P. W. McBurney, C. Liu, C. McMillan, and T. Weninger. "Improving Topic Model Source Code Summarization". In: *Proceedings of the 22Nd International Conference on Program Comprehension*. ICPC 2014. New York, NY, USA: ACM, 2014, pages 291–294. ISBN: 978-1-4503-2879-1. DOI: 10.1145/2597008.2597793.

[13] P. Naur. "Programming as Theory Building". In: *Microprocessing and Microprogramming* 15.5 (May 1985), pages 253–261. ISSN: 0165-6074. DOI: 10.1016/0165-6074(85)90032-8.

[14] A. T. Nguyen and T. N. Nguyen. "Graph-Based Statistical Language Model for Code". In: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pages 858–868. ISBN: 978-1-4799-1934-5.

[15] T. T. Nguyen, H. V. Nguyen, H. A. Nguyen, and T. N. Nguyen. "Aspect Recommendation for Evolving Software". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. New York, NY, USA: ACM, 2011, pages 361–370. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985843.

[16] D. Poshyvanyk, M. Gethers, and A. Marcus. "Concept Location Using Formal Concept Analysis and Information Retrieval". In: *ACM Trans. Softw. Eng. Methodol.* 21.4 (Feb. 2013), 23:1–23:34. ISSN: 1049-331X. DOI: 10.1145/2377656.2377660.

[17] M. Post and D. Gildea. "Bayesian Learning of a Tree Substitution Grammar". In: *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*. ACLShort '09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pages 45–48.

[18] M. Taeumel, T. Felgentreff, and R. Hirschfeld. "Applying Data-Driven Tool Development to Context-Oriented Languages". In: *Proceedings of 6th International Workshop on Context-Oriented Programming*. COP'14. New York, NY, USA: ACM, 2014, 1:1–1:7. ISBN: 978-1-4503-2861-6. DOI: 10.1145/2637066.2637067.

[19] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. "Hierarchical Dirichlet Processes". In: *Journal of the American Statistical Association* 101.476 (2006), pages 1566–1581. ISSN: 0162-1459.

[20] P. Tonella and M. Ceccato. "Aspect Mining through the Formal Concept Analysis of Execution Traces". In: *11th Working Conference on Reverse Engineering, 2004. Proceedings*. Nov. 2004, pages 112–121. DOI: 10.1109/WCRE.2004.13.

[21] C. Zhang and H.-A. Jacobsen. "Efficiently Mining Crosscutting Concerns Through Random Walks". In: *Proceedings of the 6th International Conference on Aspect-Oriented Software Development*. AOSD '07. New York, NY, USA: ACM, 2007, pages 226–238. ISBN: 978-1-59593-615-8. DOI: 10.1145/1218563.1218588.

Large graph exploration

Davide Mottin

Knowledge Discovery and Data Mining
Hasso-Plattner-Institut
davide.mottin@hpi.de

The increasing interest in social networks, protein-interaction, and many other types of networks has raised the question how users can explore such large and complex graph structures in an intuitive way. Nowadays, these networks count billions of nodes and relationships and are used to study complex phenomena, such as social behaviors, marketing campaigns, and economic factors. Given this complexity and size, interactive algorithms would assist the human in finding interesting information in graphs easily. Interactive algorithms and exploratory methods have been studied for more traditional data, such as relation, semi-structured, and textual data, to allow intuitive data exploration. However, there is no counterpart for graphs.

We propose graph exploration, a novel ground of research that includes the user in the algorithms to allow interactive access to large and complex graphs. We pioneer graph exploration to overcome the limitations of the previous work by proposing efficient, interactive algorithms that cope with the big data volume.

1 Overview

Graph exploration refers to systematic approaches in which the user can understand the data without issuing complicated queries. As opposed to previous research [6] we focus on graphs for which no data exploration solution currently exists.

The work I have done in this area in the last year is articulated into three parts:

1. **Discovering notable characteristics among nodes in knowledge graphs**: This work has been done during the last Summer Semester's Master Project and has been submitted for publication to the Web Search and Data Mining conference. The purpose was the study of methods to compare nodes in a graph for the automatic discovery of non trivial characteristics (differences in edge labels, attributes) among nodes representing entities, such as persons, places, organizations.

2. **Graph exploration proposals**: I submitted two proposals for graph exploration. The first envisions the study of methods to interactively explore graphs specifying a small set of nodes and explaining portions of it by means of relationships and attributes. The second instead focuses on failing queries, i.e., queries in which the user specifies a need but the results are unsatisfactory.

3. **Tutorial in graph exploration**: I will present along with prof. Müller and Anja Jentzsch from the Information Systems group a tutorial on graph exploration, showing the results of our literature review in the field of graph exploration.

We propose a novel taxonomy of concepts to categorize the research in this area and allow for a more natural positioning. Given the initial stage of research in this area, most of the challenges are still open.

In this report, I describe in detail the work done in the Master Project regarding the discovering of notable characteristics. I also briefly introduce the tutorial on graph exploration and the proposals.

2 Discovering notable characteristics among nodes in knowledge graphs

Consider the case in which a user wants to compare nodes in the graph and see how they differ from their similars. This is the case of a student searching for differences in two or more presidents, or a biologist looking at two microorganisms. Traditionally, the user would look at the nodes, their relationships and attributes and select those that are more interesting. However, this painful approach requires a long time and would lead to scarce results or errors. In this work, we propose a novel formulation to discover what we call *notable characteristics* given an initial set of nodes. While the traditional comparison of nodes by means of node similarity provides only a score with no explanation, we go one step further. We propose a solid probabilistic approach that first retrieves nodes that are similar to the seed provided by the user, and then exploits distributional properties to understand whether a particular attribute is interesting or not. We experimentally evaluate the effectiveness of our approach and show that we are able to discover notable characteristics that are indeed interesting and relevant for the user.

A knowledge graph is a directed graph in which nodes and edges have labels or types. They are also known as information networks [14, 16] or simply labeled graphs. We are given a set $\mathcal{A}$ of node labels and a set $\mathcal{L}$ of edge labels. The term label and type are used interchangeably.

Definition 1 (Knowledge graph)**.** *A knowledge graph is a quadruple* $G : \langle \mathcal{V}, \mathcal{E}, \phi, \psi \rangle$*, where* $\mathcal{V}$ *is a set of nodes,* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *is a set of edges,* $\phi : \mathcal{V} \mapsto \mathcal{A}$*,* $\psi : \mathcal{E} \mapsto \mathcal{L}$ *are node and edge labeling functions, respectively.*

Recall that we are interested in discovering *notable characteristics* of the entities mentioned in a set of input nodes in relation to their similars. This intuitive definition entails two questions: (1) what is the set of similars? (2) what are the notable characteristics?

The set of input nodes is referred to as *seed set* (seeds in short). Formally, given a knowledge graph $G : \langle \mathcal{V}, \mathcal{E}, \phi, \psi \rangle$ the set of seed nodes is any set $S \subseteq \mathcal{V}$. The seed set is manually provide by the user and therefore considered reasonably small (i.e., ≤ 10 elements). Given any set of seed nodes, we need to define a set of similars or *context nodes*. We assume the existence of a similarity function $\sigma : \mathcal{V} \times 2^{\mathcal{V}} \mapsto \mathbb{R}$ that assigns a high score to nodes that are similar to those in the seed set and low otherwise. Then, the context are the top-k most similar nodes.

Definition 2 (Context set)**.** *Given a knowledge graph* $G : \langle \mathcal{V}, \mathcal{E}, \phi, \psi \rangle$*, a seed set* $S \subseteq \mathcal{V}$*, a similarity function* $\sigma : \mathcal{V} \times 2^{\mathcal{V}} \mapsto \mathbb{R}$*, and a parameter* k*, the* context set *(or simply context) is a set* $C \subseteq \mathcal{V}$ *such that* $S \subseteq C, |C| = k$*, and for each* $n_c \in C \wedge n \in \mathcal{V} \setminus C, \sigma(n, S) \leq \sigma(n_c, S)$*.*

The second question concerns the notable characteristics. The characteristics are attributes or relationships of a specific node since they implicitly represent a signature of the node itself. As before, we assume the existence of a generic discrimination function, whose role is to return a score whether a specific characteristic is discriminative or unexpected comparing two set of nodes. Formally, in the knowledge graph G, a discrimination function $\delta : \mathcal{L} \times 2^{\mathcal{V}} \times 2^{\mathcal{V}} \mapsto \mathbb{R}_0^+$ assigns a discrimination value or 0 if the value is not discriminative. We are now ready to define a notable characteristic.

Definition 3 (Notable characteristic)**.** *Given a knowledge graph* $G : \langle \mathcal{V}, \mathcal{E}, \phi, \psi \rangle$*, a seed set* $S \subseteq \mathcal{V}$*, a context* $C \subseteq \mathcal{V}$*, and a discrimination function* $\delta : \mathcal{L} \times 2^{\mathcal{V}} \times 2^{\mathcal{V}} \mapsto \mathbb{R}_0^+$ *a* notable characteristic *is a relationship* $l \in \mathcal{L}|_C$ *such that* $\delta(l, S, C) \neq 0$*.*

The notation $\mathcal{L}|_C$ denotes the set of edge labels restricted to those that are found in the edges directly connected to C, i.e., $\mathcal{L}|_C = \{l \mid \exists x \in C, y \in \mathcal{V} \text{ s.t. } (x, y) \in \mathcal{E} \wedge \psi(x, y) = l\}$.

The general problem we aim to solve is efficiently returning the notable characteristics, given a seed set, a similarity function and a discrimination function.

Problem 1 (Finding notable characteristics)**.** *Given a knowledge graph* $G : \langle \mathcal{V}, \mathcal{E}, \phi, \psi \rangle$*, a seed set* $S \subseteq V$*, a similarity function* $\sigma : \mathcal{V} \times 2^{\mathcal{V}} \to \mathbb{R}$ *and a discrimination function* $\delta : \mathcal{L} \times 2^{\mathcal{V}} \times 2^{\mathcal{V}} \mapsto \mathbb{R}_0^+$*, find the set of notable characteristics.*

The problem entails the definition of suitable functions σ and δ that are able to retrieve and compare nodes.

2.1 Approach

We model the discrimination function in probabilistic terms, in order to better deal with noisy settings and uncertainty. Therefore, we assume that a characteristic is interesting if its distribution in the seed set deviates from the one in the context set. In other words, the context represents the expected behavior of the population while the seed is the hypothesis to be tested.

Given the seed set S, the first step requires the definition of a similarity function σ to retrieve a set of context nodes. Although many notions of similarity functions have been developed, such as structural equivalence [15] and SimRank [8], none seems suitable to our case. Existing similarity measures are either based on restricted neighborhoods of the nodes [15], or they disregard edge and node labels [8]. We propose an algorithm that takes into account the kind of connections between pairs of nodes and combines the advantages of random walk and metapath approaches.

In the traditional random walk model, a random walker chooses one of the outgoing edges from a node with uniform probability. Instead of uniform probability, we favor choices which are more informative in terms of edge label. Intuitively, an

edge label is informative if it has low frequency. This intuition is supported by information theoretic notions, such as tf-idf and has been successfully used in graphs as well [18]. As a shorthand notation, we define $\mathcal{E}_l$ as the set of edges having label $l \in \mathcal{L}$, i.e., $\mathcal{E}_l = \{(i,j) \in \mathcal{E} | i,j \in \mathcal{V}, \psi(i,j) = l\}$. The frequency of a label l is the fraction of l-labeled edges with respect to the total number of edges. We then define the weighted adjacency matrix as a $|V| \times |V|$ square matrix, where the value A_{ij} between node i and j is defined as

$$A_{ij} = \begin{cases} 1 - |\mathcal{E}_l|/|\mathcal{E}| & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The Personalized PageRank is defined as the vector

$$\mathbf{p} = c * \tilde{A} * \mathbf{p} + (1-c) * \mathbf{v}, \tag{2}$$

where $\tilde{A}_{ij} = A_{ji} / \sum_k A_{jk}$, c is the damping factor, and $\mathbf{v}$ is vector called personalization vector. In our experiments the damping factor is 0.8, in line with previous works. We compute the PageRank starting from each node in the seed set to retrieve the k nodes with the highest score. This is done by setting $\mathbf{v}_n = 1$ for each $n \in S$, individually. We refer to this baseline as RandomWalk.

However, the RandomWalk baseline disregards common connections between the seed nodes. This is an important information, since the tf-idf approach does not consider the user's similarity notion implicitly contained in the seed set. To this end, we adopt the notion of metapath from [23] which generalizes the concept of path. A metapath for a path $\langle n_1, ..., n_t \rangle, n_i \in \mathcal{V}, 1 \leq i \leq t$ is defined as the sequence $\langle \phi(n_1), \psi(n_1, n_2), ..., \psi(n_{t-1}, n_t), \phi(n_t) \rangle$ that alternates node and edge labels along the path. We mine metapaths running PathMining [13] from the seed nodes. Differently from the original PathMining, we start from several nodes with uniform probability and consider only edge types. Our algorithm stops the exploration when another seed node is encountered. The metapaths and the counts for each path are separately stored to compute the similarity score.

Once the metapaths are computed, we compute a score for each node based on the probability that some metapath starting from a seed node ends in this node. Given the set of metapaths M obtained with our modified PathMining, we denote as $n \overset{m}{\rightsquigarrow} n'$ any path from n to $n' \in \mathcal{V} \setminus S$ matching metapath $m \in M$. Therefore, the score of a node $n' \in \mathcal{V} \setminus S$ with respect to any seed node $n \in S$ is

$$\sigma(n', S) = \sum_{m \in M, n \in S} \frac{|\{n \overset{m}{\rightsquigarrow} n'\}|}{|\{n \overset{m}{\rightsquigarrow} n'' | n'' \in \mathcal{V} \setminus S\}|} \Pr(m)$$

$\Pr(m)$ is the probability of choosing metapath m, which is the relative count computed previously divided by the sum of the counts of all metapaths M. Intuitively, σ gives a higher score to nodes that are reachable through frequent metapaths connecting the seed nodes or connected through many of these metapaths. This means that nodes that are reached from infrequent metapaths will have a low score. Once we have computed the score for each node we return the k nodes with the highest score as our context.

Assume that we have computed the distribution of values for each characteristic (i.e., edge label) for both seed nodes and context nodes found with the previous method. Intuitively, for each characteristic, the distribution of the context represents the expected, or normal behavior, the one to compare with. Therefore, the seed set becomes the hypothesis to be evaluated against the "true" distribution of the context.

Formally, for each characteristic $l \in \mathcal{L}$, we consider two distributions in order to evaluate its notability. The first evaluates the number of occurrences of a distinct node label instance connected through a specific edge label (e.g., bornIn, California). This expresses information about the actual attribute values of the nodes and can be used to identify cases where different attribute values are relevant. For instance, most people in the seeds are born in the U.S., while those in the context are equally born in the U.S. and Europe. We refer to these distributions as *instance distributions*.

$$Inst_s(l, C, S) = (x_1, x_2, ..., x_m),\ Inst_c(l, C) = (y_1, y_2, ..., y_m)$$

where x_i and y_i are the number of occurrences of node i at the end of an edge labeled l from a node in S and C, respectively. Note that both vectors have the same size, so x_i is zero if i appears only in the context. Similarly, a second distribution computes aggregates over the number of occurrences of a specific edge type in the context. This expresses information about the existence and cardinality of an attribute and can be used to identify cases where attribute cardinality is relevant. For instance, people in the seed all have a single child while in the context most have two children. Such cases cannot evidently be modeled as instance distributions that take into account distinct values. We refer to these distributions as *cardinality distributions*.

$$Card_s(l, C, S) = (x_1, x_2, ..., x_m),\ Card_c(l, C) = (y_1, y_2, ..., y_m)$$

where x_i and y_i are the number of times a node in S and C respectively has i edges labeled l.

Both distributions can be built by iterating through the nodes in each set and counting the respective occurrences. For a given $l \in \mathcal{L}$, this results in two scores δ_{Inst} and δ_{Card} for instance and count distributions. The final score δ is a maximum aggregation score between δ_{Inst} and δ_{Card}.

$$\delta(l, C, S) = \max(\delta_{Inst}(l, C, S), \delta_{Card}(l, C, S)) \tag{3}$$

Many measures have been proposed in statistics to compare two distributions. However, most of them draw specific assumptions, such as a minimum number of samples or non-zero probabilities, that are not fulfilled in our case. In particular, *Inst* and *Card* have no natural ordering and no distance-function between the values. Additionally, we compare a m-sized distribution over our context, where m is the size of the context, to a much smaller distribution over the seed-nodes. This leads to many zero values in the seed-distribution.

We resorted to a more natural multinomial test that better expresses the relationship between our distributions. The multinomial test assumes that a set of observations is drawn from a known multinomial distribution. Therefore, assuming the context to be Multinomial distributed the observations are the values found in the

seed set. If the values observed in the seed sets are drawn from the Multinomial, than the hypothesis cannot be rejected and the characteristic is marked as non-notable. On the other hand, if the test succeeds, then the two distributions are significantly different and the characteristic is notable.

Assume we have a random variable $X_{N,\pi} \sim Mult(N, \pi)$, with parameters N and π. We normalize $Inst_c$ and $Card_c$ to express a probability distribution $\pi = normalize(y) = (\pi_1, \pi_2, ..., \pi_k)$. For a given outcome $x = (x_1, x_2, ..., x_k)$, the probability, under the hypothesis of equality between context and sample, is

$$\Pr(X_{N,ß} = x) = N! \prod_{i=1}^{k} \frac{\pi_i^{x_i}}{x_i!},$$

where $N = \sum x_i$. In an exact multinomial test, the significance probability is

$$\Pr_s(X_{N,\pi} = x) = \sum_{y:\Pr(X_{N,\pi}=y) \leq \Pr(X_{N,\pi}=x)} \Pr(X_{N,\pi} = y).$$

$\Pr_s(\pi, x)$ is the probability of x or any equally or less likely outcome being drawn from the probability distribution. In case of large N, the exact test is impractical, we therefore perform a Montecarlo sampling to approximate the final result. A difference in distributions is considered significant, if the hypothesis is rejected with probability $p > 0.95$.

2.2 Related Work

Previous work mostly concerns the discovery of similar nodes or groups of individuals sharing common interests (graph clustering). Among them node comparison has a long history in graph analytics. Early methods include graph edit distance (GED) [1], structural equivalence [15], SimRank [8]. Random walk approaches, such as Personalized PageRank [2] and HITS [11] can also be used to find nodes similar to the input nodes. Node comparison measures can only return whether one node is similar or different from another but they cannot readily adapt to the discovery of notable characteristics, since the score provides no insight on the discovery process. Additionally, these methods do not consider whether similarities or differences are meaningful with respect to a "normal" state other than total equivalence.

Seed set expansion refers to methods that ask the user to provide an initial set of entities or structures and retrieve similar nodes. In graphs, the seed set can be composed of either structures (i.e., subgraphs) or nodes. Exemplar queries paradigm [18, 19] assumes that the user input is an example of the intended results. Similarly, GQBE [7] considers entity tuples to find similar other tuples in a knowledge graph.

Seed nodes are used to discover groups of nodes with similar characteristics [12, 20]. These seed-based clustering algorithms exploit the specificity of each node in the seed set to return ad-hoc communities. Although these methods provide multiple groups of nodes they cannot properly explain the characteristics and the differences among them; in general, they do not directly compare the seed nodes with the others.

3 Graph exploration (Tutorial)

The graph exploration tutorial has been accepted for presentation in the international Conference on Information and Knowledge Management (CIKM). This report provides an outline of the tutorial.

The continuously increasing interest in graphs and the growing amount of graph data available on the web require a careful design of data analysis techniques. However, from the user perspective most of the existing techniques appear as a black box that returns results without any explanation. For these reasons our community has resorted to data exploration techniques. In particular, while a huge effort has been devoted to text, relational, and semi-structured data [6], data exploration on graphs (*graph exploration* in short) is still in its infancy. Although many techniques for graphs have been studied in different domains, there is still lack of a unified graph exploration taxonomy. We abstracted user-driven graph exploration properties from techniques proposed in the literature and defined such a unified taxonomy. Our taxonomy consists of three strategies that form the backbone of our presentation along with relevant literature identified so far:

Exploratory Graph Analysis entails the process of casting an incomplete or imperfect pattern query to let the system find the closest match. Such exploratory analysis may return a huge number of results, e.g., structures matching the pattern. Thus, the system is required to provide intelligent support. One such strategy is the well known query-by-example paradigm, in which the user provides the template for the tuples and let the system infer the others.

Refinement of Graph Query Results is needed to deal with the overwhelming amount of results that is typical in subgraph processing. It includes approaches designed to present comprehensive result sets to the user or intermediate results that can be refined further. Instantiations of this kind are graph summaries, top-k methods, query reformulation, and skyline queries.

Focused Graph Mining guides the users to a specific portion of the graph they are interested in. It requires the user to provide feedback in the process to restrict the computation to some portion of the graph. Ego-networks mining belongs to this strategy, since the user search is limited to a particular area of the graph and the algorithms focus on that specific area.

4 Related Work

Graph exploration is a novel ground for research. We briefly survey the most important previous research that have been proposed in this area. Exploratory analytics in graphs include approximate search in which some research term is unspecified or incomplete. Alternatives to subgraph isomorphism base on similarity search have been proposed [10, 25, 26]. An alternative approach is the by-example paradigm in

which the input query is a tuple or an object of interest that is part of the intended result set [7, 18].

While exploratory analytics try to find results that are as close as possible to the input query, the refinement of graph query results propose alternative queries to the user. If the result set is excessively large refined queries are proposed [17, 24]. On the other hand the query may be too restrictive and return no result. In this case the query conditions have to be relaxed (i.e., removed) [24]. Orthogonal to query refinement is the top-k approach, in which the k results closest to the query are returned. Many approaches have considered diversification [3, 4] to return interesting results or user-feedback [22]. In order to help the user in formulating the correct query feedback might be provided in the process [9].

The exploration of graphs can also be done with focused analyses, in which the user is proposed relevant portions of the graph. Clustering techniques can be influenced by example nodes provided by the user [12, 20, 21]. The same can be done with outlier detection, letting the user specify what is normal and what is exceptional [5, 27].

5 Collaborations within research groups

I successfully opened a collaboration with HCI group of prof. Baudisch on mechanism mining, and advertised a thesis in that topic. Moreover, the graph exploration tutorial will be presented with Anja Jentzsch from the Information Systems group. Externally, I am collaborating with the University of Trento, the ISI foundation in Turin and we are in contact with prof. Aris Gionis from Aalto University (Finland).

6 Future Work

We are currently working in the graph exploration area to produce significant results quickly. One active direction is the study of faceted search for graphs in which a query is decomposed in topics (facets) that are interesting for the user. Moreover, we intend to explore further the discovery of notable characteristics with in a more formal and fundamental way.

References

[1] R. L. Breiger, S. A. Boorman, and P. Arabie. "An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling". In: *Journal of mathematical psychology* 12.3 (1975), pages 328–383. ISSN: 0022-2496.

[2] S. Chakrabarti. "Dynamic personalized pagerank in entity-relation graphs". In: *WWW*. 2007, pages 571–580. ISBN: 978-1-59593-654-7.

[3] W. Fan, X. Wang, and Y. Wu. "Diversified top-k graph pattern matching". In: *Proceedings of the VLDB Endowment* 6.13 (2013), pages 1510–1521. ISSN: 2150-8097.

[4] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han. "Top-k interesting subgraph discovery in information networks". In: *IEEE 30th International Conference on Data Engineering (ICDE)*. IEEE. 2014, pages 820–831.

[5] M. Gupta, A. Mallya, S. Roy, J. H. Cho, and J. Han. "Local learning for mining outlier subgraphs from network datasets". In: *Proceedings of the 2014 SIAM International Conference on Data Mining*. 2014, pages 73–81.

[6] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. "Overview of Data Exploration Techniques". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pages 277–281. ISBN: 978-1-4503-2758-9.

[7] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri. "Querying knowledge graphs by example entity tuples". In: *TKDE* 27.10 (2015), pages 2797–2811. ISSN: 1041-4347.

[8] G. Jeh and J. Widom. "SimRank: a measure of structural-context similarity". In: *KDD*. 2002, pages 538–543. ISBN: 1-58113-567-X.

[9] J. Jin, S. Khemmarat, L. Gao, and J. Luo. "Querying web-scale information networks through bounding matching scores". In: *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2015, pages 527–537. ISBN: 978-1-4503-3469-3.

[10] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. "Nema: Fast graph search with label similarity". In: *Proceedings of the VLDB Endowment*. Volume 6. 3. VLDB Endowment. 2013, pages 181–192.

[11] J. M. Kleinberg. "Authoritative sources in a hyperlinked environment". In: *JACM* 46.5 (1999), pages 604–632. ISSN: 0004-5411.

[12] I. M. Kloumann and J. M. Kleinberg. "Community membership identification from small seed sets". In: *Proceedings of KDD*. 2014, pages 1366–1375. ISBN: 978-1-4503-2956-9.

[13] S. Lee, S. Lee, and B.-H. Park. "PathMining: A Path-Based User Profiling Algorithm for Heterogeneous Graph-Based Recommender Systems." In: *FLAIRS Conference*. 2015, pages 519–523.

[14] S. Lee, S. Park, M. Kahng, and S.-g. Lee. "Pathrank: a novel node ranking measure on a heterogeneous graph for recommender systems". In: *CIKM*. 2012, pages 1637–1641.

[15] F. Lorrain and H. C. White. "Structural equivalence of individuals in social networks". In: *The Journal of mathematical sociology* 1.1 (1971), pages 49–80.

[16] C. Meng, R. Cheng, S. Maniu, P. Senellart, and W. Zhang. "Discovering meta-paths in large heterogeneous information networks". In: *WWW*. 2015, pages 754–764. ISBN: 978-1-4503-3469-3.

[17] D. Mottin, F. Bonchi, and F. Gullo. "Graph Query Reformulation with Diversity". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pages 825–834. ISBN: 978-1-4503-3664-2.

[18] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. "Exemplar queries: Give me an example of what you need". In: *PVLDB* 7.5 (2014), pages 365–376.

[19] D. Mottin, A. Marascu, S. B. Roy, G. Das, T. Palpanas, and Y. Velegrakis. "A holistic and principled approach for the empty-answer problem". In: *VLDB J.* (2016), pages 1–26. ISSN: 0949-877X.

[20] B. Perozzi, L. Akoglu, P. Iglesias Sánchez, and E. Müller. "Focused clustering and outlier detection in large attributed graphs". In: *KDD*. 2014, pages 1346–1355. ISBN: 978-1-4503-2956-9.

[21] P. I. Sanchez, E. Müller, U. L. Korn, K. Böhm, A. Kappes, T. Hartmann, and D. Wagner. "Efficient Algorithms for a Robust Modularity-Driven Clustering of Attributed Graphs". In: *Proceedings of the 2015 SIAM International Conference on Data Mining*. 2015, pages 100–108. DOI: 10.1137/1.9781611974010.12.

[22] Y. Su, S. Yang, H. Sun, M. Srivatsa, S. Kase, M. Vanni, and X. Yan. "Exploiting Relevance Feedback in Knowledge Graph Search". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pages 1135–1144. ISBN: 978-1-4503-3664-2.

[23] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks". In: *Proceedings of the VLDB Endowment* 4.11 (2011), pages 992–1003. ISSN: 0957-4174.

[24] E. Vasilyeva, M. Thiele, C. Bornhövd, and W. Lehner. "Answering "Why Empty?" and "Why So Many?" queries in graph databases". In: *Journal of Computer and System Sciences* 82.1 (2016), pages 3–22. ISSN: 0022-0000.

[25] S. Yang, Y. Xie, Y. Wu, T. Wu, H. Sun, J. Wu, and X. Yan. "SLQ: a user-friendly graph querying system". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pages 893–896.

[26] Y. Yuan, G. Wang, L. Chen, and H. Wang. "Efficient subgraph similarity search on large probabilistic graph databases". In: *Proceedings of the VLDB Endowment* 5.9 (2012), pages 800–811. ISSN: 2150-8097.

[27] H. Zhuang, J. Zhang, G. Brova, J. Tang, H. Cam, X. Yan, and J. Han. "Mining query-based subnetwork outliers in heterogeneous information networks". In: *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE. 2014, pages 1127–1132.

Optimizing Noisy Functions: Resampling vs. Recombination

Francesco Quinzan

Algorithm Engineering
Hasso-Plattner-Institut
francesco.quinzan@hpi.uni-potsdam.de

Noise is pervasive in real-world optimization, but there is still little understanding of the interplay between the operators of randomized search heuristics and explicit noise-handling techniques, such as statistical resampling. We build and report on several statistical models and theoretical results that help to clarify this reciprocal relationship for a collection of randomized search heuristics on noisy functions.

We consider the optimization of pseudo-Boolean functions under additive posterior Gaussian noise and explore the trade-off between noise reduction and the computational cost of resampling. We first perform experiments to find the optimal parameters at a given noise intensity for a mutation-only evolutionary algorithm, a genetic algorithm employing recombination, an estimation of distribution algorithm (EDA), and an ant colony optimization algorithm. We then observe how the optimal parameter depends on the noise intensity for the different algorithms. Finally, we locate the point where statistical resampling costs more than it is worth in terms of run time. We find that the EA requires the highest number of resamples to obtain the best speed-up, whereas crossover reduces both the run time and the number of resamples required. Most surprisingly, we find that EDA-like algorithms require no resampling, and can handle noise implicitly.

1 Overview

We consider four algorithms, namely $(\mu+1)$-EA, $(\mu+1)$-GA, cGA, and λ-MMASib. $(\mu+1)$-EA and $(\mu+1)$-GA are search heuristics that mimic the process of natural selection (cf. Algorithm 1 and Algorithm 2).

Typically, they require as input a *population* of strings of fixed length n. After an *offspring* is generated, a mutation factor is introduced, to ensure full objective space exploration. The fitness is then computed, and the less desirable result is discarded. The $(\mu+1)$-EA and the $(\mu+1)$-GA differ in how the offspring is generated. In the first case, the offspring is selected u.a.r. from the input population, while in the latter case a crossover operation is performed on two u.a.r. chosen strings. We use uniform crossover, which consists of assembling a new element by choosing coefficients of one parent or the other with probability $p = 0.5$.

The cGA is a search heuristic similar to $(\mu+1)$-EA and $(\mu+1)$-GA. As shown in Algorithm 3, this process consists of sampling two *individuals* with given probability distribution, and swapping them according to the fitness evaluation. At each step, the distribution by which individuals are chosen is updated according to the fitness gain, and proportionally to a parameter K. The offspring generation procedure of the Compact Genetic Algorithm is equivalent to a concrete population with the same allele frequencies engaging in *gene pool recombination* introduced by Mühlenbein and

Algorithm 1: $(\mu + 1)$-EA

$t \leftarrow 0$
Choose a population $P_0 \subseteq \{0,1\}^n$ s.t. $|P_0| = \mu$ u.a.r
while *convergence criterion not met* **do**
    Select a parent $x \in P_t$ u.a.r
    Generate offspring y by flipping each bit of x u.a.r
    Choose $z \in P_t \cup \{y\}$ s.t. $f(z) = \max_{x \in P_t} f(x)$
    Define population $P_{t+1} \leftarrow (P_t \cup \{y\}) \setminus \{z\}$
    $t \leftarrow t + 1$
end

Paaß [16]. In gene pool recombination, all members of the population participate as parents in uniform recombination. The correspondence between EDAs and models of sexually recombining populations has already been noted (cf. Mühlenbein and Paaß [15]). The λ-MMASib is an ant colony optimization method (cf. Algorithm 4). Algorithms of this kind can be described in terms of λ ants exploring given *paths*, which correspond to pseudo-Boolean arrays of length n. The probability distribution by which ants choose one path over another is called *pheromone* vector, and it is updated at each step according to the fitness evaluation, and proportionally to a parameter ρ. Both cGA and λ-MMASib are an estimation of distribution algorithms (EDAs).

Each algorithm depends on one or two parameters. The goal of parameter tuning is to reduce the expected number of fitness evaluations until the optimum search point is found. We refer to the optimal configuration by which the expected number of fitness evaluations is minimal as the *sweet spot*. Figure 1a shows the dependence of the optimization time of the $(\mu + 1)$-EA and $(\mu + 1)$-GA on the parameter μ. Note that both axes use a logarithmic scale. We see that optimization is slow for very small population sizes μ, quickly improves to best performance, and then slowly worsens again. This is in contrast to the well-known fact that, in the absence of noise, a choice of $\mu = 1$ is optimal for the $(\mu + 1)$-EA. In Figure 1b we can see a similar trend for the cGA. In this case, however, we observe a slightly different behavior. In fact, higher K gives better worst-case and worse average-case. This observation has been already framed theoretically (cf. Droste [7]). For the cGA without boundaries on the distribution adjustment, there is a non-zero probability that the algorithm converges in infinite time ($p(+\infty) > 0$). In our case, we put boundaries on the distribution adjustment (cf. Algorithm 3), and we expect $p(+\infty) = 0$. However, during some runs the algorithm still may take a very long time to reach the optimum. The case of λ-MMASib is more involved, since we have to optimize two parameters in parallel. As we can see in Figure 2a, an evaporation factor slightly below 0.05 with a number of ants of around 5 is optimal.

In all cases there exists a sweet spot for the choice of parameters, at which the algorithm performs best. The difference between the sweet spot and the optimal param-

Algorithm 2: $(\mu + 1)$-GA

$t \leftarrow 0$
Choose population $P_0 \subseteq \{0,1\}^n$ s.t. $|P_0| = \mu$ u.a.r
while *convergence criterion not met* **do**
    Select parents $x_1, x_2 \in P_t$ u.a.r
    Generate offspring y by recombining x_1 and x_2 u.a.r
    Choose $z \in P_t \cup \{y\}$ s.t. $f(z) = \max_{x \in P_t} f(x)$
    Define population $P_{t+1} \leftarrow (P_t \cup \{y\}) \setminus \{z\}$
    $t \leftarrow t + 1$
end

Algorithm 3: cGA

$t \leftarrow 0$
$p_{1,t} \leftarrow p_{2,t} \leftarrow \cdots \leftarrow p_{n,t} \leftarrow 0.5$
while *convergence criterion not met* **do**
    for $i = 1 \dots n$ **do**
        $x_i \leftarrow 1$ w/ prob. $p_{i,t}$, $x_i \leftarrow 0$ w/ prob. $1 - p_{i,t}$
        $y_i \leftarrow 1$ w/ prob. $p_{i,t}$, $y_i \leftarrow 0$ w/ prob. $1 - p_{i,t}$
    end
    if $f(x_1, \dots, x_n) < f(y_1, \dots, y_n)$ **then**
        Swap $(x_1, \dots, x_n)$ with $(y_1, \dots, y_n)$
    end
    for $i = 1 \dots n$ **do**
        if $x_i > y_i$ **then**
            $p_{i,t+1} \leftarrow \min\left(\max\left(p_{i,t} + \frac{1}{K}, \frac{1}{n}\right), 1 - \frac{1}{n}\right)$
        else if $x_i < y_i$ **then**
            $p_{i,t+1} \leftarrow \min\left(\max\left(p_{i,t} - \frac{1}{K}, \frac{1}{n}\right), 1 - \frac{1}{n}\right)$
        else
            $p_{i,t+1} \leftarrow p_{i,t}$
        end
    end
    $t \leftarrow t + 1$
end

Algorithm 4: λ-MMASib

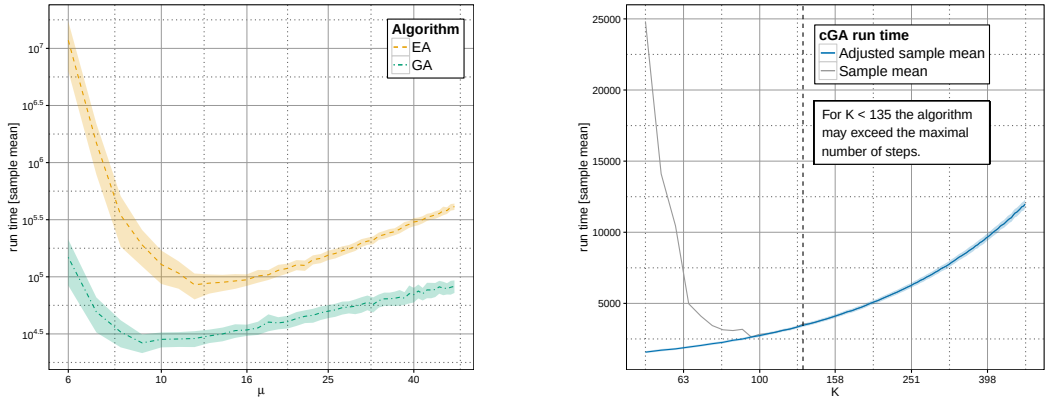
$t \leftarrow 0$
$p_{1,t} \leftarrow p_{2,t} \leftarrow \cdots \leftarrow p_{\lambda,t} \leftarrow 0.5$
while *convergence criterion not met* **do**
  for $i = 1 \dots \lambda$ **do**
    Generate x_i w/ prob. $p_t = (p_{1,t}, \dots, p_{\lambda,t})$
  end
  Choose $z \in \{x_1, \dots, x_\lambda\}$ s.t. $f(z) = \min_i\{f(x_i)\}$
  for $i = 1 \dots n$ **do**
    if $z_i = 1$ **then**
      $p_{i,t+1} \leftarrow \min\left(p_{t,i}(1-\rho) + \rho, 1 - \frac{1}{n}\right)$
    else
      $p_{i,t+1} \leftarrow \max\left(p_{t,i}(1-\rho), \frac{1}{n}\right)$
    end
  end
  $t \leftarrow t + 1$
end

eter with absence of noise depends on the algorithm's stability to fitness evaluation errors.

For each algorithm we approximate the run time function in dependent on the standard deviation and optimal parameter choice. The run time plot in Figure 2b shows that the $(\mu + 1)$-EA becomes quickly inefficient, when increasing posterior noise standard deviation. In the case of the $(\mu + 1)$-GA, results seem to be slightly better. However, we still can see that this algorithm significantly worsens with increasing posterior noise standard deviation. In the case of the λ-MMASib we see further improvement. The results in Figure 2b show that it reacts much better to increasing posterior noise standard deviation. Still, we observe a polynomial trend, with degree clearly greater than one (cf. Figure 2b). We see a similar trend for the cGA.

We show that there exists a sweet spot for the resampling operator, as in the case of the parameter tuning experiments.

We compare the run time trend of the $(\mu + 1)$-EA, with the run time trend of $(\mu + 1)$-GA, both given when the resampling operator is used and when it is not. For both algorithms we observe improvement. However, for posterior noise standard deviation $\sigma \leq 10$, it seems that the $(\mu + 1)$-EA with resampling still performs worse than the $(\mu + 1)$-GA without resampling. Additional experiments show that the resampling sweet-spot for the cGA and λ-MMASib is $r_\sigma = 1$. In Figure 2b, we compare the $(\mu + 1)$-EA and $(\mu + 1)$-GA with resampling, with the cGA, and λ-MMASib without resampling. We see that the cGA and λ-MMASib perform better then the $(\mu + 1)$-EA and $(\mu + 1)$-GA with resampling. This confirms the fact that the benefit of resampling is limited, and noticeable only with algorithms that perform particularly



(a) Number of fitness evaluations (run time) for a given parameter choice (μ) for the $(\mu+1)$-EA and $(\mu+1)$-GA. The shading is proportional to the sample standard deviation.

(b) Number of fitness evaluations (run time) for the parameter K for the cGA. The shading is proportional to the sample standard deviation.

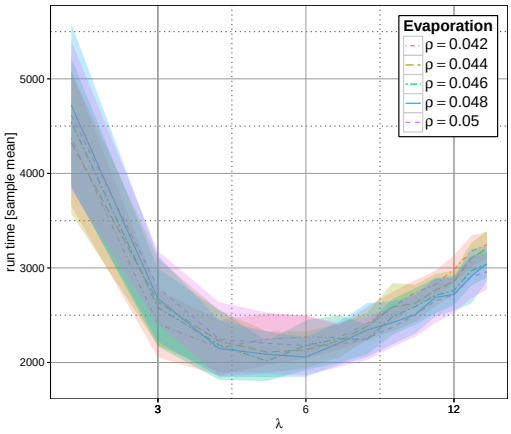
Figure 1: Sweet spot experimental trend.

Table 1: Run time for the four algorithms, with additive posterior gaussian noise standard deviation σ

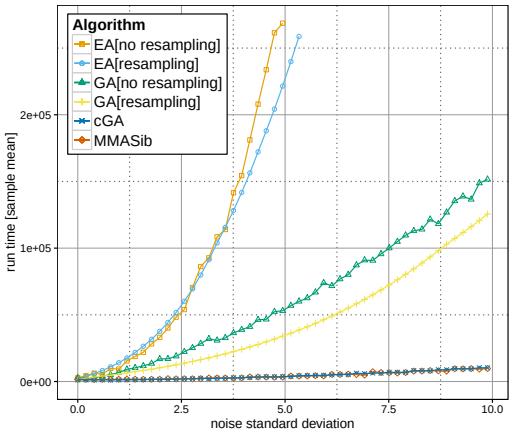
algorithm	**run time no resampling**	**resampling sweet spot**	**run time w/ resampling**
$(\mu+1)$-EA	$\mathcal{O}(e^{c\sqrt{\sigma}})$	$\mathcal{O}(\sigma^2)$	$\mathcal{O}(\sigma^2)$
$(\mu+1)$-GA	$\mathcal{O}(e^{c\sqrt{\sigma}})$	$\mathcal{O}(\sigma^2)$	$\mathcal{O}(\sigma^2)$
cGA	$\mathcal{O}(\sigma^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\sigma^2)$
λ-MMASib	$\mathcal{O}(\sigma^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\sigma^2)$

bad under noisy environments. By means of model regression, we estimate the run time with and without re-sampling for the four algorithms, as displayed in Table 1.

These results are not in contrast with the experiments displayed in Figure 2b. In fact, the resampling operator gives a faster run time for $(\mu+1)$-GA and $(\mu+1)$-EA, but its effect is noticeable only for very large posterior noise standard deviation σ. In the case of $\sigma \leq 10$, the benefit of recombination overwhelms the effect of the resampling operator. Nevertheless, for very large σ, we expect the $(\mu+1)$-EA with resampling to give better performance then the $(\mu+1)$-GA without resampling.



(a) Number of fitness evaluations (run time) for a given pair of parameters (λ, ρ) for the λ-MMASib. The shading is proportional to the sample standard deviation.



(b) Number of fitness evaluations (run time) for the$(\mu+1)$-EA and $(\mu+1)$-GA with optimal resampling, and the cGA, $(\mu+1)$-EA, $(\mu+1)$-GA and λ-MMASib with no resampling, for a given posterior noise standard deviation.

Figure 2: Sweet spot and run time experimental trend.

2 Approach

All tests are carried out by performing a global optimum search of the following function:

$$\text{OneMax}\colon (x_1, \dots, x_n) \longmapsto \sum_{i=1}^{n} x_i$$

with $x_i \in \{0, 1\}$ for all $i = 1, \dots n$, and with objective space consisting of all pseudo-Boolean strings of fixed size n. We add posterior Gaussian noise in order to simulate an environment where errors of controlled standard deviation are produced. In some cases we simulate a resampling operator: we compute the fitness function r times, and take the average. Thus the test function in its generic form is:

$$\frac{1}{r}\sum_{j=1}^{r}\left(\text{OneMax} + \mathcal{N}(0, \sigma^2)\right) = \text{OneMax} + \frac{1}{r}\sum_{j=1}^{r}\mathcal{N}(0, \sigma^2)$$

for a given $r > 0$. The OneMax function has the advantage of being symmetric and well-understood. The goal of the hereby presented experiments is to statistically infer the asymptotic trend of some algorithms in view of a broader theoretical setting.

For each set of experiments we look at the sample mean, sample standard deviation, and infer the trend toward asymptotic behavior via model regression. All samples considered are of size $N \geq 10^2$; the exact size and relevant information is given in the description of each experiment. Statistical models with different properties are considered:

- polynomial models: $X \sim \alpha x^k + \beta$;

- rational models: $X \sim \frac{1}{\alpha x^k + \beta}$;
- square-root models: $X \sim \alpha\sqrt{x} + \beta$;
- square-root exponential models: $X \sim \alpha e^{\beta\sqrt{x}}$;
- any linear combination of the above.

In all cases, tests on the predictions made by the fitting models are performed. For a given experiment described by pairs $\{(x_i, y_i)\}_{i\in I}$, denote with $\bar{y}$ the sample mean, and let $\{f_i\}_{i\in I}$ be the predictions of a given model. We assume that the model is valid if $R^2 > 0.95$, with R^2 the coefficient of determination. This choice is intuitively motivated by the fact that R^2 is the "percent of variance explained" by the model.

We perform a Student's t-Test on each model, to determine whether it outperforms "random noise" as a predictor. We look at the corresponding p-value, and consider the model valid only for p-value < 0.05. We accept variables such that the probabilities $p_{|t|}$ of obtaining a corresponding value outside the confidence interval are $p_{|t|} < 10^{-5}$. Thus all variables have a very high level of significance.

All tests are carried out on MacBook Pro (13" Retina, Beginning 2015), with operating system Mac OS X Version 10.11.1, processor 2.7 GHz dual-core Intel Core i5 (Turbo Boost up to 3.1 GHz) with 3 GB shared L3 cache, and memory 8 GB of 1866 MHz LPDDR3. All algorithms are implemented in C++ on Xcode Version 7.3.1 (7D1014), and implemented as OSX command line executables. The fitting is performed using least-square methods implemented by the 'lm' command of R 3.2.2 GUI 1.66 Mavericks build (6996).

3 Related Work

The idea of statistical resampling to address noise in the context of genetic algorithms has been studied as far back as 1988. In a paper by Fitzpatrick and Grefenstette [8], it was argued that the implicit parallelism of a GA is a sufficient mechanism for handling noise. They found that the amount of explicit resampling can be reduced in a GA by increasing the population size. In the context of Evolution Strategies (ES) Arnold and Beyer [3] also found that increased population sizes are preferable to resampling as long as mutation strength is optimally adapted. Goldberg et al. [11] studied the influence of GA population size in the presence of noisy functions, but also with more general sources of noise (such as noisy operators). Arnold and Beyer [4] also noticed that the same algorithm may react in different ways to different types of posterior noise distributions.

The issue of resampling as a noise-handling technique has been approached from many different perspectives over the last few decades. Aizawa and Wah [1] proposed a detailed adaptive strategy for modelling the underlying noise, in order to determine the appropriate number of samples to be drawn from each individual. Stagge [19] recognized that noise can be reduced by repeated sampling, even at the cost of a higher number of function evaluations. However, he argues that computational

effort can be saved by focusing only on resampling for the *best* individuals in the population. Many other detailed sampling frameworks have been proposed, such as ones based on *selection races* from the machine learning community [13, 18].

Branke and Schmidt [5, 6] also recognised that resampling strategies produce a trade-off between noise reduction and computational effort. They consider a number of different sampling procedures that attempt to characterize the error probability during the selection step. They also raise the interesting point that sometimes noise can be *beneficial* to stochastic search algorithms (e.g., for promoting objective space exploration), and therefore attempting to eliminate noise completely may not always be the best strategy. Akimoto et al. [2] explicitly study the run time effect of resampling on various noise models to derive the extra cost incurred by performing enough resampling to ensure the underlying optimization algorithm "sees" a noiseless function. For Gaussian noise, they show the existence of a resampling scheme such that any optimization algorithm that requires $r(\delta)$ function evaluations to optimize a noise-free function f with probability $1-\delta$ requires $\mathcal{O}(r(\delta)\max\{1, \sigma^2 \log(r\delta)/\delta\})$ evaluations to optimize $f + \mathcal{N}(0, \sigma^2)$ with probability $(1-\delta)^2$ under their resampling scheme.

Recently, Friedrich et al. [10] proved that an Estimation of Distribution Algorithm (EDA) called the Compact Genetic Algorithm (cGA) can *scale gracefully* with noise: its runtime on a noisy OneMax function is bounded by a polynomial in the variance, regardless of noise intensity. The cGA does not explicitly keep a population in memory, but only an array of so-called *allele frequencies* that represent the product distribution of alleles in an implicit population (cf. Goldberg et al. [12]). Offspring are then generated by drawing from this product distribution, and this has the same effect as *gene pool recombination*: all members of the population participate in recombination (rather than, e.g., two) to produce an offspring. On the other hand, they also proved that a mutation-only evolutionary algorithm exhibits superpolynomial runtime due to its reliance on hill-climbing a gradient that becomes obscured in the presence of heavy noise.

The array of allele frequencies employed by the cGA is similar to the pheromone values stored by Ant Colony Optimization (ACO) algorithms. In the latter, the update rule is distinct, but we should still expect similar protection against noise, which has also been recently noted [9]. None of these cases employ resampling, but instead rely on the implicit mechanisms of the GA/EDA/ACO to somehow "filter" the noise. Prügel-Bennett, Rowe and Shapiro [17] proved that a generational evolutionary algorithm using uniform crossover finds the optimum in $\mathcal{O}(\sigma^2 \log^2(\sigma^2))$ fitness evaluations, on OneMax with additive Gaussian noise of standard deviation σ.

4 Future Work

In the context of Las Vegas algorithms, theoretical results have been presented to address the problem of finding the optimal restart point. Particularly interesting is a technique that finds this point by minimizing a function solely dependent on the

Algorithm 5: Luby universal strategy for an algorithm $\mathcal{A}$

```
k ← 1
while convergence criterion not met do
    for i in 0 : k do
        for j in 0 : i do
            run A for 2^j steps
        end
    end
    k ← k + 1
end
```

cumulative density associated to the probability of converging at a given time step (cf. Luby et al. [14]).

More specifically, let $\mathcal{A}$ be any Las Vegas algorithm, $p(t)$ the probability of convergence at step t, and $q(t) = \sum_{t' \leq t} p(t)$ the probability of converging before step t—i.e. the cumulative distribution function. The optimal restart strategy for $\mathcal{A}$ is the repeating sequence $\mathcal{S} = (t_*, \dots, t_*, \dots)$ with t_* defined as

$$t_* := \inf_t \left\{ \frac{1}{q(t)} \left(t - \sum_{t' < t} q(t') \right) \right\} . \tag{1}$$

Thus, t_* is the first point in time that minimizes the probability of not converging over the probability of converging at previous steps. We refer to this result as Luby theorem. Luby et al. prove $\mathcal{S}$ to be optimal, both for t_* a finite number, and $t_* = +\infty$—in the latter case the optimal strategy being not restarting the process at all. In most real world settings, however, $p(t)$ is not known a priori, and a numerical approximation of it is not feasible. Because of this, Luby et al. present a universal method to simulate $\mathcal{S}$ when $p(t)$ is unknown. This strategy is indicated by

$$\mathcal{S}^{\mathbf{univ}} = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \dots)$$

Pseudo-code for an algorithm $\mathcal{A}$ running as indicated by $\mathcal{S}^{\mathbf{univ}}$ is given in Algorithm 5. We adapt the results presented above to the case of evolutionary heuristics running with time budget constraints, or with solution quality constraints. We focus on finding an explicit relationship between the optimal restart point given in Equation 1, the total time budget, and the impact on overall performance.

References

[1] A. Aizawa and B. W. Wah. "Scheduling of Genetic Algorithms in a Noisy Environment". In: *Evolutionary Computation* 2.2 (1994), pages 97–122.

[2] Y. Akimoto, S. Astete-Morales, and O. Teytaud. "Analysis of Runtime of Optimization Algorithms for Noisy Functions Over Discrete Codomains". In: *Theoretical Computer Science* 605 (2015), pages 42–50.

[3] D. Arnold and H. G. Beyer. "Efficiency and Mutation Strength Adaptation of the (μ, μ_I, λ)-ES in a Noisy Environment". In: *Proceedings of PPSN*. 2000, pages 39–48.

[4] V. D. Arnold and G. H. Beyer. "A General Noise Model and Its Effects on Evolution Strategy Performance". In: *Evolutionary Computation, IEEE Transactions on* 10.4 (2006), pages 380–391.

[5] J. Branke and C. Schmidt. "Selection in the Presence of Noise". In: *Proceedings of GECCO*. 2003, pages 766–777.

[6] J. Branke and C. Schmidt. "Sequential Sampling in Noisy Environments". In: *Proceedings of PPSN*. 2004, pages 202–211.

[7] S. Droste. "A Rigorous Analysis for the Compact Genetic Algorithm for Linear Functions". In: *Natural Computing* 5.4 (2006), pages 257–283.

[8] M. J. Fitzpatrick and J. J. Grefenstette. "Genetic Algorithms in Noisy Environments". In: *Machine Learning* 3.2 (1988), pages 101–120.

[9] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. "Robustness of Ant Colony Optimization to Noise". In: *Proceedings of GECCO*. 2015, pages 17–24.

[10] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. "The Benefit of Recombination in Noisy Evolutionary Search". In: *Proceedings of ISAAC*. 2015, pages 140–150.

[11] D. E. Goldberg, K. Deb, and J. H. Clark. "Genetic Algorithms, Noise, and the Sizing of Populations". In: *Complex Systems* 6 (1992), pages 333–362.

[12] G. R. Harik, F. G. Lobo, and D. E. Goldberg. "The Compact Genetic Algorithm". In: *Proceedings of IEEE*. 1998, pages 523–528.

[13] V. Heidrich-Meisner and C. Igel. "Hoeffding and Bernstein Races for Selecting Policies in Evolutionary Direct Policy Search". In: *Proceedings of ICML*. 2009, pages 401–408.

[14] M. Luby, A. Sinclair, and D. Zuckerman. "Optimal Speedup of Las Vegas Algorithms". In: *Information Processing Letters* 47 (1993), pages 173–180.

[15] H. Mühlenbein and G. Paaß. "From Recombination of Genes to the Estimation of Distributions I. Binary Parameters". In: *Proceedings of PPSN*. Springer-Verlag, 1996, pages 178–187. ISBN: 3-540-61723-X.

[16] H. Mühlenbein and H. Voigt. "Gene Pool Recombination in Genetic Algorithms". In: *Meta-Heuristics*. Springer US, 1996, pages 53–62.

[17] A. Prügel-Bennett, J. Rowe, and J. Shapiro. "Run-Time Analysis of Population-Based Evolutionary Algorithm in Noisy Environments". In: *Proceedings of Foundations of Genetic Algorithms (FOGA)*. ACM, 2015, pages 69–75. ISBN: 978-1-4503-3434-1.

[18] P. Rolet and O. Teytaud. "Bandit-Based Estimation of Distribution Algorithms for Noisy Optimization: Rigorous Runtime Analysis". In: *Proceedings of LION*. 2010, pages 97–110.

[19] P. Stagge. "Averaging Efficiently in the Presence of Noise". In: *Proceedings of PPSN*. 1998, pages 188–200.

Active Expressions as a Basic Building Block for Reactive Programming Concepts

Stefan Ramson

Software Architecture Group
Hasso-Plattner-Institut
Stefan.Ramson@hpi.uni-potsdam.de

Reactive programming simplifies the implementation of complex, interconnected behavior by completely automating the propagation of changes. Researchers proposed a huge variety of approaches to execute on this initial idea. Most approaches present specialized solutions to maximize expressiveness and meet unique use cases. But usually, they are not intended for reuse. Thus, when implementing novel reactive concepts, developers often have to start from scratch.

We identify a class of reactive programming concepts, *state-based reactive programming*, that incorporates a common underlying change notification mechanism. Furthermore, we propose a reusable primitive, *active expressions*, to facilitate the implementation of instances of this class of reactive concepts. Finally, we present an implementation of active expressions along with multiple strategies for state change monitoring.

1 Reactivity in Software

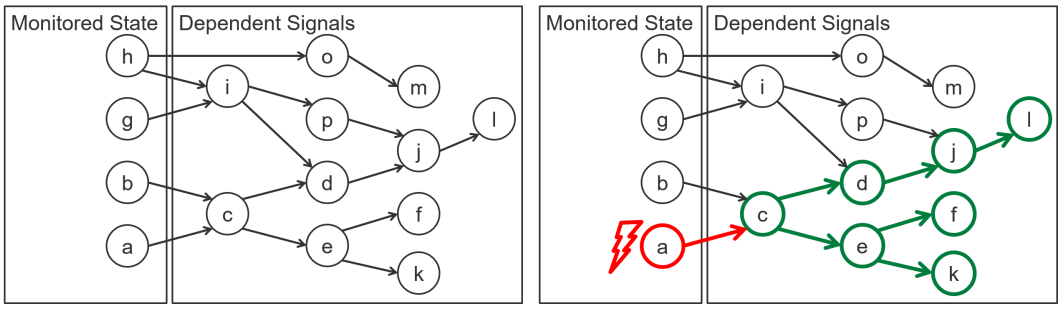
Many contemporary software systems are reactive. A Web application needs to react to messages from the server. When editing a cell in a spreadsheet, other cells might change as well if they depend on edited cell's value. In signal processing, embedded software might react to an external sensor that signals changing temperature and has to monitor or compute on this data. In contrast to simple batch applications that simply compute outputs from inputs according to an algorithm, reactive systems have to react to changes in the input or state and act accordingly. Once a certain event is detected, consequential state changes have to be propagated consistently throughout the entire system.

The traditional, object-oriented way to handle reactivity is by employing the observer design pattern [4]. According to the observer pattern, multiple observers can subscribe to a subject. Then, whenever the subject's state changes, it notifies its subscribers about the change. While the observer pattern provides a useful functionality, it also implies some disadvantages. First, it is not syntactically integrated into the language. Second, it introduces additional complexity in your program. Third, it couples the subject with the notification mechanism, and cannot be employed without access to the subject module [11].

Due to the disadvantages of the observer pattern, researchers developed more dedicated mechanisms to support reactive behavior. These solutions are often integrated into the underlying language to provide a clean abstraction for reactivity [12].

Additionally, they automate the updating of dependent modules without the need for complex boilerplate code.

1.1 A Recurring Reactive Pattern



(a) Signal implementations maintain a dependency network of signals as basic variables.

(b) Changing a basic variable causes all signals in its convex hull to update.

Figure 1: Reactive patterns with Signals.

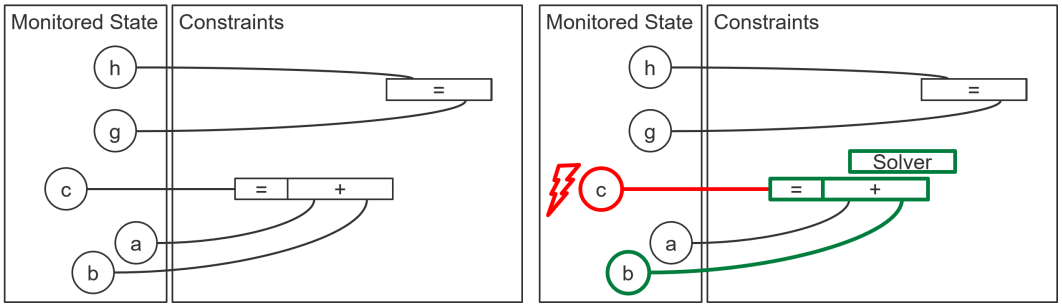
In the past few years, reactive programming has become of particular interest for researchers in the language design community due to its relevance for contemporary software. Thus, many different kinds of reactive concepts have been proposed or are still under active research. In the following, we present a subset of these approaches.

Signals Signals are time-varying values [1]. They can be declared like any other variable by providing an expression:

```
1 var a = 5;
2 var b = 6;
3 signal c = a + b;
4 // ...
5 console.log(c); // 11
6 a = 10;
7 console.log(c); // 16
```

However, instead of performing the assignment once, the declaration of a signal introduces a functional dependency. As a result, `c` is recomputed according to its production rule by the underlying runtime whenever `a` or `b` are changed. Thus, the relation `c = a + b` is always true.

To keep track of the functional dependencies in a program the signal runtime maintains an acyclic dependency graph as exemplified in Figure 1. Additionally, all normal variables referenced by a signal are continuously monitored. Whenever an assignment to a monitored variable is detected in the signal runtime, the change of this variable propagates through the dependency network, ultimately updating all dependent signals.

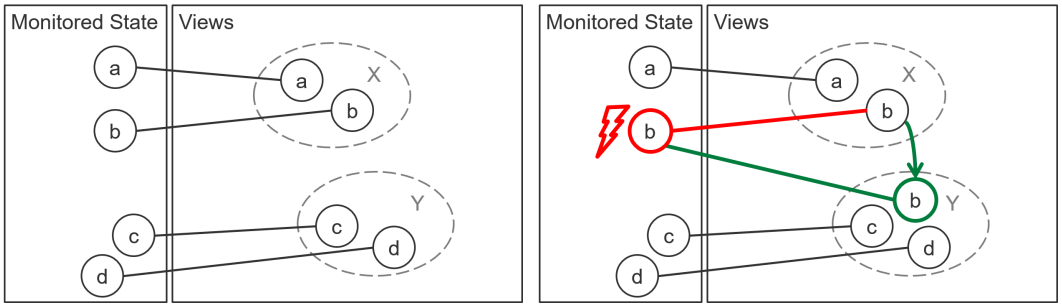


(a) Constraints maintain desired relations between variables.

(b) Imperative code might invalidate certain constraints by assigning new values to variables. This causes a constraint solver to adjust variables to keep the desired relations.

Figure 2: Reactive patterns with Constraints.

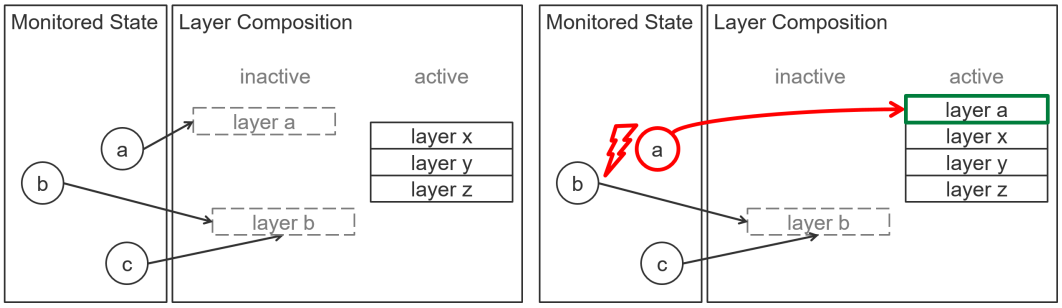
Constraints Constraint-imperative programming [3, 5] and object constraint programming [2] aim to integrate constraints into imperative languages. Constraints are relations between objects that should hold. When specifying a constraint, specialized constraint solvers are used to adapt variables in a way to fulfill the given condition. During further execution, imperative code might again invalidate the given condition as shown in Figure 2. Thus, the variables referenced in the constraint are monitored for changes by the underlying execution environment. When a change invalidates a constraint, the system again uses the constraint solvers to maintain a consistent system state according to the specified constraints.



(a) Views containing objects according to their properties

(b) An underlying system detects changes to those properties and adds objects to or removes them from certain views in reaction to these changes.

Figure 3: Reactive patterns with Object Queries.



(a) The layer composition stack determines in which order active class extensions are applied when calling a method. With implicit layer activation, the activation state of a layer depends on the program's state.

(b) Changing the monitored state might alter the layer composition.

Figure 4: Reactive patterns with implicit layers.

Reactive Object Queries Reactive Object Queries [7] apply reactivity to data structures, namely sets. Instead of manually constructing and maintaining a set of objects, one can use a query to automatically construct a set of instances of a certain class that fulfill a given condition. The resulting set acts similar to a view in conventional databases: it automatically updates whenever the program state changes. As a result, the set always maintains consistency with the underlying system state. To do so, the underlying system monitors all class instances for changes that could affect the given condition as shown in Figure 3. When relevant values change, the condition is reevaluated and the set is updated accordingly.

Implicit Layer Activation In context-oriented programming, one can define multiple class extensions in a single unit of modularity, called layer. During the execution of a program this layer can be activated to dynamically apply the class extensions, thus, adapting the behavior of a program. There are multiple means to activate a layer [6]. One of those activation means, implicit layer activation [8], has fairly reactive semantics. Using implicit layer activation, a layer is not activated or deactivated at a fixed time, but instead is active while a given condition holds. While typically implemented differently, an underlying system could monitor variables referenced by the given condition as depicted in Figure 4. When such a variable changes, the condition is reevaluated and the corresponding layer is adjusted accordingly.

⁂

Although each of the presented concepts works differently and tackles specific use cases, they share a similar overall mechanism: they monitor certain parts of the system's state and, whenever a change is detected, react to this very change in a concept-specific way. This recurring pattern is not limited to the presented concepts, but also includes for example two-way data bindings and incremental

lists [9]. Because all these concepts react to changes in the state of a program, we coin this identified class of reactive concepts *state-based reactive concepts*.

Despite this common underlying pattern, when developing a new state-based reactive concept, one can hardly reuse existing concepts. This is due to the fact that existing implementations are typically highly focused on the specific use case of the concept in question. As a consequence, state change recognizing mechanisms and concept-specific reactive behavior are tightly coupled together to maximize performance and convenience. As existing concepts are not intended for reuse, researchers are forced to reimplement rather complicated behavior such as the state change recognition every time they come up with a new reactive concept.

1.2 Designing the State-based Reactive Primitive

With the class of state-based reactive concepts, we identified a structure common to reactive programming concepts. However, the existing instances of the class are not intended for reuse, making the development of new state-based concepts difficult. This is due to the fact that a common foundation for state-based reactive mechanisms is still missing. In this work we aim to provide this common foundation which we call *active expressions*. To form a suitable foundation for state-based reactivity, active expressions have to fulfill multiple properties:

- The concept needs to represent the easiest instance of the class of state-based reactive concepts.
- The concept needs to be highly customizable.
- The concept should be useful on its own.

To fulfill the first point, we support only the identified mechanisms that are most essential and most common to state-based reactive concepts: monitoring certain parts of the system state and reacting to detected changes. For state change recognition, we choose to support expressions rather than variables as the unit of monitoring, because expressions provide more convenience and can express variable values as well. In general, one can monitor any expression as long as it follows certain restrictions: the expression has to be deterministic, free of visible side effects, and synchronous.

To make active expressions as customizable as possible, we assume no common behavior for change reactions. Instead, treat the change reaction as a variation point: users can subscribe to changes of a particular expression. Then, their specified callback is executed every time the result of the monitored expression changes. In addition to the new value of the expression, its old value as well as the reason for its change is provided to the given callback. Using these information, one can build rather specialized functionality on top of active expressions.

Active expressions provide an easy semantic and are highly configurable to act as the low-level foundation of state-based reactive concepts. From that perspective,

they are similar to event listeners that are the basis of many event-based reactive concepts. Therefore, we provide a simple interface based on well-known event listener mechanisms: `activeExpression(expression).onChange(callback);`.

2 Implementation

In industry, JavaScript has become the de-facto standard for Web programming with a rapidly growing amount of code that exists in the language. This fact, along with JavaScript's unique design and its execution environment in a Web browser, also make it of great interest to the research community. Many researcher were motivated to revise and adapt useful features of other languages to the domain of Web programming [10, 13].

An implementation of active expressions has to solve 2 tasks: recognizing changes in the system's state, and notifying dependent functionality about these changes. Our implementations reflect these two tasks with a two-part architecture.

The latter task is straight-forward to implement. Once a change in an expression result of an active expression is recognized, we first compare the current expression result with the last one, and, if they differ, invoke each callback associated with that active expression with the new result. This type of notification resembles a typical event emitter.

To solve the former task, we have to continuously monitor given expressions. In order to achieve practical support for active expressions in JavaScript, we must not rely on modifications of the underlying Virtual Machine (VM). Thus, we can run active expressions in modern browsers and use them in a variety of practical Web applications. However, because JavaScript does not provide rich meta programming facilities by default, our implementation has to provide a custom mechanism to hook into system state changes. We provide multiple recognition mechanisms that detect and handle states changes in either push- or pull-based manner. Each active expression uses exactly one of the mechanisms. These mechanisms are described in detail in the following subsections.

2.1 Ticking

The first implementation strategy requires the user to explicitly specify at which points during the execution the implementation should check for possible changes. The user marks such a point in execution by calling the exposed `check` function. At those user-defined points the ticking implementation notifies all active expressions that there have been potential changes, and that now is a good time to check on the new results. To do so, the ticking implementation maintains a global set of currently enabled active expressions. Any active expression created by using this implementation's `activeExpression` function is added to that global set. Optionally, you can provide an iterable over active expressions as a parameter to `check`. If you do, only those active expressions are informed about a potential change in their results.

2.2 Interpretation

One possibility to track changes in the system's state in JavaScript is through the use of property accessors. Property accessors allow to intercept get and set operations on member variables. To install appropriate property accessors, we first have to identify all variables that contribute to the result of an active expression. To do so, we perform an abstract interpretation for each active expression using Neil Fraser's JavaScript Interpreter,[1] which is also used for Google Blockly.[2] The interpreter is customized using means of context-oriented programming to intercept the access to properties. Each property accessed during interpretation is wrapped in a transparent property accessor. If the property is already wrapped, we add the currently interpreted active expression to a set of active expressions depending on this property instead.

Once the interpretation is finished, whenever a new value is assigned to a wrapped property, dependent active expressions are notified that their expression result might have changed. Assignments of complex values might necessitate reinterpretation.

The used interpreter relies on explicit access to the local scope of the expression. However, JavaScript does not support computational access to local scope by default. To solve this issue, we provide a babel[3] plugin that expands undeclared occurrences of the identifier `locals` to an object with all locally accessible references. The following JavaScript source code exemplifies this process:

```
var alice;
var bob;

activeExpression(expression, locals);
// expands to:

activeExpression(expression, {
  alice,
  bob
});
```

As the majority of JavaScript projects is transpiled via babel as part of their build process by the time of writing, developers can include this plugin to simplify the process of providing the necessary references.

2.3 Rewriting

The third implementation strategy captures changes to the system's state by injecting explicit hooks into the code itself using a source code transformation. In contrast to the interpretation strategy which makes use of limited built-in language features to detect changes to the system state, the rewriting strategy makes use of the vast build tool environments for JavaScript that emerged over the last couple of years. By the time of writing most new JavaScript projects include an additional transpi-

[1] https://github.com/NeilFraser/JS-Interpreter (last accessed 2016-09-28).
[2] https://developers.google.com/blockly/ (last accessed 2016-09-28).
[3] https://babeljs.io/ (last accessed 2016-09-29).

lation step to ensure compatibility to legacy environments while being able to use newest standards and even experimental language features. Because of its current prominence, the babel transpiler seems like the most viable option to implement our source transformation plugin.

The state change recognition mechanism is threefold. We first inject hooks into the source code at transpilation time. Then, when creating a new active expression we identify on what part of the state this active expression depends on. Finally, we notify active expressions once we recognize a change in the corresponding system state.

Only a limited number of language concepts can actually alter program state relevant to active expressions, among these concepts are assignments to object members or variables. So, in order to capture changes to the program state, we have to make these concepts computationally accessible through user code. Therefore, whenever we encounter a state-modifying or -accessing node during AST traversal, we replace the nodes with appropriate function calls. In addition to providing the needed hooks into program execution, these functions resemble the original semantics of the programming concept, making the rewritten program unaware of the source transformation. The required functions are automatically imported in a non-conflicting manner.

The presented transformation step is done statically during compile time. One downside of this approach is that we cannot determine upfront, which portion of the source code will be relevant for active expressions. As a result, the user is in charge to decide which modules should be rewritten and which should not be rewritten. Rewritten code can interact seamlessly with non-rewritten code, because all wrappers are completely transparent. However, the parts of the source code that are not rewritten will not trigger active expressions.

With these hooks set up, we can now create the correct dependencies between program state and active expressions during runtime. When a new active expression is created, we analyze which parts of the program state affect the expression result. To do so, we simply execute the expression with a special flag indicating its analysis. During this analysis, whenever a read access is performed, we associate the object along with its member with the current active expression. As a result, all parts of the program state that might affect the result of the given expression are now associated with the active expression.

Finally, during normal execution, we treat write accesses specially. In addition to the normal behavior, we notify all dependent active expressions about the state change, thus, achieving the desired semantics.

3 Future Work and Conclusion

In this work, we have identified a common underlying pattern in many reactive programming concepts. For the resulting class, state-based reactive programming, we designed and implemented a suitable primitive, active expressions. Active expressions combine a clean and simple callback mechanism with powerful state-monitoring

capabilities. As such, active expressions can be used to implement other state-based reactive approaches on top of it. The work reported here is quite recent, and we expect to continue to evolve both the concept of active expressions and its implementation. There are a number of directions for future work:

Comparison of Implementation Strategies Our work features multiple implementation strategies to choose from. To increase the usability of active expressions, we have to make the choice of an appropriate implementation strategy simply and communicate the resulting implications clearly. Therefore, we have to clarify the strengths and limitations of each implementation. Additionally, we will perform a quantitative analysis to determine the runtime overhead in various cases.

Usability as a Basic Concept Active expressions are designed to act as a foundation for other state-based reactive concepts. To check whether active expressions fulfill this goal, we will implement several of the presented state-of-the-art reactive concepts on top of active expressions. These implementations will highlight strengths and weaknesses of the concept. Additionally, we will explore, whether active expressions are useful as a language primitive by applying them to problem domains beyond the reimplementation of existing concepts.

Integration with Event-based Concepts Besides the identified category of state-based reactive concepts, another big family of similar mechanisms are event-based reactive concepts. Due to the discovered common ground of state-based concepts, one could use active expressions to represent state change as explicit events as required from event-based concepts. Doing so would allow to apply event-based operators on state-based concepts, which provides interesting, yet unexamined opportunities.

References

[1] C. Elliott and P. Hudak. "Functional Reactive Animation". In: *International Conference on Functional Programming (ICFP)*. ACM, 1997, pages 263–273. DOI: 10.1145/258948.258973.

[2] T. Felgentreff, A. Borning, R. Hirschfeld, J. Lincke, Y. Ohshima, B. Freudenberg, and R. Krahn. "Babelsberg/JS – A Browser-Based Implementation of an Object Constraint Language". In: *European Conference on Object-Oriented Programming (ECOOP)*. Springer, 2014, pages 411–436. DOI: 10.1007/978-3-662-44202-9_17.

[3] B. N. Freeman-Benson. "Kaleidoscope: Mixing Objects, Constraints and Imperative Programming". In: *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*. ACM, 1990, pages 77–88. DOI: 10.1145/97945.97957.

[4] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[5] M. Grabmüller and P. Hofstedt. "Turtle: A Constraint Imperative Programming Language". In: *Research and Development in Intelligent Systems (RDIS)*. Springer, 2004, pages 185–198. DOI: 10.1007/978-0-85729-412-8_14.

[6] T. Kamina, T. Aotani, and H. Masuhara. "Generalized layer activation mechanism through contexts and subscribers". In: *Proceedings of the 14th International Conference on Modularity*. ACM. 2015, pages 14–28.

[7] S. Lehmann, T. Felgentreff, J. Lincke, P. Rein, and R. Hirschfeld. "Reactive Object Queries". In: *Constrained and Reactive Objects Workshop (CROW)*. ACM, 2016. DOI: 10.1145/2892664.2892665.

[8] M. von Löwis, M. Denker, and O. Nierstrasz. "Context-oriented Programming: Beyond Layers". In: *International Conference on Dynamic Languages (ICDL)*. ACM, 2007, pages 143–156. DOI: 10.1145/1352678.1352688.

[9] I. Maier and M. Odersky. "Higher-order reactive programming with incremental lists". In: *European Conference on Object-Oriented Programming*. Springer. 2013, pages 707–731.

[10] L. A. Meyerovich, A. Guha, J. P. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi. "Flapjax: A Programming Language for Ajax Applications". In: *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*. 2009, pages 1–20. DOI: 10.1145/1640089.1640091.

[11] G. Salvaneschi, S. Amann, S. Proksch, and M. Mezini. "An empirical study on program comprehension with reactive programming". In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2014, pages 564–575.

[12] G. Salvaneschi and M. Mezini. "Reactive Behavior in Object-oriented Applications: An Analysis and a Research Roadmap". In: *Proceedings of the 12th Annual International Conference on Aspect-oriented Software Development*. AOSD '13. New York, NY, USA: ACM, 2013, pages 37–48. ISBN: 978-1-4503-1766-5. DOI: 10.1145/2451436.2451442.

[13] T. Van Cutsem and M. S. Miller. "Proxies: design principles for robust object-oriented intercession APIs". In: *ACM Sigplan Notices*. Volume 45. 12. ACM. 2010, pages 59–72.

Brain Image Analysis with convolutional Neural Network

Mina Rezaei

Internet Technologies and Systems
Hasso-Plattner-Institut
Mina.Rezaei@hpi.uni-potsdam.de

This report describes my recent activities on Service-oriented Systems Engineering in the HPI Research School and summarizes my research activities on magnetic resonance brain abnormality classification, detection and segmentation by deep learning. In this report, we explore ways of applying Deep Learning techniques for brain abnormality detection. We consider brain abnormality detection in three different stages and by using different neural network architectures. We proposed the architectures based on deep learning for most important clinical problem on medical imaging, classification, detection and segmentation.

1 Introduction

During the past years deep learning has raised a huge attention by showing promising result in some state-of-the-art approaches such as speech recognition, handwritten character recognition, image classification [3], and detection [7, 9] and segmentation [1, 4]. There are expectations that deep learning improve or create medical image analysis applications, such as computer aided diagnosis, image registration and multi-modal image analysis, image segmentation and retrieval. There has been some application that using deep learning in medical application like cell tracking [5] and organ cancer detection [6]. Doctors use magnetic resonance images as effective tools to diagnosis the diseases. Computer aided medical diagnosis can perform fast and objective with high robustness and reliability to support the health system. The brain is particularly complex structure, and analysis of brain magnetic resonance images is an important step for many diseases. Doctors use Magnetic Resonance Imaging (MRI) as an effective tool to diagnosis diseases and treatment planning. Magnetic resonance imaging (MRI) provides detailed images of the brain, and is one of the most common tests used to diagnose brain diseases. Medical application diagnosis tools make it faster and more accurate. Some of the practical applications of image segmentation in medical imaging are: local abnormal region and other pathologist, measuring tissues sizes, computer guided(/aided) surgery, diagnosis, treatment planing and study of anatomical structure.

2 Approach

An advance medical application based on deep learning methods for diagnosis, detection and semantic segmentation of brain magnetic resonance imaging (MRI) is my goal. For this work, Brain lesion diagnosis consist of several steps. Classification,

detection and segmentation are main steps. The flow chart in Figure 1 describes my plan for brain lesion detection based on deep leaning algorithms.

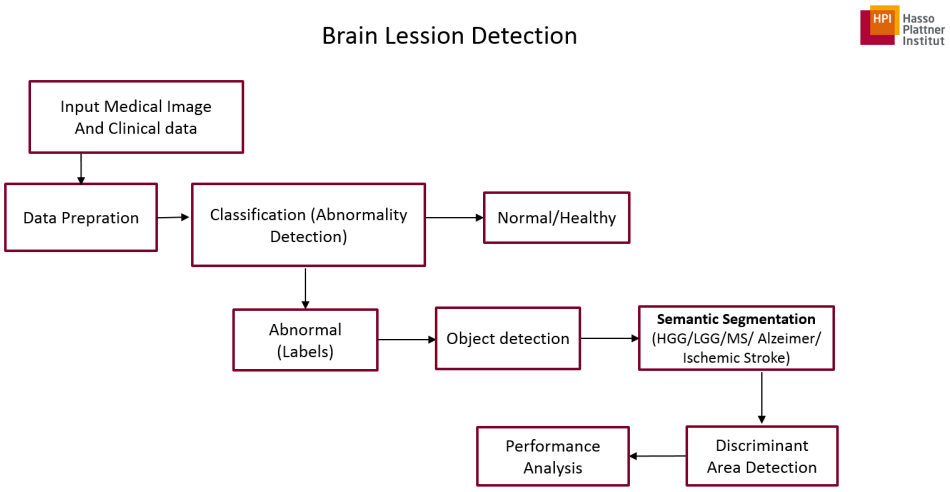


Figure 1: Overall process of brain lesion identification from MRI.

I prepared data in two steps, first volume of interest selection on MRI and next step is preparing them for classification, lesion detection or even segmentation. Depends on problem. Volume of interest (VOI) is the portion of the image in the dataset on which we want to focus. It may be either one slice or multiple slices throughout the dataset.

2.1 Classification

For classification our network architecture is based on the network proposed by Krizhevsky et al [3]. The architecture is summarized in Figure 2. Input image for the network has three channels and we use axial, coronal and axial planes in a Volume-of-Interest (VOI) for each categories. In order to prevent over fitting in the next step we exploit data augmentation. We increase convolutional layer to seven to achieve better results because it is deeper. We also consider computation complexity and the time of process by training and testing using very efficient GPU implementation of the convolution operation. The network includes three pooling layers, average pooling after 3th convolution layer and two max pool layers consequently after 5th, 6th CNN layers that has effective impact on decreasing size of the feature map. At the end of 7th convolutional layer we put three fully-connected layers which have 4096 individual neurons. We apply regularization after last fully connected layer to reduce chance of over fitting. We put in the final a 5-way SVM to classify 4 abnormalities and healthy brain images. Figure 2 presents current network for classification task.

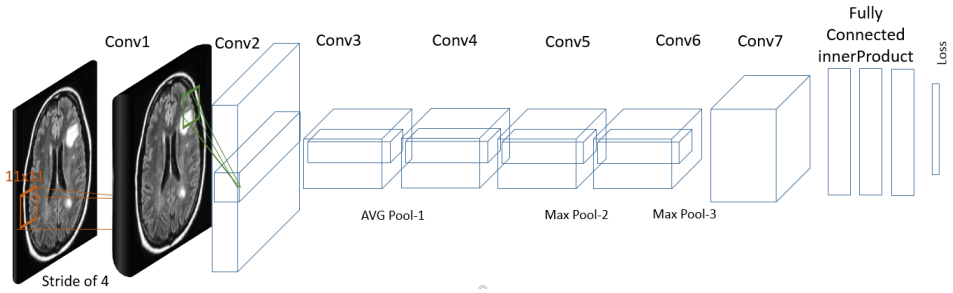


Figure 2: classifynet architecture.

Table 1: Classification result on five different MRI dataset by deep learning

	Total MRI	accuracy	sensitivity	specificity
Our propose method	1500	95.07 %	0.91	0.87

Table 2: Comparing classification result with other approaches

	Total MRI	Number of classes	accuracy
Our propose method	1500	5	95.07 %
Wavelet (DAUB-4) + PCA + SVM-RBF	75	2	98.70 %
Wavelet (Haar) + PCA + KNN	1500	2	98.60 %

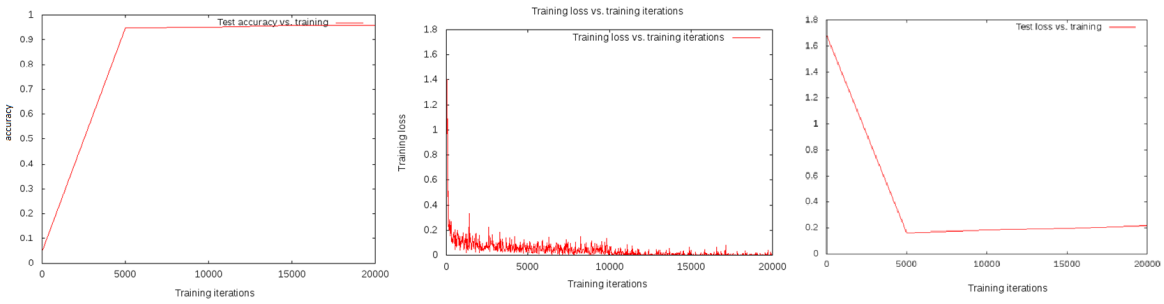


Figure 3: classifynet architecture.

2.2 Detection

Unlike image classification, detection requires localizing (likely many) objects within an image. We propose an approach for tumor high and low grade glioma classification. Our network is based on Fast-RCNN [2] but we make it six times faster. The model introduces multiple built-in subnetworks which detect Tumor types with location(s). Our network gets multi-modal MR images as input in training and testing. We consider T1 contrast, flair, and T2 as red, blue, green channels and input them as one time to make the network 3 time faster. We map the axial and coronal planes in volume of interest to Red, Green and Blue channels. Same as Roth et al. [6] we consider volume of interest of planes by following. In both types of input we make network six times faster by combination of modalities and planes. We use the VGG16 model [8] to initialize our network, which is used in the most recent state-of-the-art method. The first seven convolutional layers and three max pooling layers of the VGG16 network are used as the shared convolutional layers before the two sub-networks to produce feature maps from the entire input image. Following the Fast R-CNN, the last max pooling layer of the VGG16 network is replaced with the region of interest pooling layer to pool the feature maps of each object proposal into fixed resolution that we set as $w_i \div 9 * h_i \div 9$ same as SPP net. The final fully connected layer and softmax are replaced with two sibling fully-connected layers. For evaluation we apply our model on BRATS-2015 dataset which contains 220 subjects with high grade and 54 subjects with low grade tumor. We have achieved dice 0.72, 0.89 sensitivity and 94.3 % accuracy for whole tumor detection.

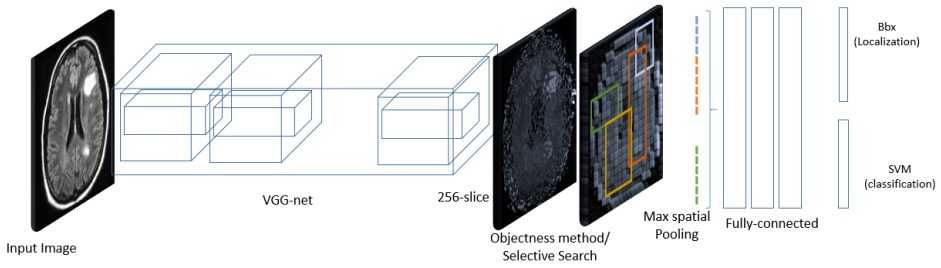


Figure 4: Brain lesion detection architecture.

2.3 Segmentation

In medical applications, image segmentation is used to classify different anatomy features, such as bones, muscle, blood vessels, soft tissue, etc. from the background and from each other. It is also used for identification of the anatomical areas of interest or as a preprocessing step for data analysis. In treatment and diagnosis of multiple sclerosis, high and low grade glioma, segmentation of regions of interest is

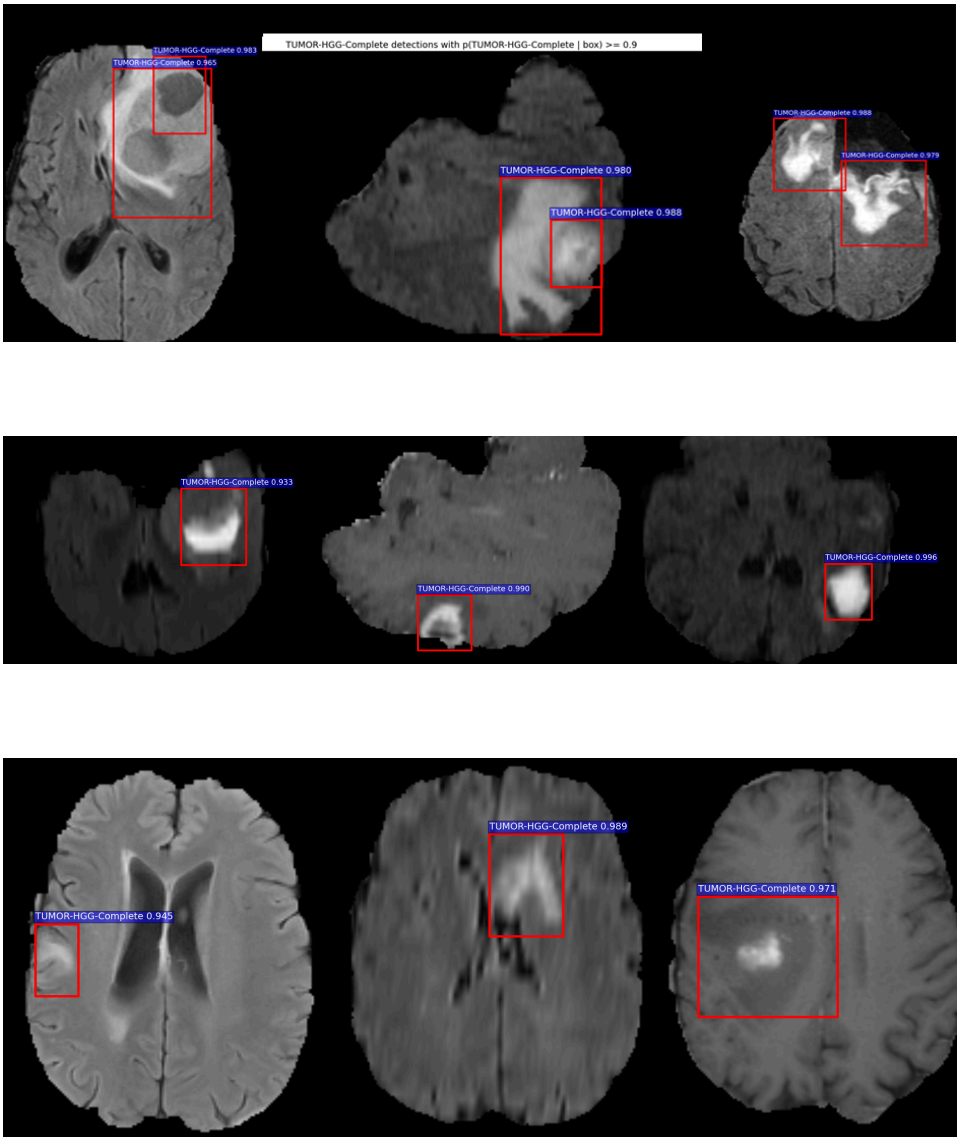


Figure 5: tumor high grade glioma detection by fast-rcnn.

Table 3: Classification result on five different MR images by deep learning

Methods	Dice	Sensitivity	Specificity
Havaei	0.84	0.84	0.84
Zhao	0.79	0.85	0.84
My Method	72.63	89.86	0.84
Tustison	0.79	0.81	0.84
Meier	0.72	0.82	0.84

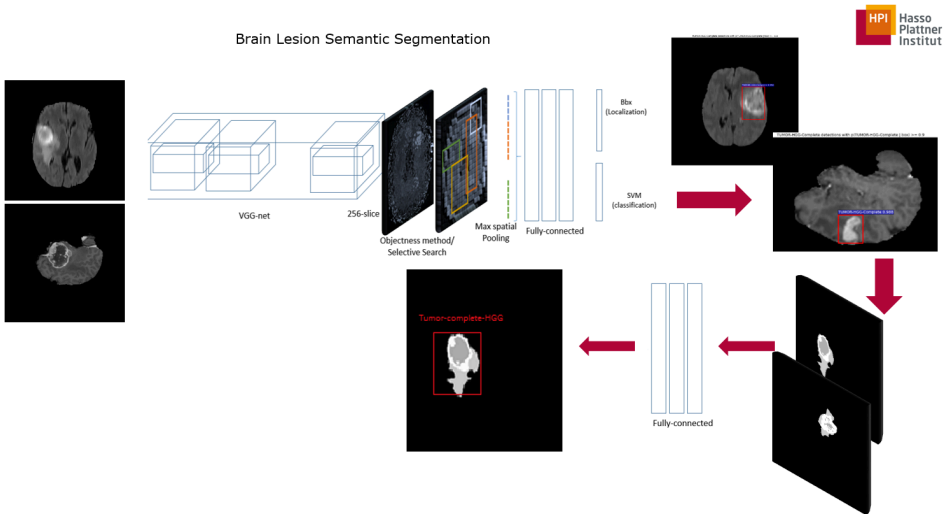


Figure 6: Brain lesion segmentation architecture.

used for tumours and lesion size estimation, calculation of thickness of the cartilage, and for visualisation for surgical planning and simulation. Image guided surgery is an important application of segmentation. Segmentation of medical imagery is challenging problem due to the complexity of the image as well as to the absence of the anatomy that fully capture the possible deformations in each structure. I continue segmentation based on my detection network and based on fully convolution network [4] which recently Microsoft AI group published it as Multi-task Network Cascades for instance-aware semantic segmentation [1]. This model consists of three networks, respectively differentiating instances, estimating masks, and categorizing objects. These networks form a cascaded structure, and are designed to share their convolutional features. I will continue segmentation based on MNC network because its based on my detection network and single-step training framework and can be generalized to cascades that have more stages and very fast as well.

3 Data Description

In this research I have used five different brain datasets to evaluate my proposed method.

1. Healthy Brain Images:[1] This data has collected nearly 600 MR images from normal, healthy subjects. The MR image acquisition protocol for each subject includes:
   - T1,
   - T2,

[1] http://brain-development.org/ixi-dataset/ (last accessed 2016-10-20).

- T1-Contrast,
- PD-weighted images (Diffusion-weighted images in 15 directions).

The data has been collected at three different hospitals in London and as part of IXI – Information extraction from Images project. The format of images is NIFTI (*.nii) and it is open access. Figure 1 column (a) shows healthy brain from IXI dataset in sagittal, coronal and axial section.

2. High and Low grade glioma(Tumor)[2] This data is from BRATS (Brain Tumor Segmentation) challenges in MICCAI conference 2015. The data prepared in two parts training and testing for high and low grade glioma tumor. All datasets have been aligned to the same anatomical template and interpolated to 1mm 3 voxel resolution. The training dataset contains about 300 high- and low- grade glioma cases. Each dataset has T1 (spin-lattice relaxation), T1 contrast-enhanced MRI, T2 (spin-spin relaxation), and FLAIR MRI volumes. In the test dataset 200 MR images without label but in the same format is available. Figure 1 column (b, c) shows high and low grade glioma, both categories are .mha format.

3. Alzheimer disease [3] The Alzheimer dataset was downloaded from Open Access Series of Imaging Studies (OASIS). The dataset consists of a cross-sectional collection of 416 subjects aged 18 to 96. For each subject, 3 or 4 individual T1-weighted MRI scans obtained in single scan sessions are included. The data format is *.hdr.

4. Multiple sclerosis[4] Multiple Sclerosis MR Images were downloaded from ISBI conference 2008 (The MS Lesion Segmentation Challenges). This dataset collected by e-Health lab of Cyprus University. Figure 6 column (e) shows multiple sclerosis images which training dataset consists of 18 multiple sclerosis as .nhdr format that ground truth (manual segmentation by expert) is available.

4 Future Work

As future work, I plan to continue work on the lesion detection and segmentation of brain lesion by convolutional neural network.

References

[1] J. Dai, K. He, and J. Sun. “Instance-aware Semantic Segmentation via Multi-task Network Cascades”. In: *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*. 2016.

[2] https://www.virtualskeleton.ch/BRATS/Start2015/ (last accessed 2016-10-20).
[3] http://www.medinfo.cs.ucy.ac.cy/index.php/downloads/datasets/ (last accessed 2016-10-20).
[4] http://www.oasis-brains.org/ (last accessed 2016-10-20).

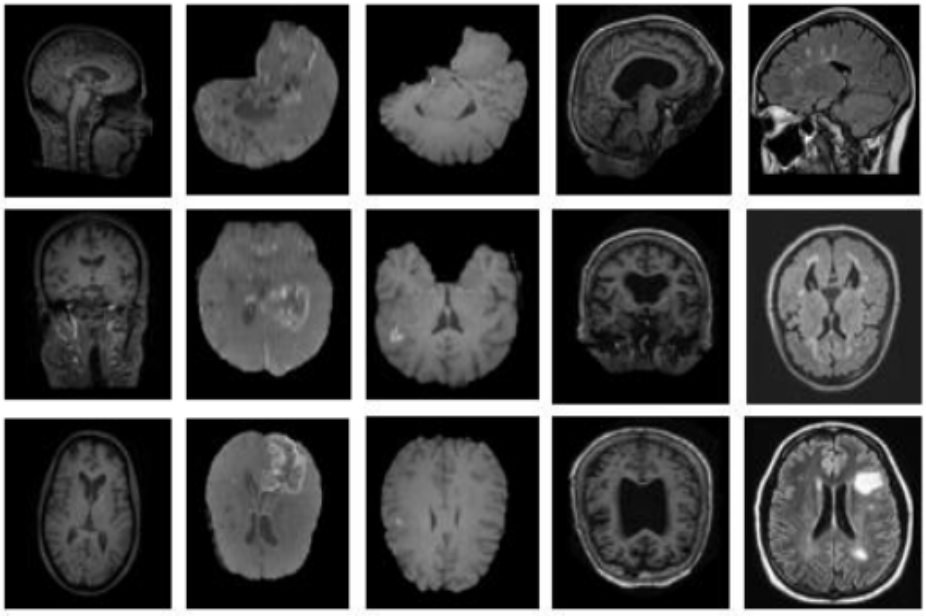


Figure 7: We trained our architecture by five different categories of brain MRI. Column (a) shows healthy brain in sagittal, coronal and axial section. Column (b) and (c) show tumor high and low grad glioma. Column (d) and (e) present some brain data on Alzheimer and multiple sclerosis.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pages 580–587. ISBN: 978-1-4799-5118-5.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pages 1097–1105.

[4] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pages 3431–3440.

[5] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer International Publishing. 2015, pages 234–241.

[6] H. R. Roth, L. Lu, A. Farag, H.-C. Shin, J. Liu, E. B. Turkbey, and R. M. Summers. "Deeporgan: Multi-level deep convolutional networks for automated pancreas segmentation". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer International Publishing. 2015, pages 556–564. ISBN: 978-3-319-24553-9.

[7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *International Conference on Learning Representations (ICLR 2014)*. CBLS. 2013, page 16.

[8] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv: 1409.1556* (2014).

[9] C. Szegedy, A. Toshev, and D. Erhan. "Deep neural networks for object detection". In: *Advances in Neural Information Processing Systems*. 2013, pages 2553–2561.

Power-Law Distributions in Random Satisfiability

Ralf Rothenberger

Algorithm Engineering
Hasso-Plattner-Institut
ralf.rothenberger@hpi.de

One of the most fundamental problems in computer science is Propositional Satisfiability (SAT). Its computational hardness gave rise to many complexity-theoretical concepts, e.g. NP-hardness and lower bounds on algorithmic runtime via the (Strong) Exponential Time Hypothesis. In order to analyze the average-case complexity of SAT and to generate benchmarks for solvers, instances are created at random (random SAT). Random SAT initiated the development of sophisticated rigorous and non-rigorous techniques for analyzing random structures. Despite a long line of research and substantial progress, nearly all theoretical work on random SAT assumes a *uniform* distribution on the variables. In contrast, real-world instances often exhibit community structure and large fluctuations in variable occurrence, similar to big real-world networks.

We want to develop and analyze more sophisticated models for the creation of realistic random SAT instances. This would provide a means of creating realistic benchmarks for SAT solvers on a large scale as well as giving insights on the structural properties of such instances. Our hope is that techniques for the creation and analysis of scale-free networks can be transferred to the domain of random SAT. This report documents our work analyzing a first model proposed by Ansótegui et al. [3]. For this model we rigorously proved that the satisfiability of generated formulas does not only depend on the number of clauses, but also on the power-law exponent of the variable distribution. This is a stark contrast to the behavior of formulas created in the uniform random SAT model. Furthermore, we experimentally validated this behavior.

1 Overview

Over the last decades it was observed that many real-world networks exhibit degree distributions that follow a power-law, including Internet topologies, the Web, social networks, power grids, and literally hundreds of other domains. Furthermore, it was shown that these networks feature a unique set of additional properties besides their power-law degree distribution. These properties include the existence of a giant component, which contains all but a linear fraction of nodes, small average distance between two nodes, normally double-logarithmically in the number of nodes, community structure, i.e. the emergence of densely connected subgraphs which are only sparsely connected to the rest of the network, and the existence of hub nodes, which are connected to a large fraction of nodes in the network[10].

One domain, where problem instances with these properties occur is the satisfiability of propositional formulas (SAT). For *industrial instances* it was recently shown, that their clause- and variable distributions resemble power-laws[2] and that they exhibit community structure[4]. Industrial instances arise from problems in practice,

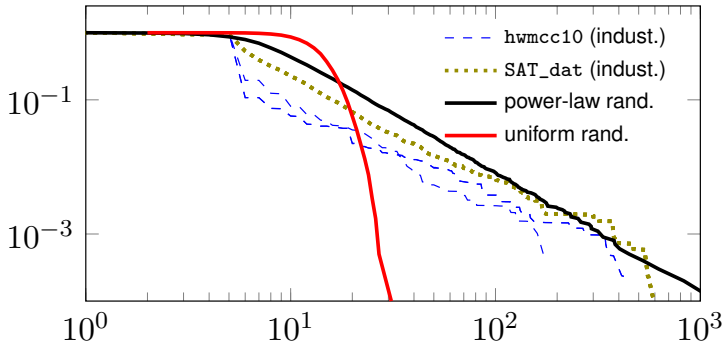


Figure 1: Cumulative variable occurrence distributions of two industrial categories from SAT Race 2015 compared to a random power-law formula ($n = 10000$, $m = 45000$, $\beta = 2.75$) and a uniform random formula ($n = 10000$, $m = 45000$). The distributions of the industrial instances are much closer to the random power-law formula than to the uniform random formula.

such as hardware and software verification, automated planning and scheduling, and circuit design. Although SAT is NP-complete in theory, even *large industrial instances* with millions of variables can often be solved very efficiently by modern SAT solvers. This implies that their structural properties might be beneficial for the runtime of some classes of solvers.

We would like to develop models, which generate synthetic formulas that are similar to real ones, and to analyze these formulas. Having realistic models would provide both a means of creating large-scale benchmarks for solvers as well as a foundation for rigorous analysis. The purpose of an analysis is to validate properties of generated instances against their real-world counterparts and to design heuristics to efficiently solve generated instances. Our hope is that the results and algorithms can be transferred to the original real-world problems. This might improve, or at least explain, the effectiveness of state-of-the-art solvers.

In this report we summarize and discuss our results analyzing a model for random formulas with power-law distributed variable occurrences proposed by Ansótegui et al. [3]. These results are currently under review and not published yet.

2 Preliminaries and Related Work

For modeling typical inputs, we study random propositional formulas. In random satisfiability, we have a distribution over Boolean formulas in conjunctive normal form (CNF). Normally, the CNFs are also restricted to having clauses with a certain number k of literals (k-CNFs).

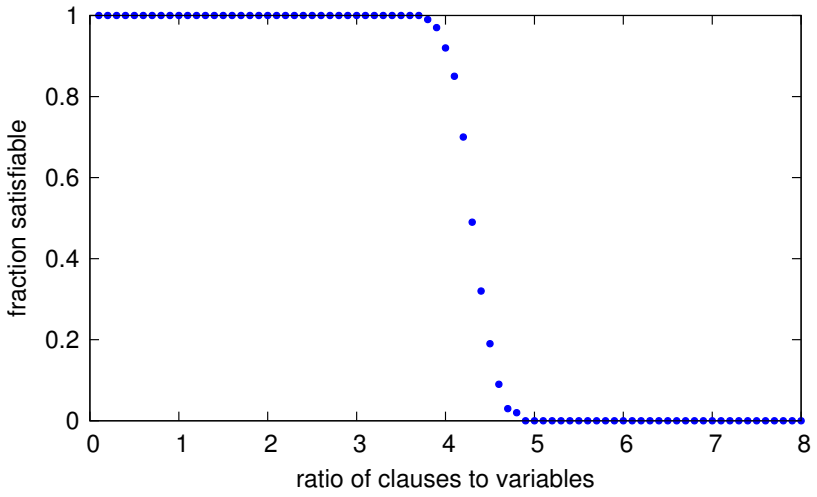


Figure 2: Fraction of satisfiable formulas for uniform random SAT with clause length $k = 3$ and $n = 100$ variables. Each data point created with 500 random formulas.

2.1 Uniform Random SAT

In the uniform model, a random k-CNF is created as follows:

1. Repeat m times:
   a) Choose a k out of n variables uniformly at random. Repeat if at least one variable is chosen twice.
   b) Negate each variable independently at random with probability $1/2$.
   c) Connect the resulting k literals via disjunctions to form a clause.
2. Connect the resulting m clauses via conjunctions to form a propositional formula in conjunctive normal form.

Uniform random k-SAT is being analyzed for over thirty years now and there are many papers concerning the model.

One of the oldest and most prominent research questions is proving the *Satisfiability Threshold Conjecture*. The conjecture states that for each $k \geq 2$ there exists a constant ratio r_k of clauses to variables such that random k-CNFs are almost surely[1] satisfiable if $\frac{m}{n} < r_k - \varepsilon$ and almost surely unsatisfiable if $\frac{m}{n} > r_k + \varepsilon$ (c.f. Figure 2). Friedgut [8] showed that the threshold is sharp, i.e. that there only is a single value $r_k(n)$ instead of a whole range. Note that this does not prove the Satisfiability Threshold Conjecture, since it does not prove that there is a limiting value of $r_k(n)$ for $n \to \infty$. Chvátal and Reed [5] proved that the threshold is at $r_2 = 1$ for $k = 2$. For $k \geq 3$ the exact location has not been proven rigorously, although the approximate

[1] By *almost surely* we denote a probability going to one as the problem size goes to infinity. In the context of random SAT the problem size is n, the number of variables.

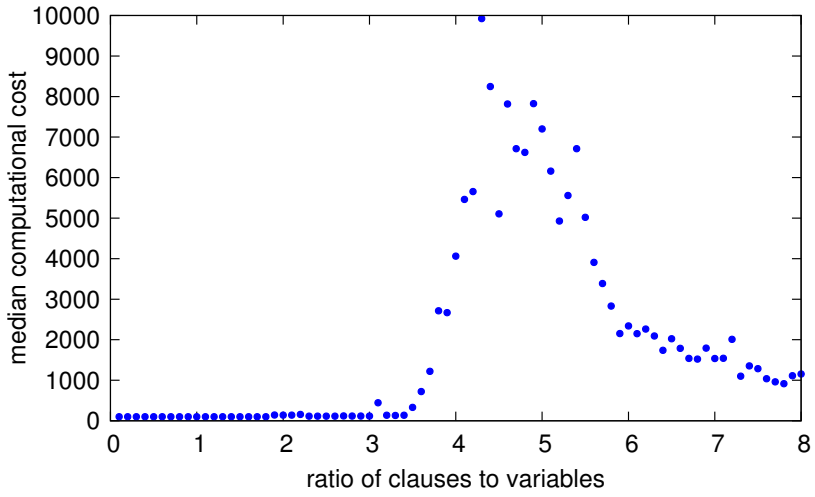


Figure 3: Median number of propagations when solving with MiniSAT (no preprocessing or randomization) for uniform random SAT with clause length $k = 3$ and $n = 100$ variables. Each point is the median of 500 random formulas.

location of the threshold can be determined experimentally for fixed values of k. Recently, Coja-Oghlan [6] proved $r_k = 2^k \ln 2 - \frac{1}{2}(1 + \ln 2) + o(1)$ for large k.

Determining the location of the satisfiability threshold can give insights about structural properties of the random formulas. Furthermore, it can be observed that the runtime of SAT solvers is especially high around the satisfiability threshold [13] (c.f. Figure 3). Therefore, knowing the location of the threshold can be helpful when trying to generate difficult benchmarks.

2.2 Relevant Non-Uniform Models

There are two models proposed so far, which each capture one of the properties of industrial formulas.

The *Community Attachment Model* by Giráldez-Cru and Levy[9] creates random formulas with clear community structure, but lacking a power-law distribution of variable frequencies. Furthermore, the work of Mull et al.[14] shows that community structure alone is not sufficient to explain the efficiency of industrial SAT solvers based on conflict-driven clause learning (CDCL). More precisely, they show that instances generated by the Community Attachment Model have exponentially long resolution proofs with high probability, making them hard for CDCL on average.

The model by Ansótegui et al.[3] creates random formulas with power-law distributed variable frequencies, which lack the community structure observed in [4]. It generates k-CNF formulas in the same way as the uniform model with the difference that variables are now chosen according to a power-law distribution instead of uniformly at random. In the following we will call this model *power-law random SAT*. The authors also propose a way to create random CNF-formulas with power-law distributed variable frequencies and power-law distributed clause lengths. They show

experimentally that the runtime of solvers on formulas generated in their models resembles that of the same solvers on real industrial instances, but they do not show any theoretical results. In fact, prior to our work there were no theoretical results for power-law random SAT at all.

3 Our Results

We analyze the power-law random SAT model and show several results regarding its satisfiability threshold. A summary of our results can be seen in Figure 4.

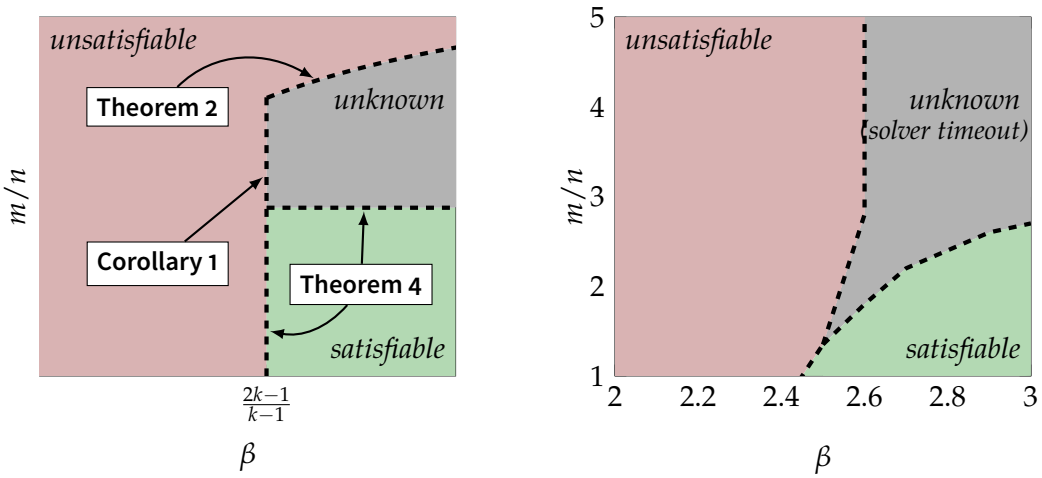


Figure 4: Illustration of our asymptotic results for the power-law satisfiability threshold location when $n \to \infty$ (left) compared with empirical results for randomly generated power-law 3-SAT formulas on $n = 10^6$ variables checked with the SAT solver MiniSAT (right). The timeout was set to one hour.

3.1 Upper Bounds

A first step in trying to understand the behavior of the satisfiability threshold in power-law random SAT is to try and derive some upper bounds on it. That is, we want to see up to which clause-variable-ratios $\frac{m}{n}$ random power-law CNFs are still satisfiable. Surprisingly, our first result already establishes, that for some ranges of power-law exponents there *does not exist* a satisfiability threshold.

Corollary 1. *Let Φ be a random k-SAT formula that follows an arbitrary power-law distribution. If the power-law exponent is $\beta \leq \frac{2k-1}{k-1} - \varepsilon$ for an arbitrary $\varepsilon > 0$, Φ is unsatisfiable with high probability.*

The intuition behind this is simple: If the power-law exponent is too small, then there are k variables with very high probabilities. In fact, their probabilities are so high that they appear together in a polynomial number of clauses. Especially, they appear together with all 2^k different combinations of signs, therefore making it impossible to satisfy all their clauses.

To see how the threshold behaves for the regime of power-law exponents $\beta > \frac{2k-1}{k-1}$, we use methods from the analysis of uniform random SAT. A first upper bound can be derived from the probabilistic method: The expected number of satisfying assignments for a random formula on *any* probability distribution is $2^n(1-2^{-k})^m$, as long as clauses are drawn independently at random and each variable is negated with probability $1/2$. Therefore, if the clause-variable ratio is higher than $\ln(2)/\ln(\frac{2^k}{2^k-1})$, the formula will be unsatisfiable with high probability.[2]

However, this constant is rather large: For random 3-SAT this yields an upper bound of ≈ 5.191, whereas the true value for power-law random SAT seems to be much smaller (c.f. subsection 3.3).

To get a better upper bound, we use the so called Single-Flip Method introduced by Kirousis et al. [11]. The main idea of the method is the following: Instead estimating the expected number of all satisfying assignments, we only estimate the expected number of satisfying assignments which have the single-flip property. The property states that by flipping any single bit of the assignment from zero to one, we do not get another satisfying assignment. These always exist, if the formula is satisfiable and since their number is a lot smaller than the total number of satisfying assignments, we also get a smaller expected value, which in turn gives us a lower upper bound.

By generalizing this method to arbitrary probability distributions, we get the following result:

Theorem 2. *Let Φ be a random k-SAT formula with $k \geq 2$ and $r = \frac{m}{n}$ that follows a power-law distribution. Let further $N \in \mathbb{N}^+$ be any constant. If the power-law exponent is $\beta > 2$, then Φ is w. h. p. unsatisfiable if*

$$\left(\left(1-\tfrac{1}{2^k}\right)^r 2^{\frac{1}{N}} \prod_{l=1}^{N-1}\left[2-\exp\left(-(1+o(1))\, r\frac{k}{2^k-1}\frac{\beta-2}{\beta-1}\left(\frac{N}{l}\right)^{\frac{1}{\beta-1}}\right)\right]^{\frac{1}{N}}\right) < 1.$$

Unfortunately, we cannot solve this expression analytically, but in Table 1 we numerically determined the smallest values of r such that the formulas are unsatisfiable. Furthermore, we compare these values to the upper bounds the Single Flip Method gives for uniform random SAT.

3.2 Lower Bounds

To complement our upper bound on the satisfiability threshold for $\beta > \frac{2k-1}{k-1} + \varepsilon$, we now want to show that there also is a constant lower bound. Since this is a

[2] We say that an event E holds *with high probability* (w. h. p.), if there exists an $\delta > 0$ such that $\Pr[E] \geq 1 - \mathcal{O}(n^{-\delta})$.

Table 1: Numerical upper bounds on the satisfiabilty threshold obtained from the Single-Flip Method (cf. Theorem 2). Empty fields indicate unsatisfiability for *all* constant densities by Corollary cor:unsat. The values for uniform random SAT are obtained from the Single-Flip Method. To the best of our knowledge, the bounds for uniform random SAT with $k \geq 4$ are the currently best known numerical upper bounds. For $k = 3$ the best known unconditional numerical upper bound is 4.4898 [7].

	power-law distribution with exponent β								**uniform dist.**
k	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	
3				3.48	3.71	3.87	3.99	4.08	4.67
4			7.87	8.42	8.78	9.04	9.23	9.37	10.23
5		16.27	17.75	18.64	19.21	19.61	19.90	20.11	21.33
7	67.21	75.74	79.81	82.09	83.49	84.42	85.07	85.54	87.88
10	619.28	662.48	680.93	690.36	695.77	699.12	701.34	702.88	708.94

difficult task even in the arguably simpler uniform random SAT model, we start by considering $k = 2$. Although 2-SAT can be solved in polynomial time, it exhibits the same threshold behavior in the uniform case. Furthermore, uniform random 2-SAT is the only case with matching lower and upper bounds on the satisfiability threshold. This is due to the fact that the structure of 2-CNFs is not as complicated as that of formulas with higher clause lengths.

By adjusting a proof technique by Chvátal and Reed [5] to power-law random SAT we are able to establish the following result:

Theorem 3. *Scale-free random 2-SAT with power-law exponent $\beta > 3$ and clause-variable ratio $m/n < \frac{(\beta-1)(\beta-3)}{(\beta-2)^2}$ is satisfiable with probability $1 - o(1)$.*

The proof of this statement utilizes the notion of bicycles introduced by Chvátal and Reed [5]. A bicycle is a certain sub-formula which appears in every unsatisfiable 2-CNF. With the probabilistic method, we can show that the expected number of bicycles is smaller than 1 as long as $\beta > 3$ and $m/n < \frac{(\beta-1)(\beta-3)}{(\beta-2)^2}$. Using Markov's inequality, this gives us the probability bound. As can be seen in Figure 5 this lower bound matches the threshold we observe in practice quite well.

Now we can use the former statement to show a similar lower bound for k-SAT with $k \geq 3$.

Theorem 4. *Let Φ be a random k-SAT formula that follows an arbitrary power-law distribution. If the power-law exponent is $\beta > \frac{2k-1}{k-1} + \varepsilon$ for an arbitrary $\varepsilon > 0$, Φ is satisfiable with high probability if $\frac{m}{n}$ is a small enough constant.*

The proof idea is the following: From each k-clause we choose the two variables with lowest probability to form a 2-clause. These 2-clauses are then conjoined to form a 2-CNF. We can show that, if the variable distribution of the original formula follows a power-law with exponent at least $\frac{2k-1}{k-1} + \varepsilon$, then the variable distribution in the

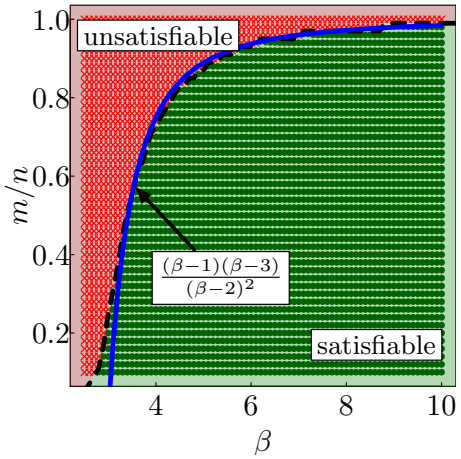


Figure 5: Phase diagram for scale-free 2-SAT formulas with $n = 10^7$ variables. We empirically observe a sharp phase transition (– –), which closely matches the theoretical bound of Theorem 3 (—).

resulting 2-CNF is dominated by a power-law with exponent bigger than $3 + \varepsilon'$. Due to Theorem 3 this means, that there has to be a clause-variable-ratio below which the reduced 2-CNF is almost surely satisfiable. Since every satisfying assignment for the 2-CNF also satisfies the original k-CNF, the original formula is almost surely satisfiable below the same clause-variable-ratio, thus proving the statement.

3.3 Experiments

We did extensive experiments to study the behavior of power-law random SAT and to validate our theoretical results. The diagrams in Figure 5 and in the left panel of Figure 6 are generated as follows. To see the asymptotic behavior, we choose n very large, $n = 10^7$ for $k = 2$ and $n = 10^6$ for $k = 3$. 50 scale-free formulas are generated for each exponent $\beta = 1.5, 1.6, \ldots, 3.5$ and each m such that $m/n = \frac{1}{10}, \frac{2}{10}, \ldots, 10$. The resulting formulas are then solved by the CDCL-based solver `MapleCOMSPS` [12] with a time cutoff of 900 seconds. We chose `MapleCOMSPS` because of its good performance on the Application Benchmarks at the SAT 2016 competition. If the solver does not finish in time, the formula's satisfiability is unknown and it is marked as "hard". If all 50 formulas are unsatisfiable, a red cross (×) is drawn. If some formulas are satisfiable, a green dot (●) is drawn with the dot's size corresponding to the fraction of satisfiable instances. Note that the threshold appears to be sharp, and in most cases, either all formulas are satisfiable, all are unsatisfiable, or all are hard.

4 Conclusion and Future Work

We analyzed the power-law random SAT model proposed by Ansótegui et al. [3]. For the case of 2-SAT we were able to prove a lower bound on the satisfiability threshold

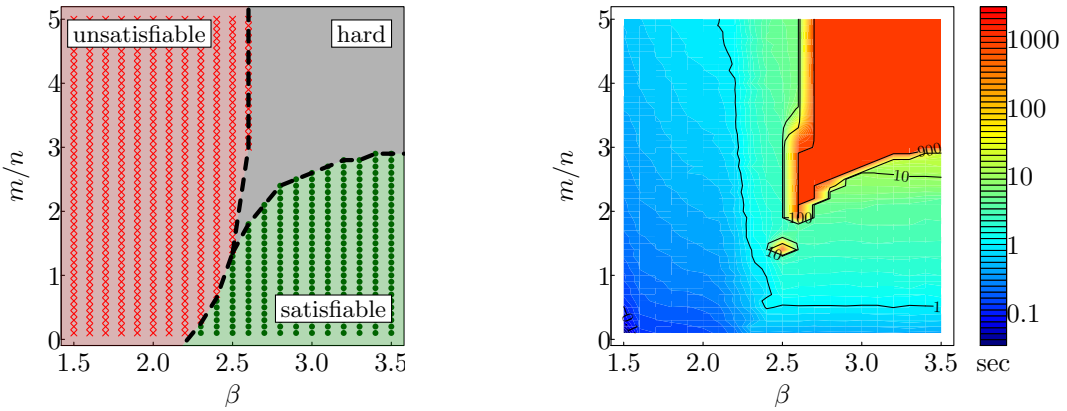


Figure 6: Phase diagrams (**left**) and timing contour plots (**right**) for scale-free k-SAT with $n = 10^6$ and $k = 3$. The phase diagrams show a clear phase transition from unsatisfiable (×) to satisfiable (●) and a patch of very hard instances (■), close to the phase transition for higher m/n and β. The contour plots and heat maps in the right column report mean solver time on the formulas (blue=fast, red=slow); solver run time strongly increases around the phase transition.

which depends on the power-law exponent of the variable distribution and seems to be tight in practice. For k-SAT with $k \geq 3$ we showed that only the regime of distributions with power-law exponent bigger than $\frac{2k-1}{k-1}$ exhibits a satisfiability threshold, while distributions with a lower exponent almost surely generate unsatisfiable formulas. Furthermore, for power-law exponents only slightly above $\frac{2k-1}{k-1}$ the threshold is smaller than the one for uniform random SAT (see Table 1).

At the time of writing we are trying to prove a matching upper bound for the lower bound in Theorem 3. Furthermore, we are trying to show a sharpness result for power-law random SAT similar to the one by Friedgut [8]. This would enable us to use second moment techniques similar to the ones in [1] to derive better lower bounds on the satisfiability threshold of power-law random SAT. Also, we would like to try and propose a model which combines community structure with power-law distributed variable frequencies. Furthermore, it would be beneficial to discover additional structural properties of real-world instances, which can be incorporated into a model for generating formulas.

References

[1] D. Achlioptas and C. Moore. "Random k-SAT: Two Moments Suffice to Cross a Sharp Threshold". In: *SIAM J. Comput.* 36.3 (2006), pages 740–762.

[2] C. Ansótegui, M. L. Bonet, and J. Levy. "On the Structure of Industrial SAT Instances". In: *Principles and Practice of Constraint Programming – CP 2009, 15th International Conference*. 2009, pages 127–141.

[3] C. Ansótegui, M. L. Bonet, and J. Levy. "Towards Industrial-Like Random SAT Instances". In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*. 2009, pages 387–392.

[4] C. Ansótegui, J. Giráldez-Cru, and J. Levy. "The Community Structure of SAT Formulas". In: *Theory and Applications of Satisfiability Testing – SAT 2012 – 15th International Conference*. 2012, pages 410–423.

[5] V. Chvátal and B. A. Reed. "Mick Gets Some (the Odds Are on His Side)". In: *Proc. of the 33rd Annual Symposium on Foundations of Computer Science*. 1992, pages 620–627.

[6] A. Coja-Oghlan. "The asymptotic k-SAT threshold". In: *Symposium on Theory of Computing, STOC 2014*. 2014, pages 804–813.

[7] J. Díaz, L. M. Kirousis, D. Mitsche, and X. Pérez-Giménez. "On the satisfiability threshold of formulas with three literals per clause". In: *Theor. Comput. Sci.* 410.30-32 (2009), pages 2920–2934.

[8] E. Friedgut. "Sharp thresholds of graph properties, and the *k*-sat problem". In: *J. Amer. Math. Soc.* 12.4 (1999). With an appendix by Jean Bourgain, pages 1017–1054. ISSN: 0894-0347.

[9] J. Giráldez-Cru and J. Levy. "A Modularity-Based Random SAT Instances Generator". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*. 2015, pages 1952–1958.

[10] R. van der Hofstad. *Random graphs and complex networks*. 2011. URL: http://www.win.tue.nl/~rhofstad/NotesRGCN.pdf (last accessed 2016-10-20).

[11] L. M. Kirousis, E. Kranakis, D. Krizanc, and Y. C. Stamatiou. "Approximating the unsatisfiability threshold of random formulas". In: *Random Struct. Algorithms* 12.3 (1998), pages 253–269.

[12] J. H. Liang, C. Oh, V. Ganesh, K. Czarnecki, and P. Poupar. "MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB". In: *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. Volume B-2016-1. Department of Computer Science Series of Publications B, University of Helsinki. 2016, pages 52–53.

[13] D. G. Mitchell, B. Selman, and H. J. Levesque. "Hard and Easy Distributions of SAT Problems". In: *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.* 1992, pages 459–465.

[14] N. Mull, D. J. Fremont, and S. A. Seshia. "On the Hardness of SAT with Community Structure". In: *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference*. 2016, pages 141–159.

Matching Unstructured Product Offers to a Product Catalog (A Case Study)

Ahmad Samiei

Information Systems Group
Hasso-Plattner-Institut
ahmad.samiei@hpi.uni-potsdam.de

Product matching is a recent, and more challenging variation of entity resolution, or duplicate detection, which links product offers from different sources to their corresponding entries within a product catalog. Tens of millions of offers for a wide variety of product categories are daily collected by price comparison websites. This data is highly heterogeneous, because product offers are gathered from thousands of different retailers and most use different attributes and styles describing their offers. Data heterogeneity and variety of the product categories makes the task more difficult than conventional duplicate detection, which usually deals with data from a specific domain. Moreover, the velocity of data updates and volume of the data make the task even harder.

Prior work typically deals with this problem using different text similarity measures or machine learning approaches for a dataset of a specific product category. In this work, we initially attempt to adapt these approaches to the problem of having a larger dataset that contains a wider range of product categories.

1 Product Matching

With the increasing popularity of the internet, the amount of eCommerce has also grown tremendously during last decade. Based on a recent study by eMarketer, the amount of worldwide retail eCommerce sales will reach $1.9 trillion by the end of 2016, and its yearly double digit growth will continue until 2020 [20]. The internet has already achieved an important position in retail commerce and there are tens of thousand online retailers, which sell hundreds of millions of products across the globe. This number of online shops has already made finding a particular product a difficult task for the consumers. That prevails mainly because there is not any universal product identifier for all of the products. As a result, many product search engines and price comparison websites (e.g. Amazon, Google's product search engine shopping.bing.com) have emerged to help the consumer in this regard, by aggregating and clustering product offers from different online retailers.

Offers from different retailers have heterogeneous data formats and sparse information. There is no global unique identifier for all of the products that could be simply used as a key to link different offers of a specific product. Offers also contain a few textual attribute fields. These fields are usually very short. In addition to missing attribute values, offer data often have many misspellings and abbreviations, which make product matching even more challenging task. Product matching attempts to identify product offers with different syntactical representations that refer to the

same real-world entity, and place them in the same cluster. The aim of this work is to explore datasets for our partner idealo,[1] a leading online product comparison website. We plan to investigate different approaches and devise effective similarity functions and necessary similarity indices. They should improve accuracy of the product matching task in terms of precision and recall in comparison to state of the art methods, yet retain the efficiency. Initially, we examine text-similarity based approaches, and then we will investigate different machine learning approaches in combination with text similarity measures.

2 Related Work

Duplicate detection has being investigated under different names such as entity resolution, record linkage, purge and merge, reference matching, and reference reconciliation [8, 11]. Product matching is a rather new sub-field in duplicate detection that deals with the product data. Literature in this field could be classified in two main categories as follows:

Text similarity based approaches Work in this category solely relies on text similarity measures to discover duplicates in product offers. They aim at finding the most suitable similarity metrics for the product matching task. They typically employ general similarity metrics, tailor or combine some of them together so that the newly generated similarity measures produce better results. For instance, Bezu et al. propose a multi-component similarity measure to compute the similarity of product offers [4]. In their work authors combine two already existing approaches, Title Model Word Method (TMWM) [19] and weighted average of Key-Value attribute Pairs (KVP) similarity method [1]. Application of this combined approach followed by agglomerative clustering on the output shows significant improvement over any of the former methods independently. The combined approach also outperforms another baseline method, which basically applies TF-IDF on the offer attribute values and uses cosine similarity to measure the similarity of two products. In order to decrease the number of redundant comparisons in the method introduced in [4], Dam et al. propose to leverage Locality Sensitive Hashing (LSH) in order to preselect potential duplicates. Their evaluation show that utilizing LSH improves the performance significantly, however it sacrifices the quality (i.e. recall, F-measure) of the result to some extent [10].

Hybrid approaches This category of work typically utilizes various machine learning methods in combination with similarity measures to discover duplicates. For instance, Kannan et al. try to match unstructured product offers to structured product specification exploiting a binary logistic regression model. They start with semantically parsing the offer descriptions to extract attribute values. Thereafter, by applying

[1] https://www.idealo.de (last accessed 2016-10-20).

a similarity function on two attribute vectors of a product and an offer, they construct similarity feature vector for each pair. Subsequently, as a matching function they train a binary logistic regression model using the already constructed similarity feature vectors and a manually labeled training set. Finally, the trained model is used to match unforeseen offers to the product catalog [13].

Köpcke et al. evaluate and show that offer matching is a harder task than conventional entity matching, which usually deals with well-structured and a domain specific datasets [15]. In a series of work the authors attempt to detect duplicates in a totally unstructured product offer dataset, which they believe is even harder than mapping the product offers to a structured product catalog. In their proposed approach they start with an extensive pre-processing step where they try to extract product code from the offer titles and verify it through an external knowledge source. They observed that product codes, if exist, greatly help to discern similar but different products. The authors continue pre-processing step to extract manufacturer name and other relevant features from the product offer textual fields. They use TF-IDF, Trigram Jaccard as string similarity measures for title and description attributes and a specific similarity measure for the extracted product code. As a final step, they train a SVM classifier using different strategies to select training sets [16]. In another work, Köpcke et al., investigate the effect of using different classifiers: decision tree, logistic regression, SVM, and hybrid classifiers with respect to the result quality. Based on their observation, none of the latter performs well over all different datasets, whereas a hybrid classifier outperforms all other individual classifiers [12].

Most of the already mentioned work uses a specific category of the products, namely electronic products in addition to a general dataset, which contains products from a wider spectrum of categories to evaluate their approaches. The evaluation shows a significant difference in performance between these two categories and specific categories produce much higher result quality.

In addition to these two categories, additional related work focuses on other aspects of the product matching, such as [2, 18] which investigate on devising adaptive similarity measures for product matching. Product classification techniques proposed in [3, 5, 14] are basically approaches used to increase efficiency of the matching process by limiting the number of comparisons. Product classification also could be used to construct a list or hierarchical product catalog or incrementally update it if exists.

3 Case Study (Product Matching in idealo)

idealo is the largest price comparison website in Germany. It daily receives over 650 million unique product offers from more than 58 thousand online retailers. These retailers periodically send snapshot of their entire database or an increment including any changes from their last data transfer (i.e., update, delete, and insert new records). As the data mainly arrives as a continuous stream, to be competitive idealo has to process the data and match them to the product catalog or other similar product offers in real-time and very accurately. Hence product matching is a key feature of

Table 1: Structured product record of the product catalog

Attribute Name	Attribute Value
category	digital camera
product name	Panasonic Lumix DMC-FX07
sensor resolution	7 megapixel
color	silver
weight	132 g
price: minimum	$ 249.99
price: maximum	$ 286
display: type	LCD display
display: diagonal size	2.5 in
flash memory: storage capacity	8 MB
lens system: optical zoom	3.6
...	

Table 2: Sample offers

offers	textual attribute, product name (title)
offer-1	Panasonic Lumix DMC-FX07 digital camera [7.2 megapixel, 2.5″, 3.6x optical zoom, LCD monitor]
offer-2	Panasonic DMC-FX07EB digital camera silver
offer-3	Lumix DMC-FX07EB-S 7.2 MP

idealo and any other product comparison websites. In order to enhance product matching, idealo manually maintains a product catalog, a well-structured dataset consists of key-value pairs for the product attributes. Table 1 presents a sample product from the catalog. idealo's current product catalog contains over 1.8 million entries, which on average contain 12 attributes per entry.

In contrast to the products, offers are not well-structured and only have two textual fields, title and description. The title contains essential information but it is very short (Figure 1, shows distribution of the offer titles length extracted from our evaluation dataset). The description field usually contains attribute names and values. Besides these two fields, offers might contain EAN[2] and ASIN[3] unique identifiers.

However, these unique identifiers exist, they are not widespread. Within the product offers, which idealo receives only about 8 % of them contain one of theses identifiers. Furthermore, presence of theses identifiers does not guarantee that two products having the same identifier are necessarily identical. It is observed that two offers with the same identifier might refer to two different products or on the contrary two

[2]European Article Number.
[3]Amazon Standard Identification Numbers.

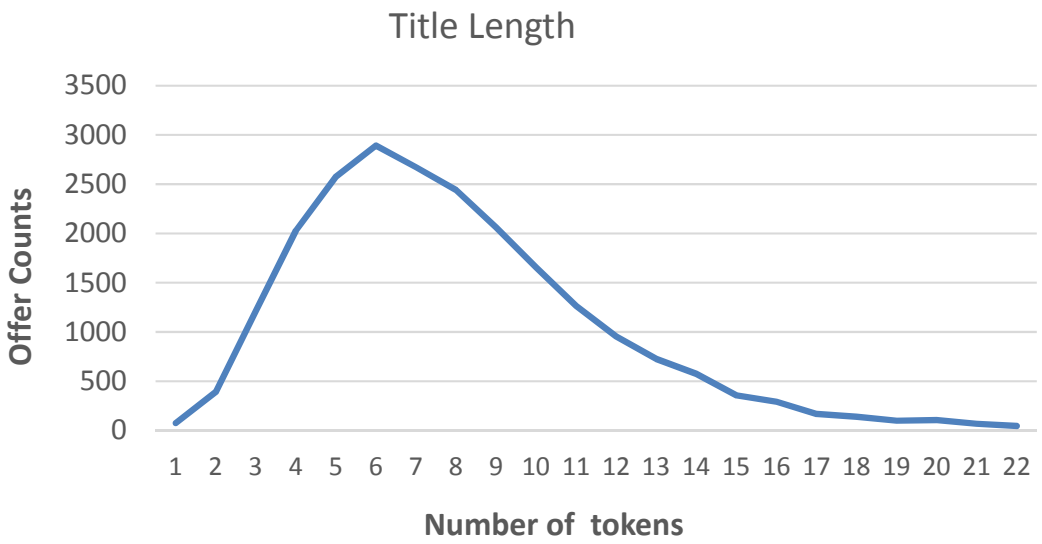


Figure 1: The length of the offer titles in the evaluation dataset.

different identifiers might also refer to a unique product. Table 2, shows title field of three sample offers for the product presented in the Table 1.

As can be seen, out of 3 offers, offer-1 contains the most detailed information, yet it contains a small subset of the attribute values of the corresponding product, and some of the values are not exact match. For instance, 7.2 megapixel vs. 7 megapixel or LCD monitor vs. LCD display. Offer-2 has a smaller subset of the attributes than offer-1 has. It does not provide part of the product name, and its model name also has an extra suffix EB, *Panasonic DMC-FX07EB* vs. *Panasonic Lumix DMC-FX07*. Offer-3 does not provide brand name, and misses part of the model name and instead adds a suffix to it, *Lumix FX07EB-S* vs. *Panasonic Lumix DMC-FX07*. In addition it uses an abbreviation *MP* instead of *megapixel*. Clearly we have to deal with a difficult problem of matching the offers with free text format to the structured product data. In summary, any product matching approach has to address the following issues:

- Wide variety of product categories (millions of diverse products).
- Huge degree of heterogeneity, because product offers are collected from thousands of different retailers which usually use different attributes, title and description.
- Textual nature of the title and description fields, which requires parsing and extracting attribute values.
- Missing or wrong values, abbreviations in attribute values.

We start our work with the following approaches:

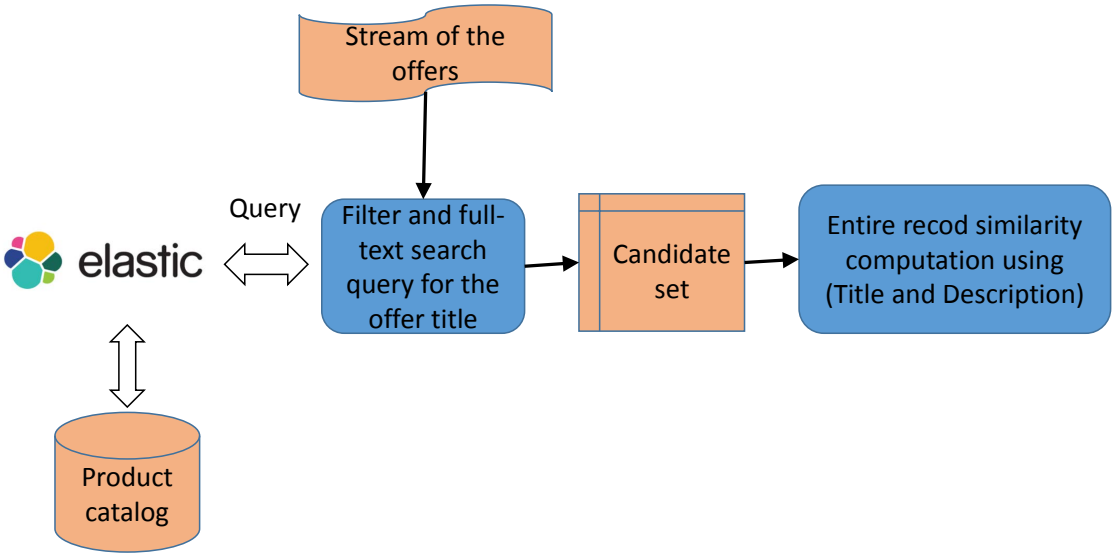


Figure 2: Product Matching Workflow for a new Offer.

3.1 Baseline approach

Conventional duplicate detection approaches are perfectly suitable for the schema based datasets that consist of multiple fields. These fields could be used for blocking or sorting to restrict the number of pair comparison. Moreover, corresponding fields could be simply compared to compute the pairs similarity values. Because product offers have only two textual fields, Title and Description, the prior work mainly tried to extract attribute values from the offers text fields and subsequently compare them to their corresponding fields in the product data. In stead of that, here we follow a different approach as our baseline. Figure 2 depicts the architecture of the proposed approach. We use an inverted index for the product catalog, which substitutes the blocking key to narrow down the search space. We decided to use Elasticsearch[4] which is a disk based index, however, it is desirable to use an in-memory inverted index. Firstly, because of the growth in size of the product catalog (idealo's product catalog currently contains about 4 million entries out of that 1.8 million are active), and secondly, due to Elasticsearch's capability to deploy on a distributed platform, which satisfies our need of processing a huge volume of input offers.

Considering the full-text search feature of Elasticsearch, submitting any fuzzy query constructed from the offer title returns a long list of the similar products found inside the product catalog as a candidate set. It is a very time intensive task to process the entire candidate set using an expensive similarity measure. In order to further restrict this result set, we use some filter such as price range (we are preparing other

[4]Elasticsearch is an open source search engine based on lucene. It provides a distributed, multitenant-capable full-text search over the schema free JSON documents.

filters such as category to compensate the deficiencies of the price range). Because of the typographical errors and abbreviations in the product titles (as seen in the sample), we use the bigram-jaccard as the similarity measure over the titles of the product offers.

3.2 Hybrid approach

In the hybrid approach, we try to add the description field of the offers to our similarity measure. Prior work attempts to first parse the textual fields and extract attribute values out of it in order to construct a structured record, and thereafter compute the similarity between two structured records similar to the conventional de-duplication approaches [7, 17]. Even computing the similarity between two structured records also seems to be a challenging task due to the wide variety in attribute types [6, 9]. We believe the process of extracting attribute values is a complicated, time expensive, and error prone due to the large number of attributes (idealo's product catalog currently contain about 3.6 thousand unique attributes), and also synonyms in the attribute names. Therefore, instead of extracting attribute values, we purify the textual fields by removing irrelevant tokens (e.g. stop worlds, verbs). Thereafter, we directly compute the similarity between two records by finding the most similar tokens for each attribute from the relevant tokens in the textual fields. To perform this task we exploit an existing NLP API, then by adopting the weighting strategy proposed by Kannan et al., [13] to compute the final record similarity value between attribute values of the product and offer textual fields. Our preliminary evaluation over a subset of our offer evaluation dataset (containing 23 thousand offers from a wide variety of the product categories that are manually labeled) shows its effectiveness to map the corresponding attribute values and improves the f-measure of the baseline by almost 8 %. This is our ongoing work and it needs to be investigated more precisely.

4 Future Work

- As it is mentioned in the related work and observed in our experiments, application of hybrid similarity measures improve the accuracy of the matching result by increasing the runtime. It would be worthy to try to optimize runtime of the solution by utilizing further necessary steps, such as more efficient indices.
- Machine learning classifiers have proved to advance quality of the product matching. Investigating on the effects of using different classifiers such as SVM, logistic regression, decision tree and other classifiers on the idealo's dataset would be an appealing direction.
- Prior work has shown that domain or category specific similarity measures outperform the general one. Hence, automatic offer categorization facilitates

designing category specific similarity measures, which in turn improve product matching quality and also makes it more efficient. Furthermore, it could also prevent the cumbersome task of manual categorization or in case of an existing category list makes it possible to update it incrementally.

- Offer clustering is another very important direction in our research, because currently more than 90 % of the offers cannot be matched to any product in the catalog during the product matching. Considering the special characteristics of offer datasets we would like to investigate on different clustering approaches in order to devise a method to improve the accuracy and runtime of the offer clustering. The proposed method should be scalable so that it can process a large volume of data from idealo project. This might require an approach runnable in a distributed platform such as Apache Spark. Moreover, due to the streaming nature of the input data, the solution has to ingest data in real time.

References

[1] M. de Bakker, F. Frasincar, and D. Vandic. "A Hybrid Model Words-driven Approach for Web Product Duplicate Detection". In: *Proceedings of the International Conference on Advanced Information Systems Engineering*. CAiSE. 2013, pages 149–161. ISBN: 978-3-642-38709-8.

[2] K. Balog. "On the Investigation of Similarity Measures for Product Resolution". In: *Proceedings of the International Conference on Discovering Meaning On the Go in Large Heterogeneous Data*. LHD. 2011, pages 49–54.

[3] S. Bergamaschi, F. Guerra, and M. Vincini. "A Data Integration Framework for e-Commerce Product Classification". In: *Proceedings of the First International Semantic Web Conference on The Semantic Web*. ISWC. 2002, pages 379–393. ISBN: 978-3-540-48005-1.

[4] R. van Bezu, S. Borst, R. Rijkse, J. Verhagen, D. Vandic, and F. Frasincar. "Multi-component Similarity Method for Web Product Duplicate Detection". In: *Proceedings of the Annual ACM Symposium on Applied Computing*. SAC. 2015, pages 761–768. ISBN: 978-1-4503-3196-8.

[5] M. Bilenko, S. Basu, and M. Sahami. "Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Shopping". In: *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. 2005, pages 58–65. ISBN: 0-7695-2278-5.

[6] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. "Adaptive Name Matching in Information Integration". In: *IEEE Intelligent Systems* 18.5 (2003), pages 16–23. ISSN: 1541-1672.

[7] V. T. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania. "Efficiently Linking Text Documents with Relevant Structured Information". In: *Proceedings of the International Conference on Very Large Databases (VLDB)*. 2006, pages 667–678.

[8] P. Christen. "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication". In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 24.9 (2012), pages 1537–1555. ISSN: 1041-4347.

[9] M. Cochinwala, V. Kurien, G. Lalk, and D. Shasha. "Efficient Data Reconciliation". In: *Information Sciences* 137.1-4 (2001), pages 1–15. ISSN: 0020-0255.

[10] I. van Dam, G. van Ginkel, W. Kuipers, N. Nijenhuis, D. Vandic, and F. Frasincar. "Duplicate Detection in Web Shops Using LSH to Reduce the Number of Computations". In: *Proceedings of the Annual ACM Symposium on Applied Computing*. SAC. 2016, pages 772–779. ISBN: 978-1-4503-3739-7.

[11] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. "Duplicate Record Detection: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 19.1 (2007), pages 1–16. ISSN: 1041-4347.

[12] K. Hanna, T. Andreas, and R. Erhard. "Learning-Based Approaches for Matching Web Data Entities". In: *IEEE Internet Computing* 14 (2010), pages 23–31. ISSN: 1089-7801.

[13] A. Kannan, I. E. Givoni, R. Agrawal, and A. Fuxman. "Matching Unstructured Product Offers to Structured Product Specifications". In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2011, pages 404–412. ISBN: 978-1-4503-0813-7.

[14] Y.-g. Kim, T. Lee, J. Chun, and S.-g. Lee. "Modified NaïVe Bayes Classifier for e-Catalog Classification". In: *Proceedings of the Second International Conference on Data Engineering Issues in E-Commerce and Services*. DEECS. 2006, pages 246–257. ISBN: 978-3-540-35441-3.

[15] H. Köpcke, A. Thor, and E. Rahm. "Evaluation of Entity Resolution Approaches on Real-world Match Problems". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pages 484–493. ISSN: 2150-8097.

[16] H. Köpcke, A. Thor, S. Thomas, and E. Rahm. "Tailoring Entity Resolution for Matching Product Offers". In: *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2012, pages 545–550. ISBN: 978-1-4503-0790-1.

[17] M. Michelson and C. A. Knoblock. "Creating Relational Data from Unstructured and Ungrammatical Data Sources". In: *Journal of Artificial Intelligence Research* 31.1 (2008), pages 543–590. ISSN: 1076-9757.

[18] A. Thor. "Toward an adaptive String Similarity Measure for Matching Product Offers". In: *Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*. 2010, pages 702–710.

[19] D. Vandic, J.-W. Van Dam, and F. Frasincar. "Faceted Product Search Powered by the Semantic Web". In: *Decision Support Systems* 53.3 (2012), pages 425–437. ISSN: 0167-9236.

[20] *Worldwide Retail Ecommerce Sales Will Reach $1.9 Trillion This Year*. URL: https://www.emarketer.com/Article/Worldwide-Retail-Ecommerce-Sales-Will-Reach-1915-Trillion-This-Year/1014369 (last accessed 2016-10-01).

Video Captioning with Deep Neural Networks

Cheng Wang

Internet Technologies and Systems
Hasso-Plattner-Institut
Cheng.Wang@hpi.de

This report summaries my research activities of the past six months in the HPI Research School on Service Oriented Systems Engineer. In this report, we explore ways of applying Deep Learning techniques to generate captions for videos. We facilitate Long Short-Term Memory using the Caffe framework and a pre-trained VGG network for image feature extraction. We show how to get started with video captioning and discuss good practices to improve the performance of video captioning.

1 Introduction

Videos are an intuitive and easy way of recording real life sceneries. In the blink of an eye humans can recognize known objects, persons and movements in a video, not even considering recorded sound. Even children and animals without reading skills or knowledge of other information systems can comprehend the information displayed in a video. For computers, understanding the content of a video constitutes a major problem since the pixels do not refer to the semantics of objects. Only by applying recent advancements in the field of image recognition, we are able to recognize and understand roughly what a human sees in a picture.

Video to text translation tries to transfer that progress from single pictures to (short) videos. This enables information retrieval of a big chunk of "undiscovered" and uncategorized data, e.g. the millions of videos uploaded to YouTube, statistics[1] shows that 300 hours of video are uploaded to YouTube every minute. Unless the authors of those videos provide detailed text information on the contents of the video, we are unable to index, sort or cluster the videos by its actual contents. By automatically generating captions and tags for videos, this no longer holds true. Furthermore, this enables improvements in the domain of recommender systems for video data. It can also not just improve the smartness of backends, but increase web accessibility for platforms like YouTube by providing textual description of videos for handicapped users. In this work, we aim to understand, implement and enhance the full process of video to text translation. We want to discover how state of the art video recognition works in detail, so we could also identify space for improvement.

Our work is based on the research of Subhashini Venugopalan and Jeff Donahue et al. [10], in which they propose an end-to-end sequence-to-sequence model to tackle the challenge of both variable length input and output. To achieve this, they

[1] https://www.youtube.com/yt/press/en-GB/statistics.html (last accessed 2016-10-20).

use recurrent neural networks, specially Long short-term memory (LSTMs). Jeff Donahue himself further elaborated on this topic is his paper "Long-term Recurrent Convolutional Networks for Visual Recognition and Description" [3], which we also highly valued in our work. Our dataset choices are based on "MSR-VTT: A Large Video Description Dataset for Bridging Video and Language" from Jun Xu et al. [11], which also was used for a Microsoft Research Challenge on video-to-sentence transformation.

2 Methodology

Jeff Donahue et al. [4] showed, that temporal deep learning for video to text conversion can outperform approaches, that neglect the order of frames, which is why we wanted to learn with temporal context. Recurrent Neural Networks (RNNs) in contrast to the feed-forward approach process sequences of inputs in such a way, that the output of the last time step can be used as input for the next one. This implements a kind of memory or hidden state and makes it possible to learn temporal information out of sequences of inputs. Conventional RNNs suffer from gradient vanishing and explosion on long term sequences, which suggests the use of long short-term memory (LSTMs) a special kind of RNN that can learn to forget information and thus are able to cope with longer sequences [4]. Therefore we use the LSTM-based model denition from Venugopalan et al. [10] available on GitHub, which is trained and used with the Caffe deep learning framework [5]. This results in the architecture displayed in Figure 1. The part (red) consists of a convolutional neural network (CNN). It takes as an input a frame from the video.

3 Training and Optimization

Based on our proposed network structure, this section describes the training and optimization of network, we also discuss some influential factors of improving network performance.

Frame rate LSTM is able to learn from sequences (i.e. multiple frame) and they are quite flexible about how many steps to learn from. This means, we can input 5 or 80 frames. During training, however, we need to have a fixed amount of steps (i.e. frames), because we want to propagate and back-propagate in batches with the averaged loss and delta. We cannot do this, if the different propagations in one batch are differently enrolled. Furthermore, the more frames we feed in, the more memory is required to keep all values for back-propagation. Sampling with a specific frame rate leads to sequence lengths as different as the video lengths. Sampling about the same amount of frames leads, in turn, to varying time intervals between frames. Our dataset contains only short video sequences, thus for training we stick to using a maximum of 80 frames at 5 frames per second with padding if necessary. In this

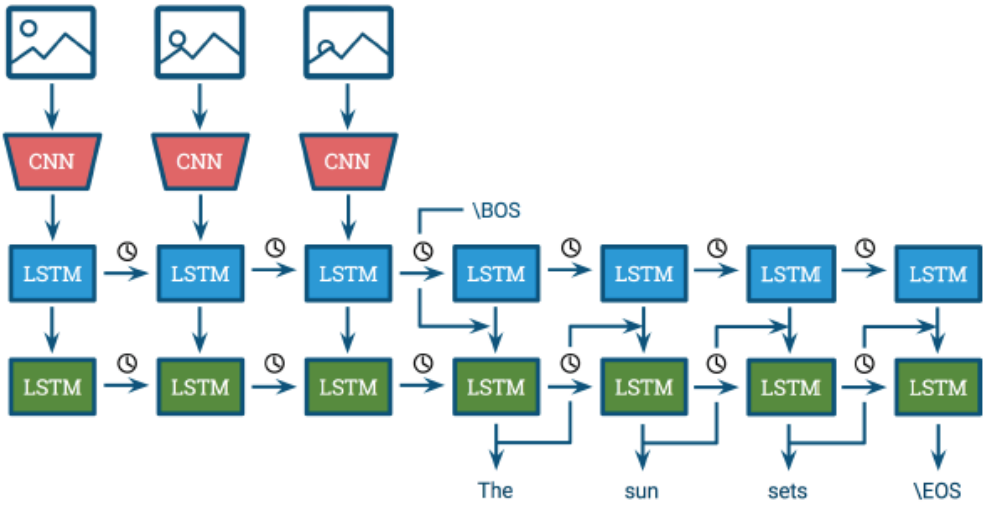


Figure 1: Architecture for video captioning. Each extracted video frame is fed to CNN that used as image encoder, the output of CNN will be fed to LSTM which encodes sentence input as well as decodes image and sentence to generate novel sentence.

setting-combined with the lengths of the videos the frame rate should not be higher than 5 frames per second to not truncate too much. Optimizing the frame rate is about finding a good trade-off between resulting sequence length and keeping a low delta between frames.

Mean pooling Averaging the values over n frame feature vectors is called mean pooling in contrast to the more naive approach of only taking the feature vector of every n-th frame for training. Mean pooling therefore helps by smoothing the data and thus, reducing the effect of outliers. Another approach to solve this is called soft attention which selectively focuses on only a small subset of frames. This may lead to small improvements over mean pooling.

Size of dataset As with many tasks in computer science, the size of the data that is used has a big influence on both performance and quality of the results. On the one hand, using a bigger dataset improves the output of our network. On the other hand, it would increase the time for training. Moreover, the results could not be tested as early thus leaving less time to evaluate different settings of other parameters.

Iteration count and batch size The batch size controls how many feature vectors calculated from images are processed in each iteration of training our LSTM in Caffe. The iteration count specifies how much iterations are used for training. Those two parameters are tightly coupled as by multiplying them we obtain the number of times a single feature vector is processed. As we show in following section they have an influence on the quality of the results.

Learning rate The learning rate influences to which extent the network's parameters are adapted after each iteration. It is one of the most important options that can be tweaked for different training results. Having an optimized learning rate schedule is important for faster convergence and the final accuracy.

Flow images Optical flow images are pictures which utilize the colors of their pixels for capturing the difference between key frames. As they convey motion information between frames they might improve the results of creating captions for videos depending on the kind of activity that is displayed. They are typically used by doing two passes through the network, one with the normal image data and one with the flow images, followed by averaging the results.

4 Evaluation

We use the Microsoft COCO Evaluation Server [1] to compare each generated sentence to 20 ground truth sentences. Those ground truth sentences were compiled by Amazon Mechanical Turks, so they represent the way a human would describe the videos.

The COCO evaluation server takes candidate captions and scores them using several popular metrics, including BLEU [8], METEOR [2], ROUGE [6] and CIDEr [9]. The evaluation server was originally designed to compare *image* captions, but as they do not depend on the images at all and rather work as a similarity metric between the sentences, we are reusing COCO to compare the generated *video* sentences from our model to the ground truths provided by Microsoft.

Furthermore the same metrics are used in the MS Multimedia Challenge [7] from which we used the data. Unfortunately we could not access all data because of the missing test sentences and deleted/blocked videos on YouTube. This makes our evaluation results less expressive compared to the other attendees of the MS Multimedia Challenge. Nevertheless we made sure to have the same splits between train, test and val data as the challenge does.

4.1 BLEU

Bilingual evaluation understudy (BLEU) is a popular automatic machine translation evaluation metric known for its high correlation with human judgement. It uses n-grams, i.e. a contiguous sequence of n items, to calculate the similarity between two sentences. BLEU is known to prefer brevity and to perform better on a corpus level by averaging over sentences. Bigger n lead to fewer matches when comparing sentences and therefore the results get worse. This is why BLEU is not *perfectly* suited for our standalone sentences and we use other metrics as well. For its multimedia challenge, Microsoft chose to only look at 4-gram BLEU scores [8].

4.2 METEOR

Metric for Evaluation of Translation with Explicit ORdering (METEOR) tries to solve problems in the BLEU metric by putting more emphasis on order and recall. This is achieved by hierarchically resolving the identity of words in two sentences by the following matchers: exact, stem, synonym and paraphrase. Moreover, METEOR prefers similarly aligned matches between compared sentences [2].

4.3 ROUGE

Recall-Oriented Understudy for Gisting Evaluation (ROGUE) is a collection of multiple evaluation metrics for machine translation specifically focused on summarization. It consists of ROUGE-N (n-gram co-occurrence Statistics), ROUGE-W (weighted longest common subsequence), ROUGE-S (skip-bigram co-occurrence statistics) and ROUGE-L (longest common subsequence).

ROUGE-L is the only one interesting for us as it is used in the Microsoft Multimedia Challenge. Unlike n-grams, it allows other words to appear among the ones that constitute the sequence. The final score is calculated using the harmonic mean of recall and precision also known as F measure. For ROUGE-L, it is common practice to use an F measure that slightly favors recall over precision [1, 6].

4.4 CIDEr

Consensus-based Image Description Evaluation (CIDEr) works by weighting the n-grams found across all sentences using term frequency–inverse document frequency (tf-idf). This gives higher weights to n-grams occurring rarely thus filtering out common phrases while promoting salient ones. The resulting CIDEr score is obtained by calculating the average cosine similarity over the vector of tf-idf values of all n-grams in the compared sentences. In the Microsoft Multimedia Challenge the extension CIDEr-D is used which tries to mitigate gaming effects. Gaming refers to the phenomenon that a metric may score sentences highly that are judged as rather dissimilar by humans. CIDEr-D prevents this by adding clipping as well as a length based Gaussian penalty [9].

5 Results

In this section, we want to show training results and discuss what we could learn from them. The results contain statistics over the training process and quality indicators of the model's performance. Overall, the model can only get as good as the data that is used for training. YouTube videos, even though they do not have much context knowledge, are not necessarily a good source for general purpose video captioning. They either often show the same specific situations, like women or men explaining something in a *vlog*; or they contain multiple scenes, which are hard to summarize

Table 1: Evaluation results for different reduction techniques and batch sizes over training iterations

Reduction	batch	It.	CIDEr	$ROUGE_L$	METEOR	$Bleu_4$	$Bleu_3$	$Bleu_2$	$Bleu_1$
Keyframe	5	30k	0.285	0.538	0.237	0.288	0.402	0.538	0.706
		60k	0.309	0.550	0.244	0.308	0.427	0.570	0.738
		90k	0.300	0.545	0.239	0.299	0.417	0.560	0.732
	16	10k	0.312	0.562	0.239	0.319	0.438	0.573	0.732
		20k	0.319	0.556	0.244	0.318	0.439	0.582	0.745
		30k	0.313	0.549	0.244	0.306	0.427	0.571	0.740
Mean Pooling	16	10k	0.302	0.558	0.240	0.313	0.435	0.576	0.737
		20k	0.319	0.556	0.244	0.318	0.439	0.582	0.746
		30k	0.317	0.552	0.244	0.307	0.431	0.578	0.746

in one caption. Imagine for the latter, a video about a dancing group of people, cut together with interviews about the dance. The ground truth may be *some people are dancing* or *a group is dancing*, but this only reflects a specific scene. During training, the model's knowledge about interview scenes becomes distorted.

Nonetheless, for the given data, we want to maximize the overall quality of the model's performance. According to section 3, we tried to focus on understanding the influence of the batch size, the iterations and the reduction strategy (which is taking key frames or to mean pool frame features). We need to keep in mind, that with around $6000 * 65\,\% = 3900$ training videos and 20 captions per video, we have 78000 training records, which leads to exhaustion of all training data once every $\frac{78000}{|\text{batch}|}$ iterations. For batch size 5, that is 15600 and for batch size 16, that is 4875. After 90000 iterations for batch size 5 and 30000 iterations for batch size 16 accordingly, we have learned every training data record about 6 times. The graphs in the Figures 2, 3 and 4 show a similar loss development. The variance for the lower batch size of 5 is higher as expected.

After having used the data about 4 times, overfitting seems to take more and more effect. The difference of the train and test accuracy grows (see Figures 2, 3 and 4 on the right) and evaluation scores decreases (see Table 1). For our specific setting, we assume a batch size of 16 trained with around 20000 iterations to yield good results. However, we cannot confirm the significant improvement of using mean pooling reduction that Jun Xu et al. [11] showed. We assume, that our frame rate for the key frames is too high, so that pooling over multiple frames does not add more information to the feature vector and only reduces noise and variance slightly.

In one experiment, we trained on key frames with 20000 iterations and a batch size of 16. But instead of generating captions from key frame features, we generated them from pooled frame features and evaluated the captions. The results are overall higher scores for each metric, with *CIDEr* improving the most from originally 0.319 to 0.324.

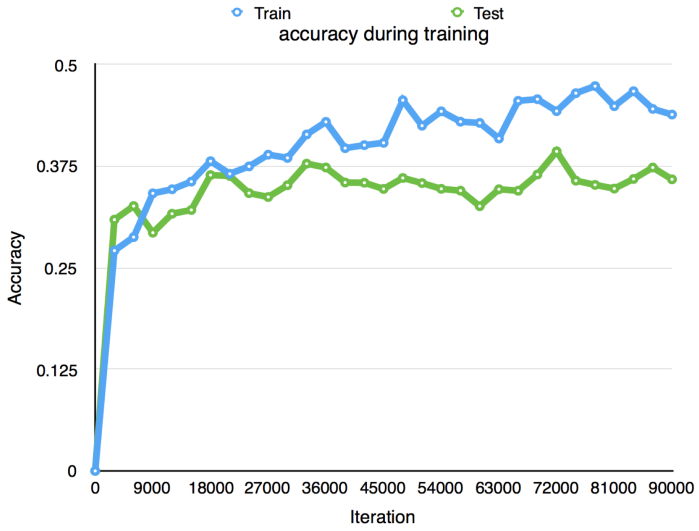


Figure 2: Keyframes, batch size 5, 90k iterations: change of training loss, learning rate and accuracy during training. One iteration means one change of the weight based on the averaged loss of the batch. n being the amount of training records, all records have been used after $\frac{n}{|\text{batch}|}$ iterations during training. To enable comparison of the different batch sizes, 90000 iterations are needed to equal the 16 batch size trainings.

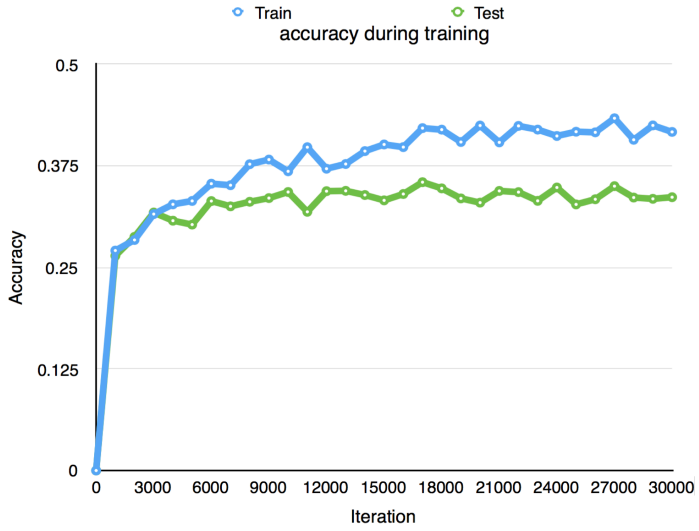


Figure 3: Keyframes, batch size 16, 30k iterations: change of training loss, learning rate and accuracy during training.

Figure 4: Mean pooling, batch size 16, 30k iterations: change of training loss, learning rate and accuracy during training.

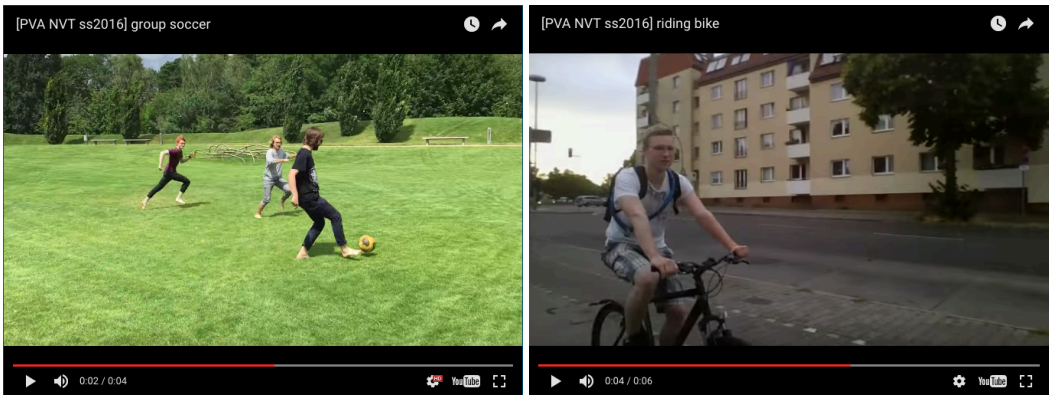


Figure 5: Successful examples of video captioning, left: "A group of kids playing with a ball in a field.", right: "A young man is riding a bike and talking".

6 Conclusion and outlook

In this report, we proposed a framework for generating video descriptions for given video input. By feeding the feature vectors that extracted from CNN model to LSTM network, our model is able to capture the semantic correlation between video frames and word sequence. Novel sentence descriptions can be generated with pre-trained model. Regarding future work, in our work the dataset only contained short video clips with one scene each. Longer videos more likely contain multiple scenes with different topics. One future improvement would be to detect those topical boundaries and process the resulting clip parts separately.

References

[1] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick. "Microsoft COCO captions: Data collection and evaluation server". In: *arXiv preprint arXiv:1504.00325* (2015).

[2] M. Denkowski and A. Lavie. "Meteor Universal: Language Specific Translation Evaluation for Any Target Language". In: *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*. 2014.

[3] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description". In: *CVPR*. 2015.

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. "Long-term recurrent convolutional networks for visual recognition and description". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pages 2625–2634.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pages 675–678.

[6] C.-Y. Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out: Proceedings of the ACL-04 workshop*. Volume 8. Barcelona, Spain. 2004.

[7] T. Mei and T. Yao. *Microsoft Multimedia Challenge*. [Online; accessed 5-August-2016]. 2016. URL: http://ms-multimedia-challenge.com/ (last accessed 2016-10-01).

[8] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pages 311–318.

[9] R. Vedantam, C. Lawrence Zitnick, and D. Parikh. "Cider: Consensus-based image description evaluation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pages 4566–4575.

[10] S. Venugopalan, M. Rohrbach, J. Donahue, R. J. Mooney, T. Darrell, and K. Saenko. "Sequence to Sequence – Video to Text". In: *CoRR* abs/1505.00487 (2015).

[11] J. Xu, T. Mei, T. Yao, and Y. Rui. "MSR-VTT: A Large Video Description Dataset for Bridging Video and Language". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
110	978-3-86956-387-9	**Transmorphic : mapping direct manipulation to source code transformations**	Robin Schreiber, Robert Krahn, Daniel H. H. Ingalls, Robert Hirschfeld
109	978-3-86956-386-2	**Software-Fehlerinjektion**	Lena Feinbube, Daniel Richter, Sebastian Gerstenberg, Patrick Siegler, Angelo Haller, Andreas Polze
108	978-3-86956-377-0	**Improving Hosted Continuous Integration Services**	Christopher Weyand, Jonas Chromik, Lennard Wolf, Steffen Kötte, Konstantin Haase, Tim Felgentreff, Jens Lincke, Robert Hirschfeld
107	978-3-86956-373-2	**Extending a dynamic programming language and runtime environment with access control**	Philipp Tessenow, Tim Felgentreff, Gilad Bracha, Robert Hirschfeld
106	978-3-86956-372-5	**On the Operationalization of Graph Queries with Generalized Discrimination Networks**	Thomas Beyhl, Dominique Blouin, Holger Giese, Leen Lambers
105	978-3-86956-360-2	**Proceedings of the Third HPI Cloud Symposium "Operating the Cloud" 2015**	Estee van der Walt, Jan Lindemann, Max Plauth, David Bartok (Hrsg.)
104	978-3-86956-355-8	**Tracing Algorithmic Primitives in RSqueak/VM**	Lars Wassermann, Tim Felgentreff, Tobias Pape, Carl Friedrich Bolz, Robert Hirschfeld
103	978-3-86956-348-0	**Babelsberg/RML : executable semantics and language testing with RML**	Tim Felgentreff, Robert Hirschfeld, Todd Millstein, Alan Borning
102	978-3-86956-347-3	**Proceedings of the Master Seminar on Event Processing Systems for Business Process Management Systems**	Anne Baumgraß, Andreas Meyer, Mathias Weske (Hrsg.)
101	978-3-86956-346-6	**Exploratory Authoring of Interactive Content in a Live Environment**	Philipp Otto, Jaqueline Pollak, Daniel Werner, Felix Wolff, Bastian Steinert, Lauritz Thamsen, Macel Taeumel, Jens Lincke, Robert Krahn, Daniel H. H. Ingalls, Robert Hirschfeld

ISBN 978-3-86956-390-9
ISSN 1613-5652