DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Test Case Prioritization as a Mathematical Scheduling Problem

MARCUS AHLBERG

ERIC FORNANDER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Test Case Prioritization as a Mathematical Scheduling Problem

MARCUS AHLBERG
ERIC FORNANDER

Degree Projects in Optimization and Systems Theory (30 ECTS credits)
Degree Programme in Industrial Engineering and Management (120 credits)
KTH Royal Institute of Technology year 2018
Supervisor at Rise Sics: Sahar Tahvili
Supervisor at KTH: Per Enqvist
Examiner at KTH: Per Enqvist

TRITA-SCI-GRU 2018:251
MAT-E 2018:51

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

Software testing is an extremely important phase of product development where the objective is to detect hidden bugs. The usually high complexity of today's products makes the testing very resource intensive since numerous test cases have to be generated in order to detect all potential faults. Therefore, improved strategies of the testing process is of high interest for many companies. One area where there exists potential for improvement is the order by which test cases are executed to detect faults as quickly as possible, which in research is known as the test case prioritization problem. In this thesis, an extension to this problem is studied where dependencies between test cases are present and the processing times of the test cases are known. As a first result of the thesis, a mathematical model of the test case prioritization problem with dependencies and known processing times as a mathematical scheduling problem is presented. Three different solution algorithms to this problem are subsequently evaluated: A Sidney decomposition algorithm, an own-designed heuristic algorithm and an algorithm based on Smith's rule. The Sidney decomposition algorithm outperformed the others in terms of execution time of the algorithm and objective value of the generated schedule. The evaluation was conducted by simulation with artificial test suites and via a case study in industry through a company in the railway domain.

Sammanfattning

Mjukvarutestning är en extremt viktigt fas i produktutveckling då det säkerställer att inga buggar finns i mjukvaran. Då nutidens produkter ofta inkluderar en komplex mjukvara, kräver mjukvarutestningen mer resurser än tidigare. Eftersom komplexiteten kräver att fler testfall för mjukvaran definieras för att upptäcka eventuella buggar. Detta har skapat ett stort intresse hos företag för strategier inom delområden av mjukvarutestning som syftar till att effektivisera och förenkla desamma. Ett av dessa uppmärksammade delområden är i vilken ordning testfallen ska utföras i syfte att upptäcka buggar i ett så tidigt skede som möjligt, vilket i litteraturen är känt som prioriteringsproblemet för testfall. I den här uppsatsen studeras en utökad version av prioriteringsproblemet där det existerar företrädesberoenden mellan testfallen samt att tiden det tar att exekvera ett testfall är känd. Som ett första delresultat presenteras en matematisk modell av detta utökade problem i form av ett matematiskt schemaläggningsproblem. Sedermera jämförs tre lösningsmetoder för denna modell. Lösningsmetoderna som jämförs är Sidneys upplösningsmetod, en egendesignad metod samt en metod baserad på Smiths regel. Sidneys upplösningsmetod var den metod som gav bäst resultat avseende både exekveringstid och numeriskt resultat. Jämförelsen genomfördes genom simulering av flera artificiellt skapade testfall samt genom en fallstudie på ett företag i järnvägsindustrin.

ACKNOWLEDGEMENTS

This thesis has been enabled through support of ECSEL and VINNOVA (through the projects MegaM@RT2 and TESTOMAT).

We would like to thank Ola Sellin, Mahdi Sarabi and Johanna Norkvist at Bombardier Transportation in Västerås for providing valuable insights within the subject.

A special gratitude goes out to our supervisor, Sahar Tahvili, for her extra ordinary engagement in our work and knowledge within the field.

We are also grateful to our supervisor at KTH, Per Enqvist, who has given us helpful feedback throughout the work with this thesis.

Finally, we want to thank Simon Park and Jimmy Lilja for their support and interesting discussions we have had during the work with the report.

Contents

Table 1: Table of Notation

C_j	$\triangleq$	The completion time of job J_j in a schedule S
D_j	$\triangleq$	The immediate dependencies of job J_j
E	$\triangleq$	A set of edges in a directed acyclic graph G
G	$\triangleq$	A directed acyclic graph
J_j	$\triangleq$	A job
K	$\triangleq$	A sufficiently large number
L_j	$\triangleq$	The lateness of job J_j
M	$\triangleq$	The total number of machines
S	$\triangleq$	An ordered set of jobs, which makes up a schedule
T_j	$\triangleq$	The tardiness of job J_j
U_j	$\triangleq$	The unit penalty of job J_j
V	$\triangleq$	The vertices in a directed acyclic graph G
Z_j	$\triangleq$	The immediate successors of job J_j
d_j	$\triangleq$	The due date of job J_j
n	$\triangleq$	The total number of jobs
p_j	$\triangleq$	The processing time of job J_j
r_j	$\triangleq$	The release date of job J_j
$x_{i,j}$	$\triangleq$	A binary variable telling if job J_i is processed before job J_j in a schedule S
w_j	$\triangleq$	The weight of job J_j
λ	$\triangleq$	A non-negative real valued parameter
ρ	$\triangleq$	A real valued parameter

Chapter 1

Introduction

Virtually every product produced by today's technological companies includes some software. A systematic approach to develop a software product or a product including any software is The Software Development Life Cycle (SDLC). It includes six phases shown in Figure 1.1.

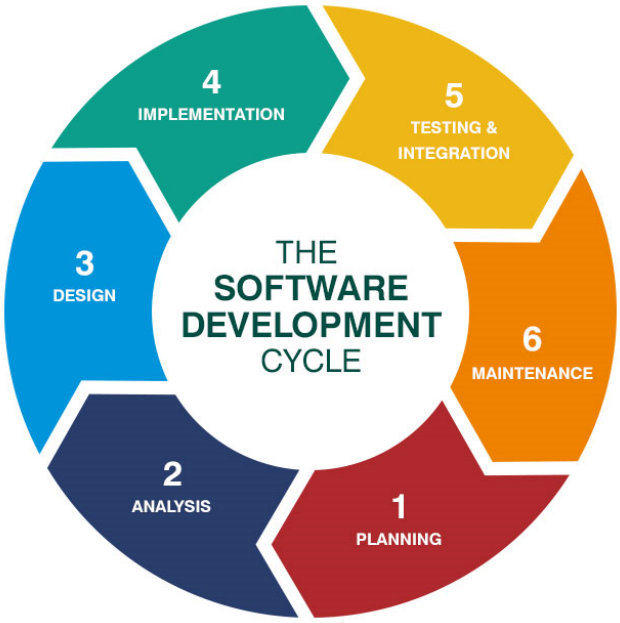

Figure 1.1: The Software Development Life Cycle

1.1 Software Testing

The software testing phase (phase 5 in Figure 1.1) plays a vital role in the software development life cycle and should be carried out effectively to guarantee and improve the quality of a product.

Definition 1. *Software testing is a process of executing a program with the aim of finding hidden bugs in a software product. There are four levels of software testing: unit testing, integration testing, system testing and acceptance testing.*

In order to test a product, a set of scenarios needs to be defined, which are called test cases.

Definition 2. *A test case is a set of conditions under which a tester will determine whether an application, software system or one of its features is working as it was originally established to do.*

1.1.1 Dependencies Between Test Cases

Test cases are constructed to test one or several requirements of an application or a product. Since some requirements are more suitable to be tested prior to others and in some cases also have to be tested before others, some restraints on the chronological order by which the test cases can be executed exist. Intuitively, one can think of a case where the turn on/off button on a cellphone is tested. Having tested if the battery works before executing that test is essential to obtain any useful result from the latter. Such relationships between test cases can be found by source code analysis, as example. These kind of relations are hereinafter referred to as dependencies between test cases.

1.2 Case Study

An industrial case study has been conducted at Bombardier Transportation (BT), which is a multinational and a high technological company.

Figure 1.2 shows a graphical sample of a systems development life-cycle, called the V-model. The part of the life-cycle analyzed at BT is the integration testing.

Definition 3. *Integration testing is a phase in the software testing, where individual software parts are combined and tested as a group.*

In integration testing the interactions between various parts of a software product are tested. This phase is particularly interesting to analyze further since it is a phase where a lot of todays methods can be developed with potentially innovative results. The reason for this is since the other parts of the testing procedure are more of formalities and thus it exists less room for changes in those phases.

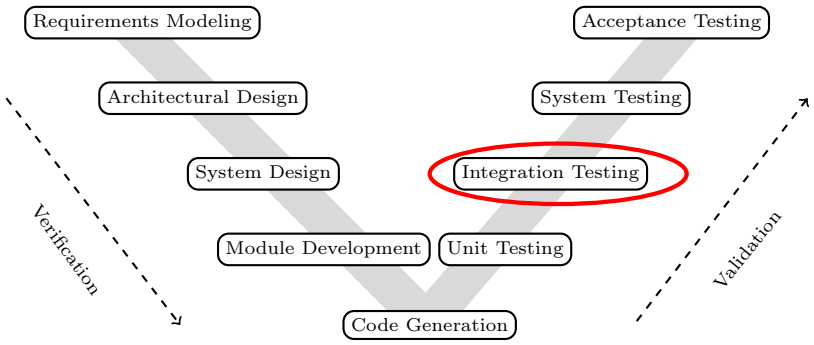

Figure 1.2: The V Model for the software development life cycle.

From the documentation of test cases in a project at BT, the existing dependencies between test cases was able to be identified (Tahvili et al., 2018). In addition, processing times of the test cases are known from an earlier project (Ameerjan, 2017) which enables deeper analysis of the testing process. Generally in comparable companies to BT, there are numerous test cases within a project. This is also true for the project considered in the case study in this thesis.

1.3 Problematization

The number of test cases which are required to test a product depends on several factors such as the size and the complexity of the software included in the product. Usually, a large number of test cases are generated by testers for testing a product, which in turn consume a large amount of testing resources when executed. Therefore, some concepts such as test case selection and test case prioritization have been considered as hot research areas in the past decade.

Test case prioritization deals with sequencing the order of the test cases for execution. Yoo and Harman (2012) defined the prioritization problem as follows:

Definition 4. *Given a test suite, T, the set of permutations of T, PT, and a function from PT to real numbers, $f : PT \rightarrow \mathbb{R}$. Find $T' \in PT$ such that $f(T^{'}) \geq f(T^{''}) \; \forall \; T^{''} \in PT$.*

There exists no solid theoretical solution to the prioritization problem nor any practical at BT. Hence, the scope of this thesis is to investigate and solve the prioritization problem in similar settings as BT. A solution to this problem is of high relevance and practically useful for BT and similar companies due to the time consuming test process they have today, which then can be limited. Solving the test case prioritization problem would also generate several other benefits for industry. For instance, goals such as earlier fault detection and

reducing the invested testing effort can be achieved.

A common primary objective, f, of the testing process is to maximize the Average Percentage of Faults Detected (APFD) metric. This metric is defined as the area under a graph where the percentage of faults detected is on the y-axis and the percentage of the test suite tested on the x-axis (Catal, 2012). However, in reality one does not know which test cases that will detect faults nor the total number of faults which exists in the software in advance. Therefore, other measures which are correlated with the probability for a test case to detect a fault or the impact of a possible fault which a test case can discover are used instead. For example, test cases can be prioritized by the customers opinions, viz., what the customers believe are important, or by the development complexity. These are two examples of measures which Khandelwal and Bhadauria (2013) discuss. Another commonly used measure is to count the number of requirements covered by each test case, so called requirement coverage. Test cases with high requirement coverage have in theory higher probability of detecting faults since they test more requirements than test cases with low requirement coverage. Hence, it can be beneficial to prioritize by requirement coverage since it will increase the probability of detecting faults early in the process.

Summarized, a common test objective is to choose a measure and then maximize the sum of values of that measure per time unit. If no processing times for the test cases are known, an arbitrary value can be used as processing time for all test cases. In such case, the objective is the same as maximizing the sum of values of the measure per test case executed. However, if the measure is defined as a fixed value per test case (which is common), the problem can be formulated similar to the APFD metric, namely maximizing an area under a graph. The graph in this case is created by having the total sum of the values of the measure which has been tested on the y-axis and time on the x-axis. Such a problem is visualized in Figure 1.3, where a good schedule (green color with large area) is compared with another schedule (red color with smaller area). At any point in time, the good schedule has higher or equal sum of the values of the given measure, which makes it a better schedule.

Moreover, dependencies between test cases play an important role in the prioritization problem. Researchers have shown that ignoring dependencies when prioritizing test cases can lead to sequential errors (Zimmerman et al., 2011; Tahvili et al., 2016c). Prioritizing with respect to dependencies will result in a schedule where any fault detected will be on the lowest level possible, which makes the debug-process more convenient and hence, a more favorable schedule is obtained.

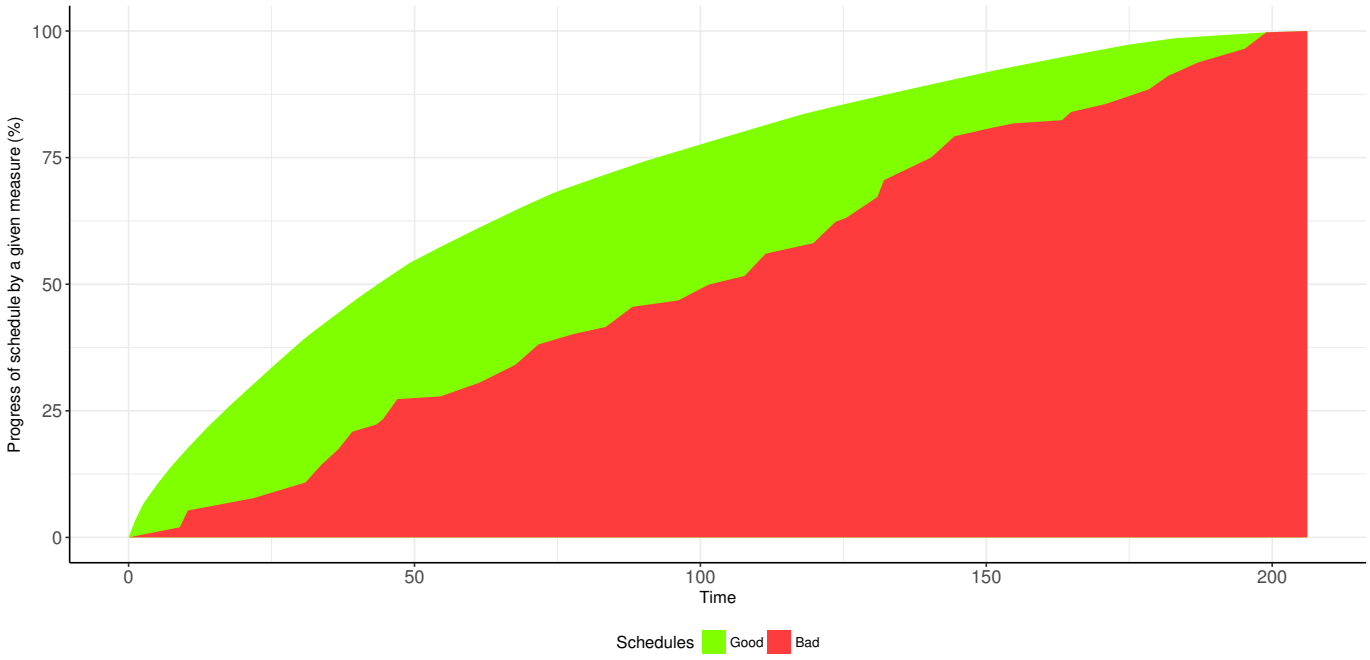

Figure 1.3: A good and a bad schedule given a certain measure

1.4 Research Questions

Based on the problematization and background information presented, the first research question which this thesis strives to answer is:

RQ1. *How can the prioritization problem be modeled when a test suite, T, consists of test cases with known time and the objective, f, is the area under the graph created by the total sum of the values of a measure tested over time for a schedule S?*

From this, a follow-up question naturally arises, which then becomes the second research question for this thesis, namely:

RQ1.1 *How can such a problem be solved?*

1.4.1 Delimitations

A schedule which neglects all information about dependencies between test cases could theoretically be better, i.e. have larger area covered by the graph in Figure 1.3 compared to a schedule which has all test cases in a feasible order

according to the dependencies. However, such a schedule would not be a good schedule in practice, since any faults detected can potentially be derived to faults on a lower level which would have been tested already by a dependent test case (Zimmerman et al., 2011; Tahvili et al., 2016c). Hence, the prioritization problem will be modeled where the dependencies between test cases must not be violated.

Chapter 2

Related Work

In this chapter previous attempts found in research to solve the test case prioritization problem are presented. There are numerous researchers which have published relevant work within this area.

2.1 The Test Case Prioritization Problem

Elbaum et al. (2002) present and compare several different methods for solving the problem. All methods assume that all test cases take the same amount of time to process and that there are no dependencies between them. Similar methods are also presented by Rothermel et al. (2001).

Another method which considers multiple measures is presented by Tahvili et al. (2016a). Although the processing time of the test cases are discussed as a good measure, no dependencies between test cases are considered.

A method which considers dependencies is presented by Ma and Zhao (2008). However, the dependencies considered are between software modules and not test cases. The method first assigns an importance to each software module in order to calculate a value for each test case by the importance of the modules which the test case partly tests. The prioritization of the test cases is then carried out by ordering the test cases in a non-increasing order according to the value.

However, there are also other researchers that have considered dependencies between test cases. Caliebe et al. (2012) presents a method for shrinking the test suite when a new version of the software is released. This by combining the dependencies with the changes between the versions in order to identify software components which theoretically are unaffected by the new version. If such a module is identified, the test cases for that module can be removed from

the test suite. Methods based on this logic has also been presented by Bates and Horwitz (1993); Rothermel and Harrold (1994). However, the prioritization problem on the contracted test suite are not solved by any of these methods.

Another method which also considers dependencies between test cases but this time no processing times is presented by Acharya et al. (2010). For each test case, the numbers of inter and intra component object interactions, which are obtained from traversing the dependency graph of the software, a value for each test case is calculated. The test cases are then prioritized in non-increasing order of this value. It is argued that this is a good solution to the prioritization problem since a higher value indicates a higher dependency rate and the test case should therefore be highly prioritized in order to detect faults in the system quickly. The idea of assigning a value based on the test case's position in the dependency structure is also used in a another method presented by Haidry and Miller (2013).

Summarized, the related work are in line with the discussions in Chapter 1 since most methods presented by researchers are based on assigning a value on a test case basis by a certain measure.

Chapter 3

Theory

In this chapter relevant theory applied throughout the upcoming sections in the thesis is presented. Some relevant proofs are deducted for what is believed to be the most important theorems. The proofs are included since they provide the reader with profound understanding of how the algorithms work and not only as proofs of the specific theorems. In addition, some proofs are important to grasp the idea of optimality of the algorithms and are therefore written out in this section as well. However, the reader has the possibility of skipping the parts of the theory he or she is already familiar with.

3.1 Scheduling Problems on the form $\alpha \mid \beta \mid \gamma$

In order to describe a large variety of scheduling problems mathematically the notation $\alpha \mid \beta \mid \gamma$, introduced by Graham et al. (1979), can be used. The α, β and γ fields take different values depending on the specific characteristics of the problem. The common denominator between problems expressed on this form is that there are n jobs J_j where $j = 1, 2, ..., n$, that are to be scheduled on M machines where each machine can only process one job at a time. Moreover, certain job data are associated with each job J_j; a processing time p_j expressing how long the job has to spend on the various machines, a release date r_j, on which J_j becomes available for processing, a due date d_j, by which the job should ideally have been processed, a weight w_j, representing the relative importance of J_j and a non-decreasing real cost function $f_j(t)$ for completing job J_j at time t.

As mentioned earlier, the α, β and γ field describe the different characteristics of the problem. Firstly, the α field describes the machine environment of the problem. For example, if the problem consists of a single machine processing

the jobs, the representation of α would be "$\circ$". In addition, for two parallel and identical machines, α would instead be represented by "$P2$".

Secondly, the field $\beta = \{\beta_1, \beta_2, ..., \beta_6\}$ describes the characteristics of the jobs. Each β_i, $i \in \{1, 2, ..., 6\}$, stands for a setting which has a certain behavior in the specific problem. For example, $\beta_1 \in \{\text{pmtn}, \circ\}$ describes if preemption (ability to stop a job and later resume it) is allowed in the problem. If β_1 is "pmtn" then preemption is allowed otherwise β_1 is "$\circ$" which means it is forbidden.

The last field γ describes the optimality criteria. Given a schedule it is possible to calculate the following measures for each job J_j:

- Completion time C_j
- Lateness $L_j = C_j - d_j$
- Tardiness $T_j = \max\{0, L_j\}$
- Unit penalty $U_j = \begin{cases} 0, & \text{if } C_j \leq d_j \\ 1, & \text{otherwise} \end{cases}$

The optimal criteria field γ is chosen from $\{f_{\max}, \sum f_j\}$ where:

- $f_{\max} \in \{C_{\max}, L_{\max}\} = \max_j\{f_j(C_j)\}$ with $f_j(C_j) = C_j$ or $f_j(C_j) = L_j$ respectively
- $\sum f_j = \sum_{j=1}^{n} f_j(C_j)$ with $f_j(C_j) \in \{C_j, T_j, U_j, w_jC_j, w_jT_j, w_jU_j\}$

(Graham et al., 1979).

The objective value of a schedule S is denoted $\gamma(S)$ and consequently the value of an optimal schedule S^* is denoted $\gamma(S^*) \triangleq \gamma^*$.

3.1.1 $\circ \mid \{\circ, \circ, \circ, \circ, \circ, \circ\} \mid \sum w_jC_j$

The $\circ \mid \{\circ, \circ, \circ, \circ, \circ, \circ\} \mid \sum w_jC_j$-problem (commonly referred to as $1 \mid \mid \sum w_jC_j$) is the problem where n jobs are to be scheduled on one machine which can only process one job at a time. Additionally, preemption is forbidden and there are no precedence constraints between the jobs. Each job J_j takes p_j time units to process on the machine. The objective value of a given schedule is given by the sum of the products of the completion time C_j and a weight w_j over all jobs J_j (Graham et al., 1979).

This problem is clearly an optimization problem. If one is familiar with these kinds of problems, one can easily see that this specific problem can be formulated as a mixed integer program (MIP) (Keha et al., 2009). Such a formulation is presented below.

To begin with, define a binary variable

$$x_{i,j} = \begin{cases} 1, & \text{if job } J_i \text{ is processed before job } J_j \\ 0, & \text{otherwise} \end{cases}$$

The problem can now together with this binary variable be expressed as:

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} w_j C_j & & \\
\text{subject to} \quad & C_j \geq p_j & & j = 1, 2, ..., n \\
& C_i + p_j \leq C_i + K(1 - x_{i,j}) & & i = 1, 2, ..., n, j = 1, 2, ..., n \mid i < j \\
& C_j + p_i \leq C_j + K x_{i,j} & & i = 1, 2, ..., n, j = 1, 2, ..., n \mid i < j \\
& C_j \geq 0 & & j = 1, 2, ..., n \\
& x_{i,j} \in \{0, 1\} & & i = 1, 2, ..., n, j = 1, 2, ..., n
\end{aligned}$$

where K is a sufficient large number, generally to be chosen as the sum of the processing times, p_j, over all jobs.

Moreover, an optimal solution to the problem when the objective is to minimize the weighted sum of the completion times under the condition that there exist no precedence constraints between the jobs and only one machine is available for processing the jobs, is to sequence the jobs according to *Weighted Shortest Processing Time first (WSPT)* principle, also known as Smith's rule. More specifically, this means that the jobs are scheduled in a non-increasing order of the ratios w_j/p_j. The computational complexity for performing this algorithm is $\mathcal{O}(n \log n)$. (Smith, 1956)

Theorem 1. *The WSPT-principle gives an optimal solution for the 1 | | $\sum w_j C_j$-problem.*

Proof. Assume a schedule S is optimal and suppose that it is not in order of the WSPT-principle. In such a schedule there must be at least two adjacent jobs, J_j and J_i, such that

$$\frac{w_j}{p_j} < \frac{w_i}{p_i}.$$

and that J_j is processed before J_i and job J_j is started being processed at time t. Now let these jobs change places in S resulting in a new schedule S' where instead the processing of job J_i starts at time t and is followed by job J_j. No other jobs in the original schedule are affected by the switch and hence, the difference between the values of the objective function $\gamma(S)$ and $\gamma(S')$ is due only to jobs J_j and J_i. The total weighted completion time caused by jobs J_j and J_i under S is

$$(t + p_j)w_j + (t + p_j + p_i)w_i,$$

whereas under S' it becomes

$$(t + p_i)w_i + (t + p_i + p_j)w_j.$$

Calculating the difference between $\gamma(S')$ and $\gamma(S)$ and using that $\frac{w_j}{p_j} < \frac{w_i}{p_i}$ yields the following inequality:

$$\gamma(S') - \gamma(S) = w_j p_i - w_i p_j < 0.$$

Since this is a contradiction to the optimality of schedule S this concludes the proof. ■

3.1.2 $\circ \mid \{\circ, \circ, \mathbf{prec}, \circ, \circ, \circ\} \mid \sum w_j C_j$

The $\circ \mid \{\circ, \circ, \text{prec}, \circ, \circ, \circ\} \mid \sum w_j C_j$-problem (commonly referred to as $1 \mid \text{prec} \mid \sum w_j C_j$) is the same problem as the $1 \mid \mid \sum w_j C_j$-problem but with the difference that there exist precedence constraints between the jobs.

Definition 5. *$G = (V, E)$ is a directed acyclic graph with vertex set $V = \{1, 2, ..., n\}$ and edges E where each edge $(i, j) \in E$ is an edge from vertex i to vertex j. In G there exists no cycles, hence, the name acyclic.*

An example of a directed acyclic graph is presented in Figure 3.1.

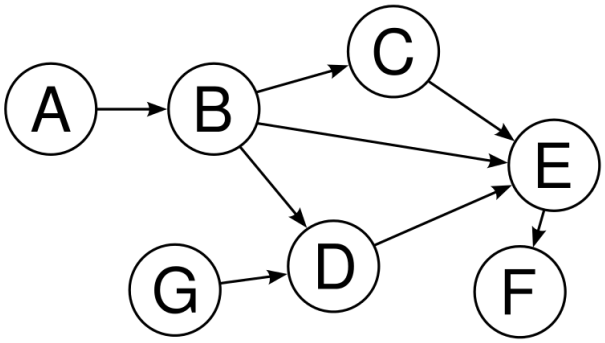

Figure 3.1: An example of a directed acyclic graph

The precedence constrains to $1 \mid \text{prec} \mid \sum w_j C_j$-problem must be given by a directed acyclic graph $G = (V, E)$. If there exists an edge $(i, j) \in E$, it is denoted

as $J_i < J_j$ which means that job J_i must be processed before job J_j (Graham et al., 1979).

Definition 6. *Define* $Z_j = \{J_i \mid (j,i) \in E\}$ *as the immediate successors to job* J_j.

Definition 7. *Define* $D_j = \{J_i \mid (i,j) \in E\}$ *as the immediate dependencies to job* J_j.

This problem can as the $1 \mid \mid \sum w_j C_j$-problem be formulated as a MIP by extending the MIP formulation with the following constraints:

$$x_{i,j} = 1 \quad \forall\, (i,j) \in E.$$

However, this problem is proven to be NP-hard in the strong sense (Lawler et al., 2006), as opposed to the $1 \mid \mid \sum w_j C_j$-problem where Smith's rule finds the optimal solution in polynomial time. The problem is also related to an open problem in scheduling listed by Schuurman and Woeginger (1999). Who formulated it as follows:

> "Prove that $1 \mid \text{prec} \mid \sum C_j$-problem and $1 \mid \text{prec} \mid \sum w_j C_j$-problem do not have polynomial time approximation algorithms with performance guarantee $2 - \delta$, unless P = NP".

Summarized, the $1 \mid \text{prec} \mid \sum w_j C_j$-problem is a very well-known problem within scheduling for which a lot of research has been carried out during the years.

A 2-approximation algorithm

There exists a 2-approximation algorithm for the $1 \mid \text{prec} \mid \sum w_j C_j$-problem. It is considered the state-of-the-art solution method since it can be computed in polynomial time and a 2-approximation is the best known approximation factor (Ambühl et al., 2011) and the core idea behind it was originally discovered and proven by Sidney (1975). The algorithm is decomposition based, meaning that it splits the original problem into hopefully smaller problems. The solution to these smaller problems are proven to be scheduled in a sequence in the optimal schedule for the original problem. Applying random scheduling but with respect to the precedence constraints is proven to give a 2-approximation result for the smaller problem if the smaller problem is chosen according to the decomposition method. Combining the approximated solutions for the smaller problems are proven to give a 2-approximation for the original problem.

The method was later rediscovered and proven by Chekuri (1998), which also presented an efficient way to calculate the decomposition of the problem. This algorithm and its proofs presented by Chekuri (1998) are summarized below.

Definition 8. *Let $q_j = \frac{p_j}{w_j}$ define the rank of job J_j. Let $q(I) = \frac{p(I)}{w(I)}$ define the rank of a set of jobs I where $p(I) = \sum_{J_j \in I} p_j$ and $w(I) = \sum_{J_j \in I} w_j$. Define the rank $q(G)$ of a graph G as the same as the rank of a sequence containing the same jobs as the graph.*

This is a reasonable definition since the rank is independent of the order of the jobs.

Definition 9. *G_j is the sub-graph of the graph G which has job J_j and all jobs preceding job J_j in it.*

Definition 10. *A sub-graph $G^{'}$ of G is said to be precedence closed if for every job $J_j \in G^{'}$, G_j is a sub-graph of $G^{'}$.*

Definition 11. *G^* is a precedence closed sub-graph of G with minimum rank.*

Note that the rank of a sub-graph is equal to the rank of any sequence containing the same jobs as the sub-graph independent of the order of the jobs and that G^* can be equal to G.

Definition 12. *A segment in a schedule S is a set of jobs that are scheduled in a consecutive order in S.*

Theorem 2. *The optimal schedule for G^* is a segment of an optimal schedule S^* for G starting at time zero.*

Proof. This is trivial if G^* is G. In other cases, suppose that the theorem is not true. Let S^* be an optimal schedule to G where the optimal schedule for G^* does not occur as a segment starting at time zero. Let $A_1, A_2, ..., A_k$, $k \geq 1$ be segments of the optimal schedule for G^* which occurs in S^* where the starting time of A_i is smaller than A_{i+1} for $i = 1, 2, ..., k-1$. Let B_1 be the segment of S^* before A_1, which starts at time zero, B_{k+1} the segment after A_k and B_i the segment between A_{i-1} and A_i for $i = 2, 3, .., k$. Note that B_1 and B_{k+1} can be empty segments. Let $B^j = \cup_{i=1}^{j} B_i$, $j = 1, 2, ..., k+1$ and $A^j = \cup_{i=1}^{j} A_i$, $j = 1, 2, ..., k$. Now, set $\alpha = q(G^*) = q(A^k)$, from the definition of G^* it is known that $q(B^j) \geq \alpha$, $j = 1, 2, ..., k$, if that was not true then $B^j \cup G^*$ would be precedence-closed and have rank less than α. Another observation which can be done is that $q(A^k - A^j) \leq \alpha$, $j = 1, 2, ..., k$ otherwise $q(A^j) < \alpha$, which would imply that A^j has lower rank than G^* while being precedence closed.

Now, let $S^{'}$ be a new schedule with $A_1, A_2, ..., A_k$ at the beginning and $B_1, B_2, ..., B_{k+1}$ after the end of A_k with their consequent order kept intact. Note that this is a feasible schedule since $A^k = \cup_{i=1}^{k} A_i$ contains all jobs in G^*, which is precedence closed by definition and the order of $B_1, B_2, ..., B_{k+1}$ comes from another feasible schedule. Let Δ be the difference in weighted completion time between S^* and $S^{'}$. If it can be proven that $\Delta \geq 0$ the proof is complete. Since the position and therefore the completion time of the jobs in B_{k+1} are unchanged, these can be ignored when calculating Δ. Let $\Delta(A_i)$ and $\Delta(B_i)$ be the sum of the

differences in weighted completion time for the jobs in A_i and B_i respectively between S^* and $S^{'}$. Hence, $\Delta = \sum_{i=1}^{k} \Delta(A_i) + \Delta(B_i)$.

It is easy to find $\Delta(A_i)$, since all the B_i, $i = 1, 2, ..., k$ are now scheduled after A_k, the completion time of the jobs in A_i is shorter with the sum of the processing time of B^i. Hence, it can be expressed as

$$\Delta(A_i) = w(A_i)p(B^i). \tag{3.1}$$

With the same reasoning it is easy to find that the jobs of B_i, $i = 1, 2, ..., k$ are placed after $A^k - A^{i-1}$ which implies that the completion time is $p(A^k - A^{i-1})$ longer for each job. Hence, it can be written as

$$\Delta(B_i) = -w(B_i)p(A^k - A^{i-1}). \tag{3.2}$$

As observed earlier $q(B^i) \geq \alpha$ which, from the definition of rank, implies $p(B^i) \geq \alpha w(B^i)$ since $q(B^i) = \frac{p(B^i)}{w(B^i)}$ and also $q(A^k - A^j) \leq \alpha$ which implies $p(A^k - A^j) \leq \alpha w(A^k - A^j)$. Combining Equation 3.1 and Equation 3.2 with these inequalities yields:

$$\begin{aligned}
\Delta &= \sum_{i=1}^{k} \Delta(A_i) + \Delta(B_i) \\
&= \sum_{i=1}^{k} w(A_i)p(B^i) - \sum_{i=1}^{k} w(B_i)p(A^k - A^{i-1}) \\
&\geq \alpha \sum_{i=1}^{k} w(A_i)(\sum_{j=1}^{i} w(B_j)) - \alpha \sum_{i=1}^{k} w(B_i)(\sum_{j=i}^{k} w(A_j)) \\
&= 0.
\end{aligned}$$

Hence, $\Delta \geq 0$ and the objective value can only be unchanged or improved by placing the jobs G^* as a segment starting at time zero. ■

The algorithm uses the above theorem as follows: Given a directed acyclic graph G which represents the jobs and their precedence constraints, calculate G^* and schedule the jobs in G^* in an arbitrary order. Repeat this with $G = G - G^*$ until G^* is equal to G.

Chekuri (1998) proves that this is a 2-approximation algorithm by proving the following theorems:

Theorem 3. *If G^* is G, then the value of an optimal schedule γ^* fulfills the following inequality $\gamma^* \geq \frac{w(G)p(G)}{2}$.*

Theorem 4. *Any feasible schedule to G with no idle time has the objective value $\gamma \leq p(G)w(G)$.*

Hence, scheduling the jobs in G^* arbitrarily, gives a 2-approximation. But the approximation can in practice be improved if the jobs are scheduled with a more sophisticated method.

In order to use this algorithm efficiently, an efficient way of finding G^* is necessary. Chekuri (1998) presents an algorithm which converts the problem to a source to sink minimum cut problem, which has a variety of efficient solution methods.

Definition 13. *Given a directed acyclic graph $G = (V, E)$ and a real positive number λ, $G_\lambda = (V \cup \{s, t\}, E^{'}, c)$ is a directed graph with capacities $c(e)$ for each edge $e \in E^{'}$ and s is a source vertex and t is a sink vertex. Moreover, let $p_s = p_t = w_s = w_t = 0$, $E^{'} = \{(s, j), (j, t) \mid 1 \leq j \leq n\} \cup \{(j, i) \mid J_i < J_j\}$ and*

$$c(e) = \begin{cases} p_j, & \text{if } e = (j, t) \\ \lambda w_j, & \text{if } e = (s, j) \\ \infty, & \text{otherwise} \end{cases}.$$

The following theorem finish the proof for the algorithm.

Theorem 5. *Given a directed acyclic graph G, there exists a sub directed acyclic graph with rank less than or equal to λ if and only if the source to sink minimum cut in G_λ has a value at most $\lambda w(G)$. Let (A, B) be a cut whose cut value is bounded by $\lambda w(G)$ then $q(A - \{s\}) \leq \lambda$ and $A - \{s\}$ is precedence closed in G.*

Proof. First, let (A, B) be a source to sink cut whose cut value is bounded by $\lambda w(G)$. If $A - \{s\}$ is not precedence closed on G then there exists a pair of vertices i and j such that $J_i < J_j$, $j \in A$ and $i \notin A$. Then $c((j, i)) = \infty$ which is a contradiction since $c(A, B) \leq \lambda w(G)$. Using this fact together with the definition of G_λ,

$$\begin{aligned} c(A, B) &= \sum_{j \in A} p_j + \lambda \sum_{j \notin A} w_j \\ &= \sum_{j \in A} (p_j - \lambda w_j) + \lambda \sum_{j \in V} w_j \\ &= \sum_{j \in A} (p_j - \lambda w_j) + \lambda w(G) \end{aligned}.$$

This implies together with the fact $c(A, B) \leq \lambda w(G)$ the following inequality $\sum_{j \in A} (p_j - \lambda w_j) \leq 0$. Which can be rewritten as

$$\begin{aligned} 0 &\geq \sum_{j \in A} (p_j - \lambda w_j) \\ \sum_{j \in A} \lambda w_j &\geq \sum_{j \in A} p_j \\ \lambda &\geq \frac{\sum_{j \in A} p_j}{\sum_{j \in A} w_j} = q(A) = q(A - \{s\}) \end{aligned}.$$

Now let (A, B) be a cut which has $q(A - \{s\}) \leq \lambda$.

$$\begin{aligned} q(A - \{s\}) = q(A) = \frac{p(A)}{w(A)} &\leq \lambda \\ p(A) &\leq \lambda w(A) \\ p(A) - \lambda w(A) &\leq 0 \\ \sum_{j \in A} (p_j - \lambda w_j) &\leq 0 \\ \sum_{j \in A} p_j + \lambda \sum_{i \notin A} w_j &\leq \lambda \sum_{j \in V} w_j \\ c(A, B) &\leq \lambda \sum_{j \in V} w_j = \lambda w(G) \end{aligned}$$

■

Using this theorem, one can find G^* by using binary search or some other suitable method over λ in order to find the smallest value as possible, which generates a cut in G_λ which is bounded by $\lambda w(G)$. Chekuri (1998) proposes other more sophisticated methods which can calculate all values of λ which corresponds to a certain cut with the same complexity as a single maximum flow calculation.

Chapter 4

Mathematical Model

Presented in the following chapter are the reasons for why the test case prioritization problem can be modeled as a scheduling problem on the $\alpha \mid \beta \mid \gamma$-form and more specifically, as a schedule intended for a single machine, with general precedence constraint and where the objective is to minimize the total weighted completion times, namely a $1 \mid \text{prec} \mid \sum w_j C_j$-problem. More importantly, argumentation to why it should be modeled as such is also presented. Followed after the presented model different possible solution algorithms are discussed from an industrial perspective.

4.1 Prioritization Problem as a Scheduling Problem

It is possible to model the prioritization of test cases as a schedule with precedence constraints since there almost always exist dependencies between the test cases, where certain test cases preferably have to be processed before others. This is perfectly in line with the discussed definition of precedence constraints in Section 3.1.2. If every single test case is independent of the others it is simply a special case where the precedence constraints in the model are non-existing. However, in the industry, the precedence graph G or the dependency structures are not restricted to be of any certain type as for example a tree, rooted tree or series parallel, for which there already exist some solution methods presented by Horn (1972); Burns and Steiner (1981). Hence, the precedence constraints are considered to be general and the model could thus be used for any dependencies which can be described by a directed acyclic graph G.

As discussed in Chapter 1, given a measure, a common objective when testing is to maximize the area under a graph generated by the associated schedule (see example in Figure 1.3). Theoretically, the best measure to use is the number

of faults a test case will detect. However, this is not possible in practice and a measure which is correlated to that can be used instead. The objective which is to maximize the area can be converted to minimizing a weighted sum of completion times as in a problem on the form $\alpha \mid \beta \mid \sum w_j C_j$.

Theorem 6. *Maximizing the area under a graph in the same context as Figure 1.3 is the same objective as minimizing a weighted sum of completion times as a problem on the form $\alpha \mid \beta \mid \sum w_j C_j$, if the weights, w_j, of the jobs are set to the value of the measure for the test case.*

Proof. See the test cases as jobs which need to be scheduled. Take an arbitrary test case and call it J_j from the test suite T with corresponding value w_j for the objective measure and processing time p_j. Let the total sum of processing time over the test cases be denoted τ. Now, schedule each test case which will give a completion time C_j to all test cases. The area under the graph added by a single test case j, can be expressed as

$$\frac{p_j w_j}{2} + w_j(\tau - C_j),$$

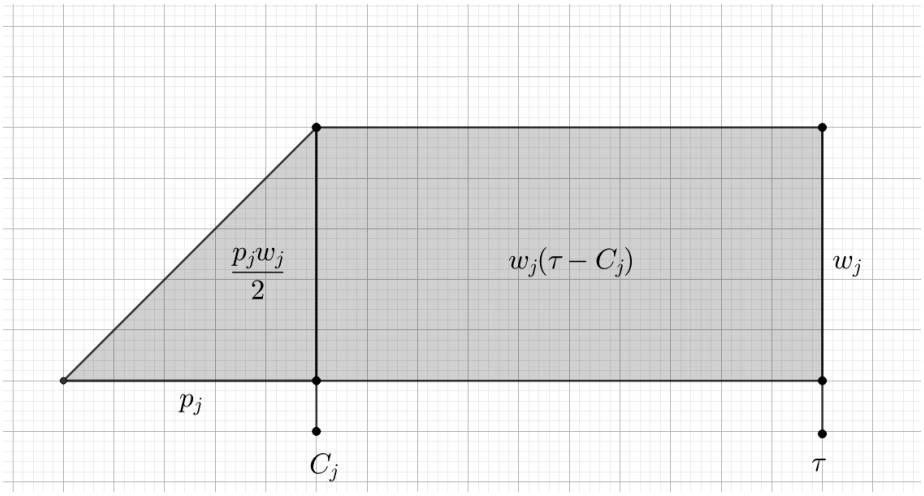

Figure 4.1: The area generated by a scheduled test case

which is shown in Figure 4.1. The objective function $f(S)$ which is to be maximized is the total sum of the areas over all test cases. This can be expressed as:

$$f(S) = \sum_{J_j \in S} \frac{p_j w_j}{2} + w_j(\tau - C_j) = \sum_{J_j \in S} \frac{p_j w_j}{2} + \tau w_j - w_j C_j.$$

Since the first and the second term of the last expression are constant regardless S, the objective function is maximized if the third term is minimized. Hence,

the testing objectives in test case prioritization within software testing is the same as the objective of a problem on the form $\alpha \mid \beta \mid \sum w_j C_j$, where the objective is to minimize

$$\sum_{J_j \in S} w_j C_j.$$

■

From the existing testing procedure at BT one can easily see that a prioritization order of the test cases can be modeled as a schedule where there is only a single (1) machine to process the jobs. In this case the machine represents a test operator. However, there could be a flexible number of test operators, but even though there are multiple testers who are executing the test cases, it is always possible for any test operator to start working on the next scheduled test case that is possible to execute. To conclude, several test operators do not work in the same way as if multiple machines were to process the jobs since all test operators do not work continuously with testing during the days as would be the case for multiple machines. Hence, it is accurate to model it as having a single machine.

The testing at BT is mainly done manually, but since the schedule can be seen as a priority list and thus, if the testing process was to be automatized, such a schedule would still be useful. Furthermore, extending the model to include multiple machines is possible. However, modeling the problem for a single machine first is necessary for adding additional machines to the model later on (Chekuri, 1998). This fact also strengthens the argument for modeling the problem for a single machine to begin with, as was discussed previous paragraph.

Seeing the objective function as weighted sum of completion times is also reasonable since the model then can be applicable in cases where a measure which can assign a value to each test case is chosen as objective in the prioritization problem. The versatility of the model can be exemplified by looking at the different measures described by Khandelwal and Bhadauria (2013) as previously mentioned in Section 1.3. Results can be obtained for any of these measures by letting the weights represent one of the measures.

Summarizing the argumentation brought up in this chapter, a conclusion that the problem is suitable and appropriate to model as a $1 \mid \text{prec} \mid \sum w_j C_j$-problem can be drawn. This is also the answer to **RQ1**.

If no precedence constraints were to be considered, the $1 \mid \text{prec} \mid \sum w_j C_j$-problem is the same as the $1 \mid \mid \sum w_j C_j$-problem. Then, as proven by Smith (1956), the optimal solution would be to, given a measure, the value of that measure w_j and the processing time p_j of each test case, prioritize the test cases in non-increasing order of $\frac{w_j}{p_j}$. In addition, if no processing times are considered, the test cases can be seen as having the equal processing times, which would be optimally solved by ordering the test cases in non-increasing order of w_j. This solution to the prioritization problem when ignoring dependencies is in line with

previous research discussed in Chapter 2 where multiple researches suggests to prioritize in a non-increasing order of a measure. This increases the validity of the model.

4.2 Requirements on the Solution Method

The solution algorithm for solving the problem in this thesis has to fulfill certain requirements since it should be applicable in an industrial setting. It is not unusual for a test suite of a product to consist of more than 1 000 test cases. Therefore, the algorithm must be able to handle numbers in that range, i.e. a solution should be obtained within reasonable time even for large test suites as input. This disqualifies some potential approaches instantly, such as brute force, Dynamic Programming or Branch and Bound, where the computational complexity is large and a solution cannot then be generated for large data sets in polynomial time.

Within the industry, the dependencies between the test cases are not necessarily structured in a specific and predetermined way. For the model/algorithm to be generalizable and useful in different projects where the dependency structures vary, it is important that it is not restricted to any ad hoc approach which is only possible to apply when the dependencies are structured as for instance as trees or as rooted trees.

Some test operators, mainly the most experienced ones, may want to deviate from the proposed prioritization list and execute test cases he or she value more despite the calculated benefits given by the results from the mathematical model. It could for example be due to practical reasons in the testing process or other factors that the tester value and that are not taken into consideration in the model. If such a decision is made by a tester, an updated priority list should be able to be obtained in reasonable time i.e. there should not be any problems with deviating from the calculated schedule. The prioritization list can be seen as a decision support tool rather than a strict scheme by which everything should be done mechanically. Hence, developing an algorithm that can be rerun several times a day is of great importance.

4.3 Algorithms

Some possible solution algorithms are listed in this section. The presented algorithms are found or developed through studies of literature within the subject. As discussed earlier the performance of the produced schedule combined with the execution time are very important factors for how well they are suited to be used in an industrial setting, hence, the algorithms are discussed with these factors in focus.

4.3.1 General MIP-solver

As stated previously in Section 3.1.2, the $1 \mid \text{prec} \mid \sum w_j C_j$-problem can be formulated as a MIP. However, it cannot be solved optimally in polynomial time with any available general solver within an industrial setting, i.e. where the input consist of around 1 000 jobs or more since the number of constraints as well as the number of variables grow exponentially with the number of jobs n. For example, a quick evaluation of an authentic test suite from the industry consisting of only 210 test cases results in 43 890 variables and approximately 4.6 million constraints. In addition, Potts (1985) solved the problem optimally but only for instances with up to 100 jobs. Hence, the potential approach to formulate it as MIP or with any other existing method try to solve the $1 \mid \text{prec} \mid \sum w_j C_j$-problem optimally can immediately be disqualified for further investigation. It is simply not practically suitable and thus not feasible as a model as requested by the industry.

4.3.2 Greedy Algorithm

Smith's rule for the optimal schedule of the $1 \mid \mid \sum w_j C_j$-problem can easily be converted to an intuitive approximation algorithm for the $1 \mid \text{prec} \mid \sum w_j C_j$-problem. This can be done by first creating an optimal schedule S for the corresponding $1 \mid \mid \sum w_j C_j$-problem. Note that the schedule is probably infeasible to the $1 \mid \text{prec} \mid \sum w_j C_j$-problem since it does not account for any of the given precedence constraints. S can then be converted to a feasible schedule by creating a new schedule $S^{'}$ by simply taking the first job in S which can be feasibly scheduled in $S^{'}$ and schedule it as the next job in $S^{'}$. This can then be repeated until all jobs have been scheduled in $S^{'}$. The algorithm is described with psuedo code in Algorithm 1. This algorithm will hereafter be referred to as the Greedy-algorithm since it has a greedy approach where it tries to maximize the measure per time unit without considering the future consequences which arises due to the precedence constraints.

Algorithm 1: Greedy Algorithm

Data:

- G, directed acyclic graph of the jobs

Result:

- S, a schedule of G

```
begin
    S ← ∅ ;
    S' ← Ordered list of the job indices j in G in non-increasing order of w_j / p_j ;
    while S' ≠ ∅ do
        i ← 0 ;
        j ← S'[i] ;
        // Make sure the job can be scheduled at the current time
        while D_j ⊈ S do
            // If not try with the next job in the list
            i ← i + 1 ;
            j ← S'[i] ;
        end while
        Append J_j to S ;
        Remove S'[i] ;
    end while
    return S;
end
```

The complexity of this algorithm is low which makes it a good candidate as solution method since a large number of jobs can be scheduled quickly. However, there is a high risk associated with this approach since no worst-case performance can be guaranteed. But some reasoning on the performance in terms of optimality can be made. If there are just a few precedence constraints the algorithm should generate a schedule which is close to the optimal value since the few precedence constraints probably do not have any significant effect on the objective value. On the other hand, if there are a large number of precedence constraints, then the number of feasible schedules are fewer. This would decrease the gap between the value of the optimal schedule and the worst possible schedule and therefore, a schedule with numerous precedence constraints generated by this algorithm should not be that far off from the optimal schedule either.

Summarized, the Greedy-algorithm is an intuitive and simple algorithm which has the potential to give good solutions in some cases. However, the performance of the schedule obtained is presumably dependent on the dependency structure of the problem.

4.3.3 Value Algorithm

One issue with the Greedy-algorithm is that the information of the precedence graph G is completely disregarded as hints for the optimal schedule and instead only used as constraints. However, the impact of this issue could be limited by assigning a value to each job J_j, $\tilde{w}_j$ which is dependent on the structure of G and then schedule the jobs in a non-increasing order of that value. Similar to the methods presented by Acharya et al. (2010); Haidry and Miller (2013). Using this insight a new algorithm can be designed which takes the structure of G into account. This algorithm is presented below.

Let $\tilde{w}_j$ be the value which accounts for the position in the graph G of job J_j. The proposed $\tilde{w}_j$ is given by

$$\tilde{w}_j = \frac{\frac{w_j}{\sum_{i=1}^{n} w_i} + \rho \sum_{J_i \in Z_j} \frac{\tilde{w}_i}{|D_i|}}{p_j},$$

where ρ is a real valued parameter. By using this formula the value should account for the weight of the job, the processing time of the job and the position of the job in the graph G. In order to make the parameter ρ independent of the sizes of the weights, the weights are normalized in the formula. The denominator $|D_i|$ makes sure that a job which has many direct dependencies is valued less in the formula for jobs which have that job in their direct successors Z_j. Note that $|D_i|$ can be zero, but this is not a problem in the formula since the sum goes over all direct successors which means that all jobs which are included in a sum have at least one direct dependency.

Assigning values to the jobs according to this formula should have many advantages compared to the greedy algorithm. For example, a job which enables a job with high value will have higher value than a job which enables a job with low value. In the Greedy-algorithm, facts like this are be disregarded.

Moreover, an algorithm for calculating the values of the jobs in a directed acyclic graph is presented with pseudo code in Algorithm 2.

Algorithm 2: Value Algorithm - value calculation

Data:

- G, directed acyclic graph of the jobs.
- ρ, a real valued parameter

Result:

- S, a list of job indicies in non-increasing order of their value.

begin

```
// Initiate a set to hold the jobs which has a value
   calculated
```

$L \leftarrow \emptyset$;

```
// Initiate a list with length |G| to hold the job index and
   job value
```

$V \leftarrow \emptyset$;

```
// Continue until all values are calculated
```

while $|G| > |L|$ **do**

  foreach J_j *in* G **do**

    ```
    // Check if all nessecary values has been calculated
    ```

    if $J_j \notin L$ *and* $Z_j \subseteq L$ **then**

      Add J_j to L ;

      $v \leftarrow (\frac{w_j}{\sum_{i=1}^{n} w_i} + \rho \sum_{J_i \in Z_j} \frac{V[i][1]}{|D_i|})/p_j$;

      $V[j] \leftarrow (j, v)$;

    end if

  end foreach

end while

Sort V in non-increasing order of the value (each element's second attribute) ;

$S \leftarrow \emptyset$;

foreach (j, v) *in* V **do**

  Add j to S ;

end foreach

return S;

end

Further, the proposed main algorithm for scheduling jobs works as follows: calculate the values for a given directed acyclic graph G according to Algorithm 2. Schedule the job J_j which has the highest value among the jobs which can be scheduled at the current time. Then, remove J_j from G and repeat until G is an empty graph. The algorithm is presented with pseudo code in Algorithm 3.

Algorithm 3: Value Algorithm - scheduling

Data:
- G, directed acyclic graph of the jobs

Result:
- S, the approximated schedule

```
begin
    S ← ∅ ;
    while |G| > 0 do
        S' ← Run value calculation on G ;
        i ← 0 ;
        j ← S'[i] ;
        // Make sure the job can be scheduled at the current time
        while D_j ⊈ S do
            // If not try with the next job in the list
            i ← i + 1 ;
            j ← S'[i] ;
        end while
        Append J_j to S ;
        Remove J_j from G ;
    end while
    return S;
end
```

The parameter ρ is presumably very dependent on the structure of G. Hence, the value needs to be investigated. Since the algorithm has relatively low complexity a good ρ can be obtained by brute forcing in smaller data sets. However, if this was to be one of the steps in the algorithm, the algorithm would not be efficient for larger sets. Hence, it is very important to find hints on how to choose a good candidate for ρ.

In order to discuss the structure of G with measures which can affect ρ, some definitions needs to be made.

Definition 14. *Let a leaf job be defined as a job J_j which has $|Z_j| = 0$. Similarly, let a root job J_j be defined as a job where $|D_j| = 0$.*

Definition 15. *Let an independent job be defined as a job which is both a leaf and a root job.*

Definition 16. *Let the depth of a job J_j be defined as $\nabla(J_j) = 1+\sum_{J_i \in D_j} \nabla(J_i)$.*

Definition 17. *Let a cluster $G^{'} = (V^{'}, E^{'})$ be defined as a directed acyclic graph which is a subgraph of $G = (V, E)$ which fulfills the following $\{(i,k) \in E \mid i = j \text{ or } k = j\} = \{(i,k) \in E^{'} \mid i = j \text{ or } k = j\}$ for each vertex j in $V^{'}$.*

Intuitively the following measures can have an affect on ρ:

- The number of jobs and the average number of immediate dependencies of the jobs
- Percentage of leaf jobs, root jobs and independent jobs
- Average depth of the jobs, the weighted average depth of the job values and the maximum depth over the jobs
- The number of clusters, the average size of a cluster and the average size of a cluster in percentage of the size of G

Summarized, the Value-algorithm is a simple and intuitive approach where some of the drawbacks with the Greedy-algorithm have been addressed. The algorithm has therefore potential of at least generating better results than the Greedy-algorithm. However, the algorithm is for obvious reasons unexplored and no performance guarantees can be assured.

4.3.4 Sidney's Decomposition Algorithm

Sidney's decomposition algorithm is considered state-of-the-art when solving the $1 \mid \text{prec} \mid \sum w_j C_j$-problem (as discussed in Section 3.1.2). Given a directed acyclic graph G, one way of how the theories can be converted to an algorithm is to calculate a precedence closed sub directed acyclic graph G^* with minimal rank. Then schedule G^* as good as you can. After that repeat the same steps with $G - G^*$ until there are no jobs left, which will occur when $G^* = G$.

However, the algorithm is complicated to implement, since it requires several decomposition calculations. Also the decomposed problem is not guaranteed to be a smaller problem (when G^* is equal to G), which would make the results same as if the initial set was scheduled using the algorithm which is used to schedule the smaller problems.

Also, no implementation of the algorithm has been found in previous research. As discussed in Section 3.1.2, a core part of the algorithm is to a find a dependency closed sub-graph G^* of G with minimal rank. Pseudo code for such algorithm is presented in Algorithm 4. Since the methods proposed by Chekuri (1998) for calculating all breakpoints of λ are very complex, a simpler binary search has been chosen for the algorithm. For the binary search an additional positive parameter ϵ is introduced, which corresponds to the precision of the binary search. A small ϵ gives high probability of finding the true G^* compared to a larger, but with a trade off in computation time.

Algorithm 4: Sidney's Decomposition Algorithm - finding a minimal sub graph

Data:

- G, directed acyclic graph of the jobs
- $\lambda_{\min}$, the minimum value of λ
- ϵ, the precision for the binary search

Result:

- G^*, a precedence closed sub-graph of G
- λ, the value which bounds the rank of G^*

begin

$G^{'} \leftarrow$ Create G_λ with two dummy vertices, a source s and a sink t ;
$\lambda_{\max} \leftarrow \lambda_{\min} + 2w(G)$;
$\lambda_{\text{previous}} \leftarrow 0$;
while $|\lambda_{previous} - \frac{\lambda_{\max}+\lambda_{\min}}{2}| > \epsilon$ *or* A *is* $\{s\}$ **do**
  $\lambda \leftarrow \frac{\lambda_{\max}+\lambda_{\min}}{2}$;
  $\lambda_{\text{previous}} \leftarrow \lambda$;
  $A \leftarrow$ Find the min-cut of $G^{'}$ with capacities according to the definition of G_λ ;
  if A *is* $\{s\}$ **then**
    `// No cut exists which is bounded by` $\lambda w(G)$
    $\lambda_{\min} \leftarrow \lambda$;
  else
    `// A cut which is bounded by` $\lambda w(G)$ `was found`
    $\lambda_{\max} \leftarrow \lambda$
  end if
end while
$S \leftarrow A - \{s\}$;
return S, λ;

end

After obtaining a possible G^* it must be scheduled. The process can then be repeated with $G - G^*$ until a complete schedule is found, when $G = G^*$. Pseudo code for such algorithm is presented in Algorithm 5.

Algorithm 5: Sidney's Decomposition Algorithm - scheduling

Data:
- G, directed acyclic graph of the jobs

Result:
- S, the approximated schedule

```
begin
    S ← ∅ ;
    λ_min ← 0 ;
    while |G| > 0 do
        G*, λ_min ← Run Algorithm 4 with G and λ_min ;
        S' ← Run Algorithm 1 with G* ;
        foreach j in S' do
            Add J_j to S ;
            Remove J_j from G ;
        end foreach
    end while
    return S;
end
```

4.3.5 Random Scheduling Algorithm

As of today, the execution order at BT (presumably in other comparable companies as well) is set arbitrarily. In order to usefully evaluate other solution algorithms, the arbitrary or random scheduling algorithm must also be evaluated. If the other algorithms performs similarly, then there is no gain in implementing a more sophisticated method.

To model such scheduling process, an algorithm which takes all jobs in G and puts them in a random order is designed. However, the algorithm must fulfill the precedence constraints, which can easily be done by scheduling the first feasible job at random until all jobs are scheduled. Pseudo code for this algorithm is presented in Algorithm 6.

Algorithm 6: Random Scheduling Algorithm

Data:

- G, directed acyclic graph of the jobs

Result:

- S, a schedule of G

```
begin
    S ← ∅ ;
    S' ← The jobs indicies of G in random order ;
    while S' ≠ ∅ do
        i ← 0 ;
        j ← S'[i] ;
        // Make sure the job can be scheduled at the current time
        while D_j ⊈ S do
            // If not try with the next job in the list
            i ← i + 1 ;
            j ← S'[i] ;
        end while
        Append J_j to S ;
        Remove S'[i] ;
    end while
    return S;
end
```

Chapter 5

Evaluation Method

In this chapter the method of acquiring empirical data as evidence for answering RQ1.1 is described and argued for. Since only one authentic test suite was available from the case study, more empirical data were needed to be gathered in order to answer RQ1.1 generally. Therefore, a large number of test suites with different sizes and precedence structure were generated. The algorithms were then applied to each data set and the value of the produced schedule and the execution times of the algorithms were noted. More details about each step are described in this chapter.

5.1 Data Generation

In order to generate test suites to evaluate the algorithms on, a test suite generation algorithm was implemented. The algorithm takes the parameters n and ζ as inputs, which corresponds to the number of test cases and the "intensity" of the resulting directed acyclic graph G respectively. The algorithm generates n test cases, J_j, which take values $w_j \sim U\{0, 10\}$ and processing times $p_j \sim U(0.1, 10)$, where $U\{a, b\}$ is the uniformed integer distribution between the integers a, b and $U(a, b)$ is the uniformed continuous distribution between the real numbers a, b. The precedence constraints was generated according to that test case J_i precedes test case J_j with probability

$$q_{i,j} = \begin{cases} \zeta \frac{i}{n(j-1)}, & \text{if } i < j \\ 0, & \text{otherwise} \end{cases},$$

where $0 \leq \zeta \leq n$. Due to this probability the generated test suites have similar appearances as the test suite found in the case study. Pseudo code for the algorithm is presented in Algorithm 7. Six examples of test suites which have been generated with the algorithm are presented in Figure 5.1 and Figure 5.2.

Note the similarity of the cluster formations between the two figures when n varies but ζ is fixed. The purpose of including these example figures in this thesis is to give a conceptual understanding of a generated precedence constraints. Note that since the names, processing times and the values in these examples are randomly generated, the actual numbers are irrelevant.

Figure 5.1: Test suite generated with $n = 5$ and $\zeta \in \{1, 2.5, 5\}$ respectively

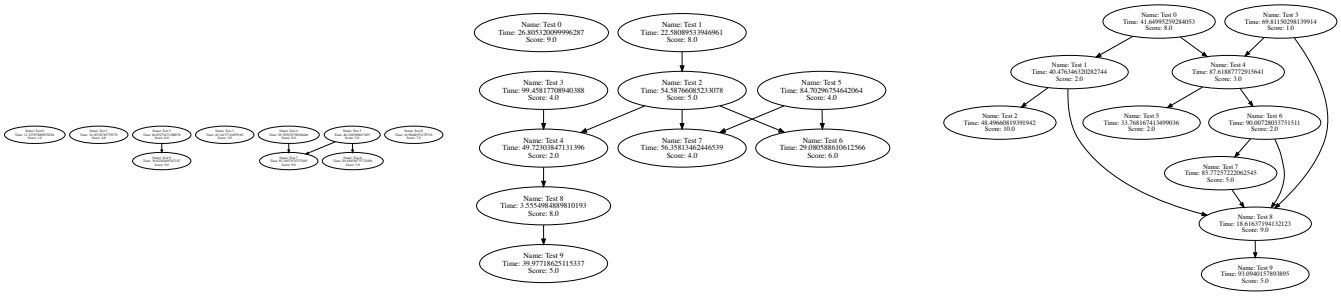

Figure 5.2: Test suite generated with $n = 10$ and $\zeta \in \{1, 2.5, 5\}$ respectively

Algorithm 7: Generation of a random directed acyclic graph G

Data:
- n, the size of G
- ζ, the intensity of G

Result:
- G, directed acyclic graph of the jobs

begin
  $G \leftarrow \emptyset$;
  while $|G| < n$ **do**
    $D \leftarrow \emptyset$;
    $i \leftarrow |G| + 1$;
    for J_j *in* G **do**
      `// Draw` u `from a uniformed distribution`
      $u \sim U(0, 1)$;
      if $u < \zeta \frac{i}{n(j-1)}$ **then**
        Add J_j to D ;
      end if
    end for
    Draw p_i from $U(0, 10)$;
    Draw w_i from $U\{0, 10\}$;
    Add J_i to G, with weight w_i, process time p_i and immediate dependencies D ;
  end while
  return G;
end

5.2 Finding a good ρ

In order to investigate the parameter ρ in the Value-algorithm, a linear regression model was implemented where ρ was explained by all the measures of G discussed in Section 4.3.3 as covariates. Obviously, some of the measures discussed are dependent on each other, which is called multicollinearity. In some regression contexts that can be a problem since the coefficients of the covariates can not be analyzed independently. But since the regression model is created and used for prediction purposes solely, the problem of multicollinearity can safely be ignored (Paul, 2018).

In order to obtain training data, 781 test suites with varying properties were generated. The number of test cases, n, was varied from 10 to 395 with intensities $\zeta \in \{0.1, 1, 2, 3, 4\}$. Given a test suite, all measures of G and a good candidate for ρ according to Algorithm 8 were calculated. These calculations were later used as training data for a linear regression model.

Algorithm 8: Finding a good ρ given G

Data:
- G, a directed acylic graph

Result:
- ρ, a good ρ for G

begin
  $\rho_{\min} \leftarrow 0$;
  $\rho_{\max} \leftarrow 100$;
  $\delta \leftarrow 0.1$;
  $N \leftarrow 10$;
  while $\rho_{\max} - \rho_{\min} \geq \delta$ **do**
    Set R to a linearly spaced vector from $\rho_{\min}$ to $\rho_{\max}$ with N steps ;
    $V \leftarrow \emptyset$;
    foreach $\rho \in R$ **do**
      Set $S^{'}$ to the schedule generated by the value algorithm (Algorithm 3) with G and ρ as inputs ;
      Add the value of the schedule, $\gamma(S^{'})$, to V ;
    end foreach
    Find the index i of V which correndsponds to the minimal value ;
    $\rho_{\max} \leftarrow R[i+1]$;
    $\rho_{\min} \leftarrow R[i-1]$;
    $\rho \leftarrow R[i]$;
  end while
  return ρ;
end

5.3 Algorithm Evaluation

The evaluation of the algorithms was done with 285 test suites, which were generated by Algorithm 7. The test suites had varying intensities, $\zeta \in \{1, 2.5, 5, 7.5, 10\}$, and different number of tests cases, $100 \leq n \leq 2000$. The evaluation was performed by for each data set calculate the schedule produced by each algorithm. The resulting schedule and the running time of the algorithm was noted.

The evaluated algorithms were:

- Greedy (Algorithm 1)
- Value (Algorithm 3), with ρ according to the linear regression model discussed in Section 5.2
- Sidney, $\epsilon = 0.01$ (Algorithm 5)
- Sidney, $\epsilon = 0.1$ (Algorithm 5)
- Sidney, $\epsilon = 1$ (Algorithm 5)

- Sidney, $\epsilon = 2$ (Algorithm 5)
- Random (Algorithm 6)

The optimal value for a schedule in the considered settings is unknown since the test suites are too big to find the optimal schedule, as discussed earlier. Another appropriate comparison between the different algorithms is therefore needed to evaluate them. In order to do such comparison, the same problem but without precedence constraints (in this thesis referred to as the "unconstrained problem") is considered. The optimal schedule this problem can be found by simply applying Smith's rule. The objective value of the generated schedule can be seen as an upper bound for the constrained problem, since an added precedence constraint can only affect the optimal value in a non-decreasing way. However, if the other algorithms are implemented and divided with this value, it gives a indication of how well they perform. It is a relative measure where the conclusion can be drawn that an algorithm is quite close to the optimal value with existing precedence constraints if the measure is close to 1. Moreover, this measure will be referred to as "percentage of the upper bound" in the following chapters.

The Random-algorithm was implemented for evaluation purposes, meaning it was only used in order to compare the performance of the other more sophisticated algorithms with something. For this algorithm, the jobs were scheduled in a random order but the precedence constraints were taken into consideration, as described in previous chapter. This can thus be seen as a lower bound of the performance of any algorithm since only a conscious decision to implement a method to impair the performance of the test prioritization would perform worse than the average value of prioritizing the test cases arbitrarily. Hence, comparing the other algorithms with this one gives a measure for how the other solutions perform. Further, it also motivates why it could be of value to implement any of the proposed algorithms at all.

After the evaluation had been done on the simulated data, the exact same evaluation procedure was repeated but for an authentic test suite at BT. The reason with this additional evaluation was to compare the more general results obtained from the simulations with a real case from the industry.

Chapter 6

Evaluation Result

This chapter describes the results obtained when the algorithms were run according to the methodology explained in the previous chapter.

6.1 Linear Regression of ρ

The final linear regression model for predicting a good ρ, which was obtained by fitting the data described in Section 5.2, is presented in Table 6.1. Since multicollinearity exist between the covariates, no conclusions can be drawn from the coefficients.

The multiple R-squared for the model was 0.1379 and the adjusted R-squared 0.1255.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	33.4994	9.2366	3.63	0.0003
Number of tests	-0.0096	0.0043	-2.24	0.0256
Avg. number of dependencies	-14.7799	5.8647	-2.52	0.0119
Percentage of leaf nodes	-5.9990	9.2868	-0.65	0.5185
Percentage of root nodes	-15.2825	9.4621	-1.62	0.1067
Percentage of indep. nodes	-7.2314	7.1926	-1.01	0.3150
Avg. depth of job	-0.7695	0.9500	-0.81	0.4182
Avg. depth of value	1.6705	0.7956	2.10	0.0361
Max depth	-0.0243	0.0300	-0.81	0.4179
Number of clusters	0.0237	0.0060	3.91	0.0001
Avg. size of a cluster	0.6652	0.3665	1.81	0.0699
Avg. size of a cluster (%)	-26.0969	4.6727	-5.58	0.0000

Table 6.1: The linear regression model for predicting a good ρ

6.2 Performance of the Algorithms

In Figure 6.1 the average percentage of the upper bound of the algorithms are presented for each intensity of the test suites. The Value-algorithm was run with ρ according to the linear regression model discussed in Section 6.1.

When the intensity is low, i.e. $\zeta = 1$, all algorithms except from the case where the test cases are executed in an arbitrary order are performing good and yield results close to the optimal value of a schedule without any precedence constraints, which is given by the 100 % mark on the y-axis. The Random-algorithm is constantly generating a schedule that is 20 percentage points less in percentage of the upper bound compared to the others.

Notably, the percentage of the upper bound of the algorithms are decreasing as the intensities of the test suites are increasing. Again, there is an exception for the Random-algorithm, which appears to generate a consistent result regardless of the values on the intensity parameter. Furthermore, when $\zeta = 10$ the Greedy-algorithm performs equally good as the Value-algorithm. However, both these two are performing greater than 5 percentage points worse compared to the Sidney-algorithm independent of the parameter ϵ.

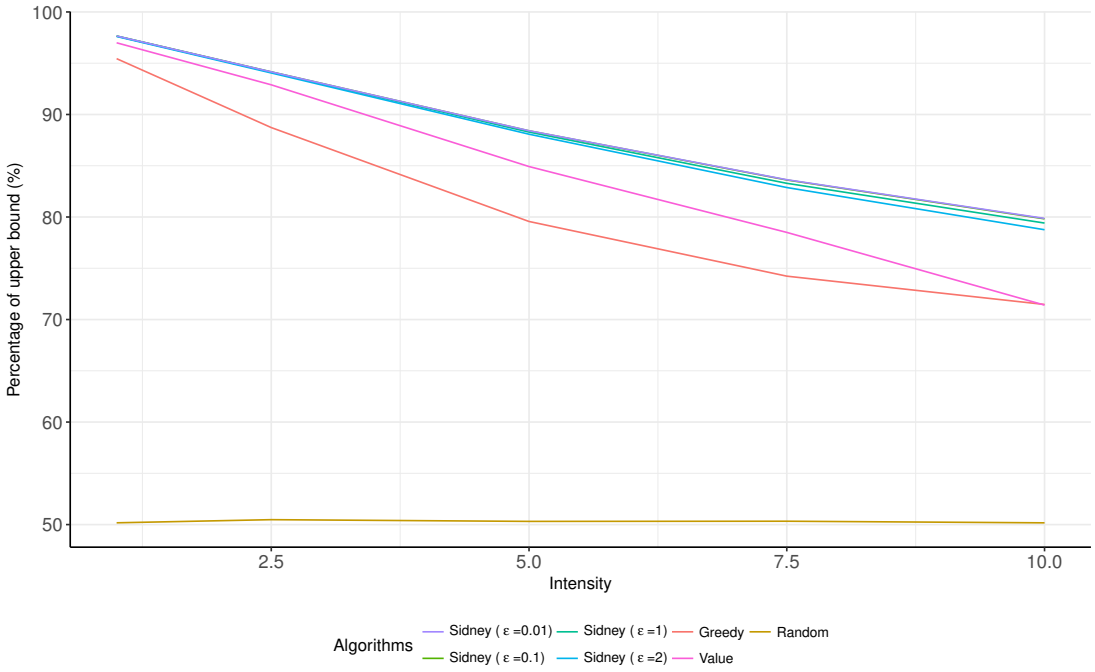

Figure 6.1: Average percentage of the upper bound compared to the intensity of the test suite

Figure 6.2 shows the average percentage of the upper bound of the different algorithms depending on the number of test cases n in the test suite. The Random-algorithm is once again performing conspicuously bad in relation to the others. Moreover, the performance of the algorithms are relatively consistent with the number of test cases within the range of 10 to 2 000.

The algorithms implemented with a Sidney decomposition are all clearly outperforming the others regardless of what value on ϵ that is used. In fact, the value of ϵ does not seem to have any big impact at all when looking at the percentage of the upper bound. This can be seen from the relatively small differences in performance between the four implemented Sidney algorithms.

One can see that the Value-algorithm generates results with higher variance than the others. However, this algorithm consistently performs better than the Greedy-algorithm but worse than the Sidney-algorithms, regardless of the value of $\epsilon \in \{0.01, 0.1, 1, 2\}$, for the considered number of test cases.

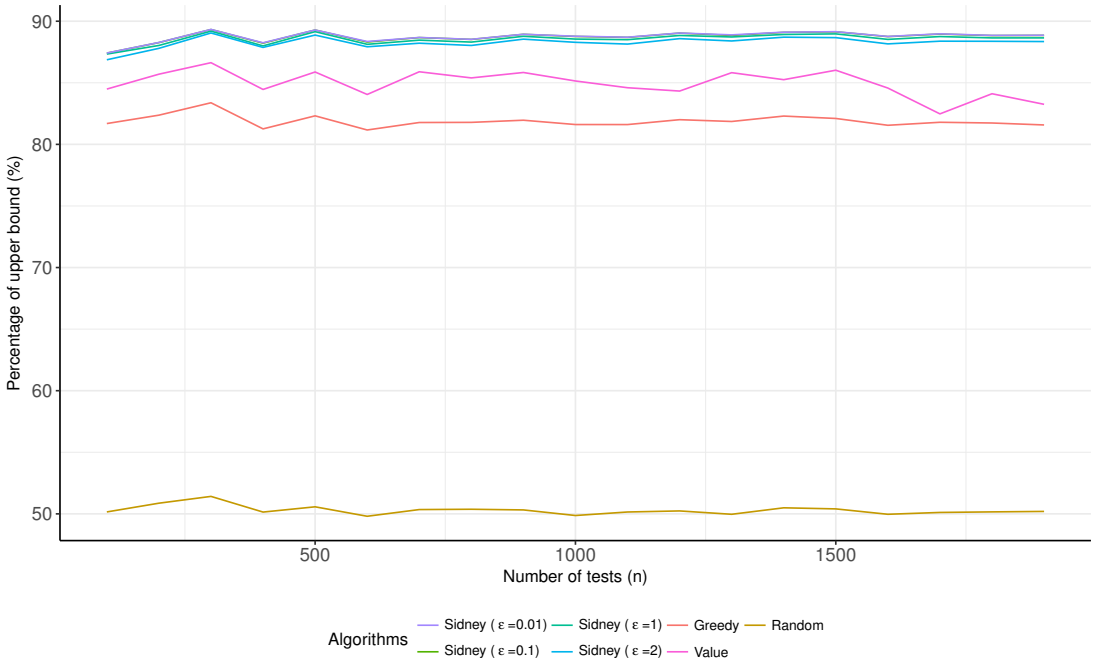

Figure 6.2: Average percentage of the upper bound compared to the size of the test suite

Lastly, Figure 6.3 shows the results when the algorithms were compared and evaluated by their execution time in relation to the number of tests cases n included in the test suites. The set value of ϵ in the Sidney-algorithm seems to have a great impact on the execution time for this particular algorithm. For $\epsilon = 0.01$ the execution time is increasing heavily as the size of the test suites

are extended. The increase in execution time is also larger in comparison to the other algorithms for $\epsilon = 0.1$. For the remaining values of ϵ that were tested, the execution time only increased slightly and notably less than the Value-algorithm when more test cases were added to the test suites.

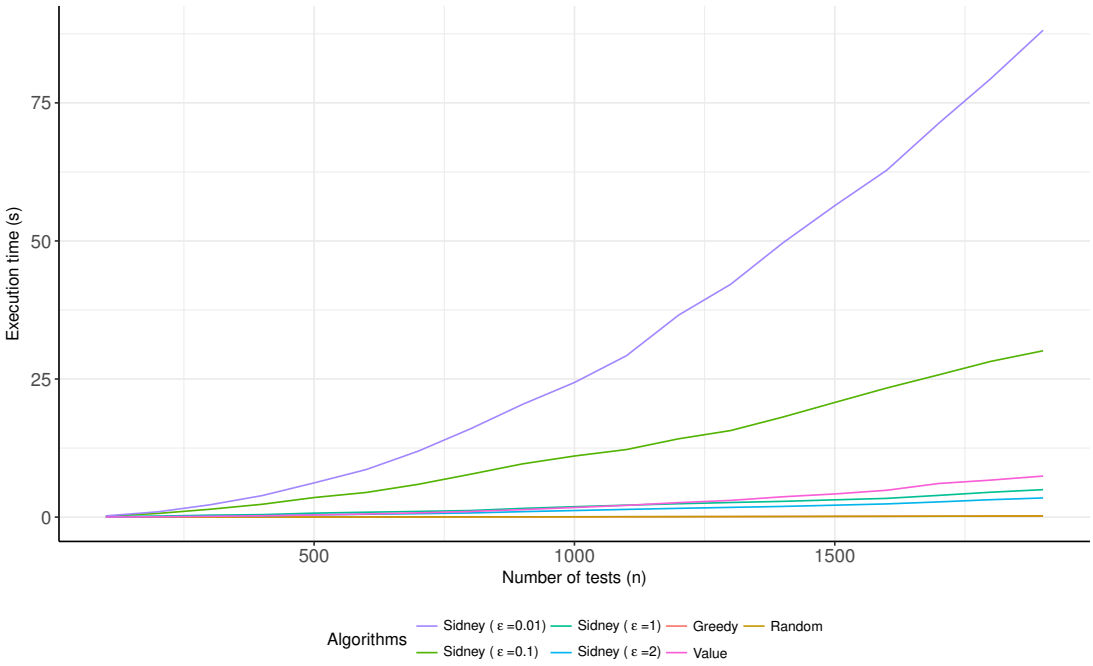

Figure 6.3: Average execution time for the algorithms compared to the size of the test suite

In summary, the Sidney-algorithm generally performed better when the algorithms were compared in relation to the intensity of the test suites as well as the number of test cases included in the test suites. However, the execution time varied for this algorithm depending on the value of ϵ and the number of test cases.

6.2.1 Empirical Evaluation

Presented in this section are the results when the algorithms were evaluated on an authentic test suite at BT. The measure considered as the weights for each test case was the requirement coverage. The number of test cases in the test suite was 1 748.

Figure 6.4 shows the percentage of the upper bound for the different algorithms. Comparing these results with the simulations, one can notice the similarities.

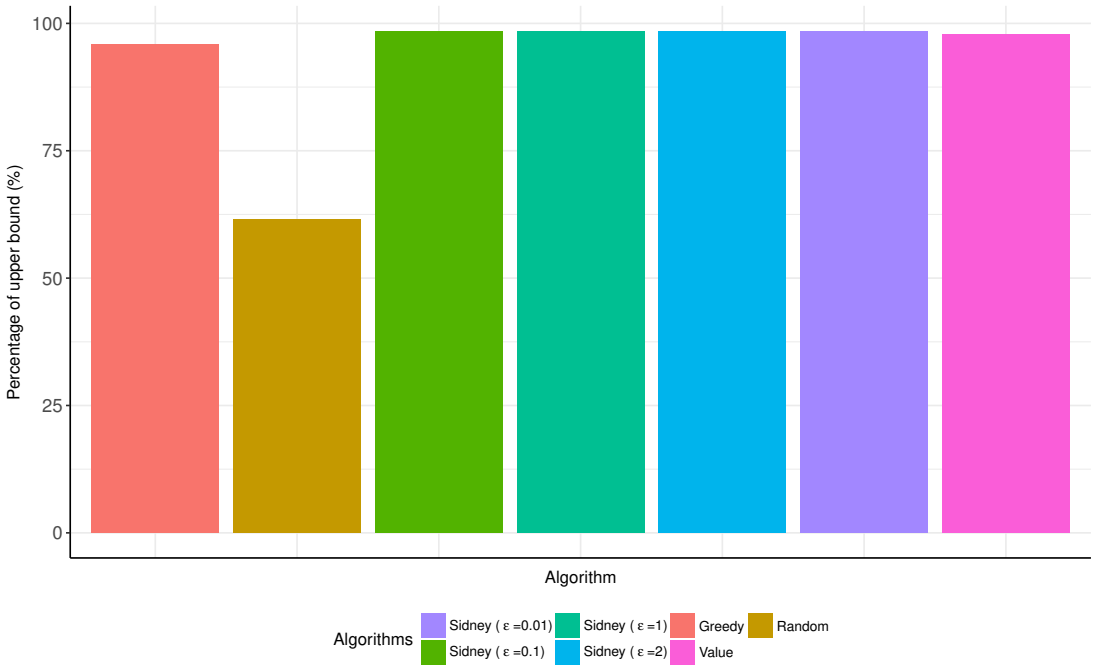

Figure 6.4: Percentage of the upper bound for the algorithms

The Random-algorithm performs worse compared to the others, which is in line with the results from the simulations. The rest of the algorithms are generating schedules close to 100 % of the upper bound, which is a better performance than the schedules obtained from the simulations, where the general performance lay between 80-90 %. The ranking between these algorithms' performances are the same as for the simulation results, namely, the algorithms implemented using Sidney decomposition are best in terms of performance followed by the Value-algorithm and thereafter Greedy-algorithm. The differences in performance are very small, however.

The execution times for the algorithms evaluated on the test suite at BT are presented in Figure 6.5. For the Sidney-algorithm with $\epsilon \in \{0.01, 0.1\}$ the times are significantly larger than the others. The Greedy-algorithm and the Random-algorithm virtually generate schedules instantly, whereas the Value-algorithm as well as the Sidney-algorithm with $\epsilon \in \{1, 2\}$ take a couple of seconds to run.

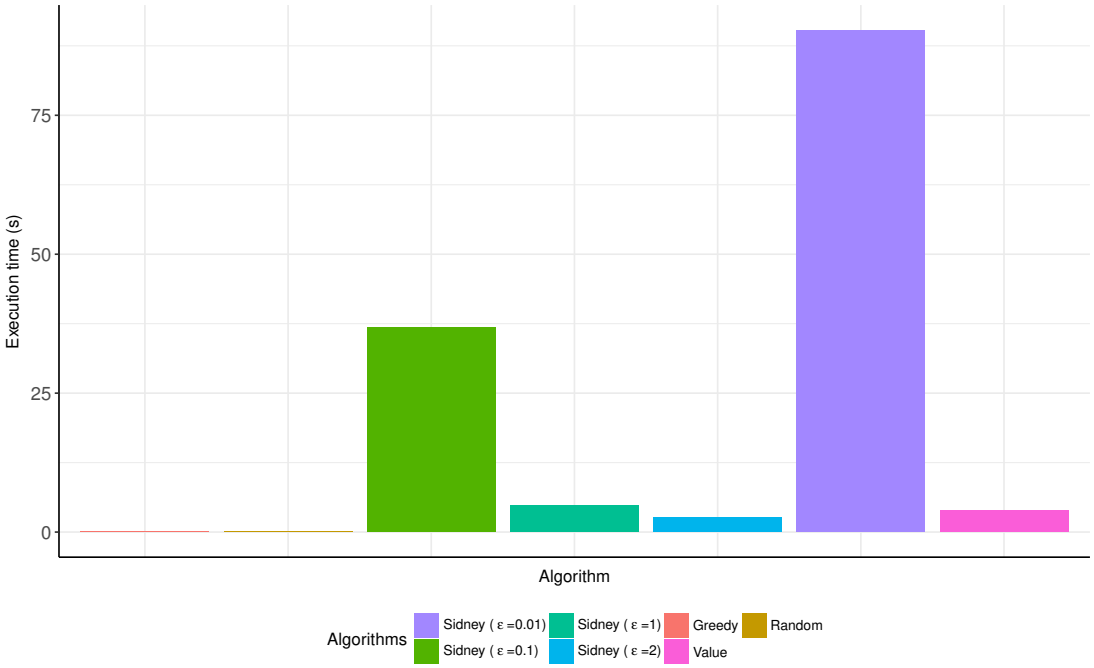

Figure 6.5: Execution time per algorithm

Chapter 7

Discussion

In this chapter, the results obtained in the thesis are discussed. In addition, some limitations of the thesis as well as potential orientation of future research are discussed.

7.1 Results

An important aspect to this thesis is whether it is valuable for a company that are having a lot of test cases to implement any of the proposed algorithms at all. Tahvili et al. (2016b) looked at the return on investment (ROI) for implementing a decision support system for a company where test cases were selected and prioritized based on results from previously executed test cases. In terms of ROI, it was beneficial in that specific case. In analogy to this, the proposed algorithms in this thesis are presumably also worth implementing. An even stronger argument for this is the fact that the proposed algorithms performed significantly better than when the test cases were executed in a random order. Another aspect which lowers the required investment regarding the implementation of such system is that pseudo code for the Sidney-algorithm is presented in this thesis, which would decrease the development effort.

The execution times of the algorithms when evaluated at BT's test suite were relatively short (a couple of seconds), given that the parameter ϵ in the Sidney-algorithm is chosen wisely. Running the algorithm several times a day is hence not a problem. This fact indicates that all algorithms are suitable for being implemented in an industrial setting. However, the ultimate decision for which algorithm to implement is then dependent on the objective value of the produced schedule. Thus, from this thesis, the conclusion would be to use the Sidney-algorithm.

A useful property of the Sidney-algorithm is that the performance of the algorithm can be improved. This can be done either by lowering ϵ, which increases the precision of the cuts or by implementing the algorithm proposed by Chekuri (1998), which can find all the breakpoints of λ. There is also room for improvement when scheduling the generated cuts, where any feasible algorithm performing better than the Greedy-algorithm would increase the performance, for example the Value-algorithm. To implement any of the discussed changes one must consider the trade-off between the increase in execution time of the algorithm and the schedule performance, which can differ between use cases. In an industrial setting this can be useful since the same algorithm can be used for a large variety of test suites with different sizes and structures by adjustment of a single parameter.

Unlike the case study at BT the processing times of the test cases are often unknown or not considered at companies. The mathematical model and the solution algorithms which are presented in this thesis can still be applied though, by setting the processing times of all test cases to the same value. This would yield an approximated schedule but the schedule would probably increase the value of the chosen measure per time unit compared to any other prioritization method which has the same information.

Furthermore, the choice of ρ is affected by the performance of the regression. To find the right selection of covariates have not been the main target of this thesis and hence, better covariates that give a better explanation to the model can possibly exist.

In this thesis, the mathematical model and solution algorithms have all accounted for the fact that the dependencies between test cases cannot be violated. This has mainly been done for lowering the risk of sequential faults (discussed by Zimmerman et al. (2011); Tahvili et al. (2016c)) but also for simplicity reasons. In real life, the dependencies are not that strict and it is potentially practically possible to execute a test case that is dependent on another test case that has not been executed. However, the concept is basically unchanged and if there are test cases that can be executed despite being dependent on other test cases, these can manually be sorted out and marked by a tester before the calculation of a schedule.

Moreover, if the objective in a prioritization problem is to maximize a measure per test case per time unit and there may exists dependencies between test cases the problem can be modeled as a $1 \mid \text{prec} \mid \sum w_j C_j$-problem. The essential insight gained from the result from the algorithm evaluation is that if the $1 \mid \text{prec} \mid \sum w_j C_j$-problem is to be solved, the Sidney-algorithm is the most efficient algorithm for larger problems in both objective value and execution time. This can of course change in the future, if the open scheduling problem, stated by Schuurman and Woeginger (1999), is not true.

7.2 Implementation

In order to use the mathematical model in practice, one needs to investigate what measure that is the most correlated with the APFD metric and also gather the necessary information regarding the test cases for the chosen measure. Another challenging part is to find the dependencies between the test cases. This can be difficult in practice since there is no general method for finding the dependencies. A general method for detection of dependencies between test cases is difficult to design since the documentation about the test cases varies a lot between companies and even between projects within companies. For example, the test cases can be documented in natural language or directly in software code which would require completely different approaches for finding the dependencies.

Once the necessary information and the dependencies have been obtained, the Sidney-algorithm needs to be implemented in a decision support system if the considered testing process is manual. Otherwise, it can be implemented directly in the current testing software. This step is relatively easy compared to gathering the necessary information about the test cases.

7.3 Limitations

The implementations of the algorithms were done without any external expertise. Potentially, the implementations could be improved and consequently give other results in terms of execution times of the algorithms. However, the simplicity of the algorithms indicate no extraordinary improvements could be possible to obtain.

Even though an intense work of finding relevant research for the $1 \mid \text{prec} \mid \sum w_j C_j$-problem was carried out, some research and thus possible solution algorithms may have not been found. If that is the case, then the algorithm evaluation would be incomplete. However, since the Sidney-algorithm is discussed as state-of-the-art in recent research (Ambühl et al., 2011) it is likely that any algorithm which were not found should not have better performance than the Sidney-algorithm.

7.4 Future Research

The Value-algorithm was outperformed in both performance and execution time by the Sidney-algorithm. The performance is heavily affected by the choice of ρ and which implies that the Value-algorithm can possibly be improved by finding a better ρ. One interesting study would be to investigate the upper bound on the performance of the Value-algorithm by calculating the best possible ρ. The

result from such study may show that the Value-algorithm can perform very well in terms of performance. In such case, the Value-algorithm is a good addition to the Sidney-algorithm by finding a good schedule for the cuts produced by the Sidney-algorithm.

One question yet to be answered within the prioritization problem is: what measure or method should be used in order to maximize the APFD-metric when scheduling a test suite? The contribution by this thesis to the subject is that under the assumptions that a schedule which follows the dependency structure of the test suite leads to earlier fault detection (as showed by Zimmerman et al. (2011); Tahvili et al. (2016c)) and that the best measure can be calculated per test case, the most suitable scheduling algorithm is the Sidney-algorithm.

Chapter 8

Conclusion

This thesis has investigated the test case prioritzation problem within software testing. The problem has been modeled mathematically and three different solution algorithms have been evaluated. The evaluation was done through simulations and also via a case study at Bombardier Transportation.

To conclude, it is argued that the prioritization problem can be modeled as a mathematical scheduling problem. The results from the simulations as well as the case study showed that the solution algorithm that is preferable to apply to this problem is the Sidney decomposition algorithm. Moreover, comparing the results with scheduling the test cases arbitrarily indicates that there is definitely a value in implementing the algorithm.

Bibliography

A. A. Acharya, D. P. Mohapatra, and N. Panda. Model based test case prioritization for testing component dependency in cbsd using uml sequence diagram. *Int. J. Adv. Comput. Sci. Appl*, 1(6):108–113, 2010.

C. Ambühl, M. Mastrolilli, N. Mutsanas, and O. Svensson. On the approximability of single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 36(4):653–669, 2011. doi: 10.1287/moor.1110.0512. URL `https://doi.org/10.1287/moor.1110.0512`.

S. H. Ameerjan. Predicting and estimating execution time of manual test cases-a case study in railway domain, 2017.

S. Bates and S. Horwitz. Incremental program testing using program dependence graphs. In *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 384–396. ACM, 1993.

R. N. Burns and G. Steiner. Single machine scheduling with series-parallel precedence constraints. *Operations Research*, 29(6):1195–1207, 1981. ISSN 0030364X, 15265463. URL `http://www.jstor.org/stable/170370`.

P. Caliebe, T. Herpel, and R. German. Dependency-based test case selection and prioritization in embedded systems. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 731–735. IEEE, 2012.

C. Catal. The ten best practices for test case prioritization. In *International Conference on Information and Software Technologies*, pages 452–459. Springer, 2012.

C. Chekuri. Approximation algorithms for scheduling problems. Technical report, Stanford, CA, USA, 1998.

S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28 (2):159–182, 2002.

R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. Johnson, and B. Korte, editors, *Discrete Optimization II*, vol-

ume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979. doi: https://doi.org/10.1016/S0167-5060(08)70356-X. URL http://www.sciencedirect.com/science/article/pii/S016750600870356X.

S. e. Z. Haidry and T. Miller. Using dependency structures for prioritization of functional test suites. *IEEE Transactions on Software Engineering*, 39(2): 258–275, Feb 2013. ISSN 0098-5589. doi: 10.1109/TSE.2012.26.

W. A. Horn. Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal on Applied Mathematics*, 23(2):189–202, 1972. doi: 10.1137/0123021. URL https://doi.org/10.1137/0123021.

A. B. Keha, K. Khowala, and J. W. Fowler. Mixed integer programming formulations for single machine scheduling problems. *Computers and Industrial Engineering*, 56(1):357 – 367, 2009. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2008.06.008. URL http://www.sciencedirect.com/science/article/pii/S0360835208001265.

E. Khandelwal and M. Bhadauria. Various techniques used for prioritization of test cases. *International Journal of Scientific and Research Publications*, 3 (6), 2013.

E. Lawler, M. Queyranne, A. Schulz, and D. Shmoys. Weighted sum of completion times. Available at: http://web.mit.edu/schulz/www/epapers/lqss.pdf, 2006.

Z. Ma and J. Zhao. Test case prioritization based on analysis of program structure. In *2008 15th Asia-Pacific Software Engineering Conference*, pages 471–478, Dec 2008. doi: 10.1109/APSEC.2008.63.

R. Paul. Multicollinearity: Causes, effects and remedies. 05 2018.

C. N. Potts. A lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science*, 31(10):1300–1311, 1985. doi: 10.1287/mnsc.31.10.1300. URL https://doi.org/10.1287/mnsc.31.10.1300.

G. Rothermel and M. J. Harrold. Selecting tests and identifying test coverage requirements for modified software. In *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 169–184. ACM, 1994.

G. Rothermel, R. Untch, C. Chu, and M. Harrold. Prioritizing test cases for regression testing. *Software Engineering, IEEE Transactions on*, 27(10):929–948, October 2001. ISSN 0098-5589.

P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999. doi: 10.1002/(SICI)1099-1425(199909/10)2:5<203::AID-JOS26>3.0.CO;2-5. URL https://onlinelibrary.wiley.com/doi/

abs/10.1002/%28SICI%291099-1425%28199909/10%292%3A5%3C203%3A%3AAID-JOS26%3E3.0.CO%3B2-5.

J. B. Sidney. Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23(2):283–298, 1975. doi: 10.1287/opre.23.2.283. URL https://doi.org/10.1287/opre.23.2.283.

W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956. doi: 10.1002/nav.3800030106. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106.

S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, D. Sundmark, and S. Larsson. Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis. In *Information Technology: New Generations*, pages 745–759. Springer, 2016a.

S. Tahvili, M. Bohlin, M. Saadatmand, S. Larsson, W. Afzal, and D. Sundmark. Cost-benefit analysis of using dependency knowledge at integration testing. In P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen, editors, *Product-Focused Software Process Improvement*, pages 268–284, Cham, 2016b. Springer International Publishing. ISBN 978-3-319-49094-6.

S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin, and D. Sundmark. Dynamic integration test selection based on test case dependencies. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 277–286, April 2016c. doi: 10.1109/ICSTW.2016.14.

S. Tahvili, M. Ahlberg, E. Fornander, W. Afzal, M. Saadatmand, M. Bohlin, and M. Sarabi. Functional dependency detection for integration test cases. In *Companion of the 18th IEEE International Conference on Software Quality, Reliability, and Security.*, July 2018.

S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 22 (2):67–120, 2012. ISSN 0960-0833. doi: 10.1002/stv.430. URL http://dx.doi.org/10.1002/stv.430.

T. Zimmerman, N. Nagappan, K. Herzig, R. Premraj, and L. Williams. An empirical study on the relation between dependency neighborhoods and failures. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 347–356, March 2011. doi: 10.1109/ICST.2011.39.

TRITA -SCI-GRU 2018:251