A GENETIC ALGORITHM FOR THE STOCHASTIC RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

JULIUS FLOHR

HOCHSCHULE DER MEDIEN

Master Thesis
Stuttgart Media University
University of Applied Sciences

exept Software AG
May 2015

SUPERVISORS:
PROF. DR. JOACHIM CHARZINSKI, STUTTGART MEDIA UNIVERSITY
PROF. WALTER KRIHA, STUTTGART MEDIA UNIVERSITY
CLAUS GITTINGER, EXEPT SOFTWARE AG

EIDESSTATTLICHE VERSICHERUNG

Name: Flohr	*Vorname:* Julius
Matrikel-Nr.: 26701	*Studiengang:* Computer Science and Media

Hiermit versichere ich, Julius Flohr, an Eides statt, dass ich die vorliegende Masterarbeit mit dem Titel *"A Genetic Algorithm for the Stochastic Resource Constrained Project Scheduling Problem"* selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden. Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen (§19 Abs. 2 Master-SPO der Hochschule der Medien Stuttgart) sowie die strafrechtlichen Folgen (gem. §156 StGB) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Stuttgart, Mai 2015

Julius Flohr

What I cannot create, I do not understand.
–Richard Feynman

ABSTRACT

Scheduling of tasks is one of the most basic, yet most complicated tasks in any production project. The work at hand introduces a real world scheduling problem encountered at exept software AG for which we are introducing a solution approach using a genetic algorithm. The work gives background information on scheduling under resource constraints with deterministic, as well as stochastic processing times. We adopt and benchmark a genetic algorithm described in literature and achieve an improvement of median processing times of 17% as compared to the algorithm which is currently used to solve the problem. The standard deviation of processing times is reduced by 33%.

ZUSAMMENFASSUNG

Die Erstellung von Zeitplänen (Scheduling) ist eine der grundlegendsten, aber auch schwierigsten Aufgaben in der Planung eines jeden Produktionsprozesses. Die vorliegende Arbeit behandelt ein Scheduling Problem, das bei der exept Software AG auftritt. Dabei stellt die Arbeit die grundlegenden Konzepte des Scheduling unter Ressourcenbeschränkung, mit deterministischen und stochastischen Bearbeitungszeiten, vor. Zur Lösung des Problemes schlagen wir einen genetischen Algorithmus vor, welcher bereits in der Literatur beschrieben wurde. Vergleichsmessungen haben gezeigt, dass der implementierte Algorithmus in der Lage ist den Median der Ausführungs-zeit um 17% zu verkürzen (verglichen mit der bestehenden Referenzimplementierung). Die Standardabweichung ist 33% geringer.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Prof. Dr. Joachim Charzinski, for not only supporting me throughout my thesis with his patience and knowledge, but also for helping me to find my way into science.

Then I would like to thank my girlfriend Diana for the endless support and understanding she showed constantly. Without her, this work would not have been possible.

Further, I want to thank my parents for encouraging me to pursue my dreams and for supporting me morally and financially.

Last but not least, I would like to thank the people at shackspace, especially momo, for lending me the server hardware.

And of course thanks to everyone else who supported me along the way.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

THEORETICAL FOUNDATION

1 INTRODUCTION

Management of projects is a task which has been done for several thousand years in human history. Whether it was the building of the pyramids in Egypt 4500 years ago, the Manhattan project which created the first atomic bomb or the construction of a new automobile: All of those projects involved a large amount of project management. In literature, this term is defined as a process involving management, timing and allocation of resources to achieve a certain well defined goal in an efficient manner [4].

Part of this management project is the task of *project scheduling* where the tasks in the project are laid out in the order in which they have to be performed and which resources to use during the process.

1.1 EXEPT AG SCHEDULING PROJECTS

The work at hand tackles a real world scheduling problem encountered at exept Software AG, a company located in southern Germany that specializes on software and hardware testing. One of their products focuses on the automated planning and execution of such tests. These tests can either ensure that new features are working correctly or that old features are unaffected by recent changes to the system (*regression testing*).

Testing is not a single task which is performed only once in the product life-cycle, but rather a complex continuous process. Also, testing is not free of charge. It always requires some kind of resource: This includes CPU-time, complex testing equipment and human resources. Since these required resources have a monetary cost attached to them, one wants the testing to be done as efficiently as possible. This reduces turnaround times as well as helps saving costs by optimizing resource usage.

The existing algorithm implemented in exept's product is a simple heuristic which has room for improvement. This work tackles this circumstance by introducing the theoretical background of scheduling under resource constraints first and then continues to introduce an improvement of the existing algorithm which is benchmarked thoroughly.

1.2 PROBLEM DESCRIPTION

Before providing a formal definition of the problem at hand, we outline the problem in a non formal way to gain a better understanding of the problem area.

An instance of the exept AG scheduling problem (*EASP*) consists of the following entities:

- a set of *activities* (or *tasks*) which have to be executed
- a set of *resources* which are required for the execution of the activities
- a set of *precedence constraints* which influence the order of activity execution
- an *objective function* which judges the quality of a solution to the EASP.

We assume that these given entities feature the following characteristics:

- each activity may require more than one resource of different types
- processing times of activities are not fixed but stochastic
- resource requirements are fixed for the whole time of execution
- resource availabilities do not change
- resources do not have temporal restrictions
- all resources are renewable (they are not consumed when used)
- activities are atomic: they may not be interrupted
- precedence constraints are also atomic: tasks may not start until their predecessor has been completed
- the objective function does not change during execution

1.3 SCOPE OF THIS WORK

The work at hand tries to find a solution to the EASP as outlined in the previous section. This informal definition makes assumptions about the entities occurring in this problem domain on which we base our work. For now, we limit ourselves to an objective function which optimizes the *makespan* of a schedule. Other objective functions such as: Optimization of machine workload, minimization of cost or strain on human resources are not in the scope of this work.

1.4 OUTLINE

The rest of this work is organized as follows: Chapter two introduces the basics of project scheduling under resource constraints. Chapter three extends these basics to scheduling problems with stochastic processing times. This concludes part one which handles the theoretical background. Part two introduces our solution to the EASP. This includes chapter four which describes the reference algorithm and the dataset used for comparison between the two solutions. Chapter five covers the mathematical models which are used to describe the processing times of tasks. Chapter six introduces the algorithm implemented to solve arbitrary instances of the EASP. An implementation of this algorithm is benchmarked thoroughly and compared to the reference algorithm in chapter seven. Chapter eight concludes this work.

2 DETERMINISTIC RESOURCE CONSTRAINED PROJECT SCHEDULING

Before we can try to find a solution to the EASP we first have to obtain an understanding of the basic concepts of scheduling under resource constraints. In literature, these problems are referred to as *Resource Constrained Project Scheduling Problems* and serve as a foundation for the EASP.

2.1 CLASSIFICATION OF PROJECT SCHEDULING PROBLEMS

Problem scheduling problems come in many different variations which often differ in significant details. To be able to make comparison and referencing of scheduling problems easier, *Graham et al.* [23] introduced a scheme for describing scheduling problems. This scheme is composed of the three fields $\alpha|\beta|\gamma$. The first field α describes the machine environment, i.e. whether the problem is a single machine problem or one of various multi-machine problems (e. g.flow shop, job shop, open shop, etc.). The second field β is used to describe the task and resource characteristics. This includes settings for the possibility of task preemption, precedence constraints, deadlines, ready times etc. The last field γ describes the performance measure (optimality criteria) of the scheduling problem. This can include minimization of project makespan, average flow time, project lateness etc.

Providing all possible values of all fields is not within the scope of this work. For a more detailed description please refer to [15].

2.2 FORMAL DEFINITION

An instance of the deterministic Resource Constrained Project Scheduling Problem (*RCPSP*) (or $m, 1|cpm|C_{max}$ in accordance with [23]) consists of a set of $N = \{0, .., n\}$ tasks with known processing times $d_i \in \mathbb{N}$ for each task $i \in N$. It is possible to enforce precedence constraints between a pair of tasks, e. g.that one task can only be started when other tasks are completed. These constraints are described as an acyclic graph $G(N, A)$ where A is a set of pairs of tasks. We assume that A is a strict order relation on N. Therefore A can be called *precedence relation*. Further, each task $i \in N$ requires a constant amount r_{ik} of each of the renewable resource types k ($k \in K$) during the whole execution time d_i. Each resource type $k \in K$ features an availability a_k during each period of the planning horizon. Further, two dummy tasks are added to N: Task 0 and n are tasks with zero duration and no resource usage. They mark the beginning and end of project and are predecessor, respectively successor to all other tasks in N. Solutions to the RCPS are obtained by denoting the starting times s_i for $i \in N$ in a vector $s = \{s_0, .., s_n\}$. This vector is also called a *schedule*. We can express all tasks which are active during a period $t \in \mathbb{N}_0$ as $\mathcal{A}(s,t) = \{i \in N : s_i \leqslant (t-1) \wedge (s_i + d_i) \geqslant t\}, t$. This definition allows us to formally describe the RCPS as follows [2]:

$$
\begin{aligned}
&\text{minimize } s_n \\
&\quad \text{subject to:} \\
&\qquad s_i + d_i \leqslant s_j \quad \forall (i,j) \in A \\
&\Sigma_{i \in \mathcal{A}(s,t)} r_{ik} \leqslant a_k \quad \forall t \in \mathbb{N}_0 \in K \\
&\qquad\qquad s_i \geqslant 0 \quad \forall i \in N
\end{aligned}
\tag{1}
$$

In this definition, the first constraint set represents the precedence constraints, the second ensures the limit of the number of different resources which can be used at the same time.

2.3 SOLUTION APPROACHES

Resource constrained project scheduling problems occur very frequently in industrial and logistical processes and are therefore researched thoroughly. The latest overview of solution approaches to the problem has been published by *Artigues et al.* [1]. When trying to find solutions to the given problem, one has to make a trade-off between computational time spent on finding a solution and goodness of the solution. Exact solution approaches guarantee to always find the *optimal* solution, but they will soon become unemployable for significant problem sizes. Since the number of publications in the field is so high, we can't directly introduce every single one. But one can identify several solution approaches which differ in minor details from publication to publication. These approaches will be presented in the following sections.

2.3.1 *Exact Solution*

2.3.1.1 *Linear and Integer Programming*

Some works tackle the problem with integer linear programming [33] [15]. Integer programming is a method for solving mathematical optimization problems in which some or are variables are limited to integer values. Further, the objective function is often constrained to be linear [41]. The Problem with these approaches is that the constraints which have to be imposed on the objective function do not fit well on real world scheduling problems [14] and do not scale well [45].

2.3.1.2 *Bounded Enumeration*

The general idea behind bounded enumeration is to create a decision tree generated from the precedence relations in the project plan. Here, the root of the tree corresponds to the first task which has to be executed. All tasks that can be scheduled after the first task has been executed are then inserted as children of the root. The resulting tree is a representation of every precedence feasible schedule for the specific problem. It is easy to see that the tree grows quickly with the number of tasks. Therefore most enumerative approaches use heuristics for pruning the tree [30]. The general goal of the algorithm is to find the optimal root-leaf path which minimizes the execution time [45].

2.3.1.3 *Critical Path Method*

The critical path method (*CPM*) is an algorithm which analyzes projects by determining the longest sequence of tasks that have to be done in consecutive order. However, it does not take temporal or resource constraints into account. It gives the shortest possible makespan assuming infinite resources [45]. But it can give a rough idea of the complexity of a scheduling problem. Several variants of the algorithm have been proposed. This includes stochastic variations [38] as well as dynamic variations [8].

2.3.2 *Heuristic Solutions*

Applicability of the exact solution approaches to real world scheduling problems is limited. Recent research [1] shows that when the problem size reaches approximately 60 activities it is not guaranteed that the optimal solution can be found in acceptable computational time (less than 3 hours of CPU time), but only when the number of resources is rather small ($\leqslant 5$). Because of this instance, lots of work has been done on the field of heuristic solution approaches. The idea is to find heuristic procedures that can produce *good enough* results within acceptable computational times [15]. Generally, heuristic procedures can be divided into two categories: *Constructive heuristics* start with an empty schedule and add tasks to the schedule until one precedence feasible complete schedule is obtained. Their counterpart are *improvement heuristics* which take a complete, feasible schedule as generated by a constructive heuristic as a starting point. The idea is to perform optimizing operations on the schedule and thus create a better solution. This process is repeated until a locally optimal solution is obtained [15]. The possible solutions of a RCSP are divided into several classes $\mathcal{C}$, depending on whether they can be found with a specific scheduling heuristic. For example, a widely used class of scheduling policies are *resource-based (priority) policies* $\mathcal{C}^{RB}$. Please note that the technologies introduced in the following sections are not mutu-

HEURISTIC	WHAT IT DOES
MIN SLK	choose the task with the smallest total slack
MIN LFT	choose the task with the nearest latest finish time
SFM	choose the execution mode with the shortest feasible duration
LRP	choose the execution mode with the least resource proportion

Table 1: Common scheduling heuristics according to [45]

ally exclusive. It is possible to combine them to achieve better results, i. e.[7].

2.3.2.1 *Simple Scheduling Heuristics*

Simple scheduling heuristics operate on a set of tasks and determine at what point in time each task should be executed. In the case of the RCSP this also includes which resources should be assigned to which task. The scheduler enforces constrained satisfaction by assigning a resource to an activity (or vice versa) when the resource is available and the task can be executed [45]. Common scheduling heuristics are enlisted in table 1.

These rather simple approaches were among the first which have been applied to the problem. In the early 70's *Davis et al.* [13] concluded that the MIN SLK algorithm performed best, but only when the number of precedence constraints is not too big.

2.3.2.2 *Simulated Annealing*

Simulated annealing (*SA*) is a meta-heuristic for optimization problems which is capable of finding solutions which are better than simple gradient descent algorithms by introducing random jumps in the solution space when a local minimum is hit. The jump probability decreases over time. Therefore the name is inspired by an analogy to annealing in solids [29].

The first use of this approach in the context of scheduling algorithms has been made by *Van Laarhoven et al.* in 1992 [44]. Throughout the years several algorithms have been proposed which use SA for the creation of schedules [12] [10]. It has been shown that SA algorithms perform better than simple scheduling heuristics [31].

2.3.2.3 *Genetic Algorithms*

An exhaustive comparison by *Kolisch et al.* [31] has shown that genetic algorithms (*GA*) deliver the best results for standard test scenarios. These findings were confirmed by a follow-up investigation of new approaches 7 years later [32]. The basic idea behind genetic algorithms is to mimic the evolution of living creatures. For this approach, the schedule is encoded in an indirect representation on which the GA operates. By performing basic evolutionary operations (crossover and mutation) on these representations, a good solution is found in an iterative process (subject to a cost function). The first application of this approach in the context of scheduling algorithms has been made by *Lawrence & Davis* in 1985 [14]. After that, a large number of publications [25] [39] [21] experimented with different representation forms, variations of the scheduling problem and usage of domain specific knowledge in the scheduling process. A good comparison of the early genetic algorithms has been published by *Bruns* in 1993 [11]. He further notes an inverse relation between the performance of an algorithm and its generality. E.g. that an algorithm which includes domain specific knowledge achieves better results than one optimized for a more general case.

2.4 VISUAL REPRESENTATIONS

In literature there are several forms of visual representation of scheduling problems which have been established as a standard in the field of operational research. This section serves as an introduction to the most common ones.

2.4.1 *Network Analysis*

Most of the time one is interested in the precedence relationship between tasks and the resulting order of execution. The two visual representation schemes presented in this subsection are the two most common ones in scheduling literature.

2.4.1.1 *Activity on the Node (*AoN*)*

The activity on the node format uses the representation introduced in section 2.2. The acyclic graph $G(N, A)$ describes the project network to be scheduled. In this representation, the nodes N are defined as the tasks which have to be carried out. The arcs between the nodes represent the precedence constraints. Figure 1 gives an example of what such a network could look like. The numbers in the circles denote the activity id; the numbers below represent the activity duration. One can clearly see that tasks 0 and 6 are dummy activities with 0 duration. This notation is called the *activity on the node* (*AoN*) notation.

2.4.1.2 *Activity on the Arc (*AoA*)*

Another common representation are *activity on arc diagrams* (*AoA*). Figure 2 depicts the same scheduling network as figure 1, but in the AoA

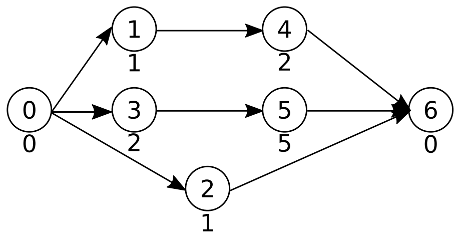

Figure 1: Example:Activity on Node Diagram [3]

notation. Here, the tasks which have to be executed are represented by the arcs (first number on arc represents task id). The nodes mark start and end points of task executions. The numbers in the braces denote the finishing time of the task.

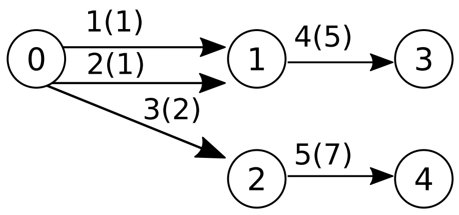

Figure 2: Example:Activity on Arc Diagram [3]

2.4.2 *Ganntt Charts*

One of the oldest visual representation of schedules are Gantt charts [46]. They mark start and finish times of tasks and visualize the duration of each activity as a bar chart. This representation can be enriched by adding additional resource constraint information. Figure 14 in the Appendix gives an example of such a gantt diagram.

2.5 NP-HARDNESS

Section 2.3 has shown that many solution approaches become unfeasible when the problem size reaches sizes which are relevant for the real world.

According to [15], the RCPSP can be generalized to a scheduling problem in which tasks with a unit processing time have to be scheduled on two parallel identical machines and chain-like precedence relations between tasks (e. g.the precedence graph has both indegree and outdegree of at most one for each vertex). The tasks possibly require one unit of a single single renewable resource type with availability 1. The minimization of the makespan in this particular scenario has been proven to be NP-hard by *Blazewicz et al.* [8]. We'll refer to that problem as $P2|res111, chain, p_j = 1|C_{max}$ in accordance with the notation described by *Graham et al.* [23]. It can be shown that the given problem is NP-hard by showing that it can be transformed to an instance of the 3-PARTITION problem which can be described as follows:

3-PARTITION: Given a set $S = 1, ..., 3t$ and positive integers $a_1, ..., a_3, bd$ with $\Sigma_{j \in S} a_j = tb$, can S be partitioned into t disjoint 3-element subsets S_i such that $\Sigma_{j \in S_i} a_j = b(i = 1, ..., t)$?

This celebrated problem was the first number problem that was proven to be NP-complete. A small example is given to illustrate this problem: The set S has a cardinality of 6 and its elements feature corresponding values of 3,3,3,3,4,4 and 5. The values of t and b are 2 ($3 * 2 = 6$ elements) and 11 ($3 + 3 + 3 + 4 + 4 + 5 = 22 = 2 * 11$), respectively. For the instance given, the answer is positive: S_1 would consist of elements 1,2 and 6 with corresponding values of 3,3 and 5. The second set S_2 then consists of the remaining three elements 3,4 and 5 with values 3,4, and 4.

The proof that $P2|res111, chain, p_j = 1|C_{max}$ can be transformed to the 3-PARTITION problem has been taken directly taken from *Blazewicz et al.* [8] and has not been modified by us: Without loss of generality it will be assumed that $\frac{b}{4} < a_j \frac{b}{2}$ for all $j \in S$. This assumption eases the description of the transformation between 3-PARTITION and $P2|res111, chain, p_j = 1|C_{max}$. This can be done as follows:

- Given is a single chain L of $2tb$ jobs:

$$\begin{aligned} L = {} & J'_1 \to J'_2 \to ...J'_b \to J_1 \to J_2 \to ... \to J_b \\ & \to J'_{b+1} \to J'_{b+2} \to ... \to J'_{2b} \to J_{b+1} \to J_{b+2} \\ & \to ... \to J_{2b} \to ... \to J'_{(t-1)b+1} \to J'_{(t-1)b+2} \\ & \to ... \to J'_{tb} \to J_{(t-1)b+1} \to J_{(t-1)b+2} \to ... \to J_{tb} \end{aligned} \tag{2}$$

- For each $j \in S$, there are two chains K_j and K'_j, each of a_j jobs:

$$\begin{aligned} K_j &= J_{j1} \to J_{j2} \to ... \to J_{ja_j}, \\ K'_j &= J'_{j1} \to J'_{j2} \to ... \to J'_{ja_j}; \end{aligned} \tag{3}$$

  moreover, it is required that K_j precedes K'_j, i. e., $J_{ja_j} \to J'_{j1}$.

- The primed jobs do require the resource, the unprimed jobs do not. *Blazewicz et al.* claim that 3-PARTITION has a solution if and only if there exists a feasible schedule with value $C_{max} \leqslant 2tb$.

Suppose that 3-PARTITION has a solution $\{S_1, ..., S_t\}$. A feasible schedule with value $C_{max} = 2tb$ is then obtained as follows: First, the chain L is scheduled on Machine M_1 in the interval $]0, 2tb]$. Note that this leaves the renewable resource available only in the intervals $](2i-1)b, 2ib](i = 1, ..., t)$. For each $i \in \{1, ..., t\}$ it is now possible to schedule the three chains $K_j(j \in S_i)$ on the machine M_2 in the interval $]2(i-1)b, (2i-1)b]$ and the chains $K'_j(j \in S_i)$ on machine M_2 in the interval $](2i-1)b, 2ib]$. The resulting schedule is feasible with respect to resource and precedence constraints and has a total length of $2tb$.

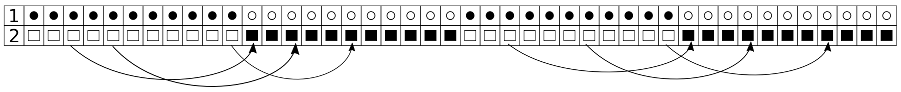
Figure 3: Proof of NP-Hardness according to [8]

This link between a solution of 3-PARTITION and that of a $]2(i-1)b, (2i-1)b]$ as proposed in [8] is depicted in Fig. 3 to ease understanding. All jobs of chain L are scheduled in consecutive order on machine M_1. The Symbol '•' indicates the primed jobs of chain L, which require an additional unit of the renewable resource type. The unprimed jobs of chain L are denoted by the symbol '○'. All jobs in the chains K_j and K'_j are scheduled on Machine M_2. The jobs corresponding with the unprimed chains K_j are represented by the symbol '□', while the jobs of the primed chains K'_j (which also require an additional unit of the renewable resource unit) are denoted by the symbol '■'. The indicated arcs represent the precedence relation $J_{ja_j} \rightarrow J'_{j1}$. Apart from these indicated precedence relations, there exists a precedence relation between all jobs, except J_{ja_j} $(J \in S)$ and $J_t b$, and the next job in the corresponding chain. These precedence relations are not indicated in Figure 3. The solution to the above mentioned 3-PARTITION instance consisted of set S_1 with elements 1, 2 and 6 of value 3, 3 and 5 and of set S_2 with elements 3, 4 and 5 of value 3,4 and 4, respectively. The chains K_1, K_2 and K_6 are scheduled consecutively in the next 11 periods. The second part of the schedule in Fig 3 is similarly constructed, but based on S_2. It should be quite clear that in this way an optimal schedule is given to the corresponding instance of the problem $P2|res111, chain, p_j = 1|C_{max}$. Conversely, suppose that there exists a feasible schedule with value $C_{max} \leqslant 2tb$. It is clear that in this schedule both machines and the resource are saturated until time $2tb$. Moreover, the chains $K_j(j \in S)$ are executed in the intervals $]2(i-1)b, (2i-1)b](i = 1, ..., t)$ and the chains $K'_j(j \in S)$ in the remaining intervals. Let S_i be the index set of chains K_j completed in the interval $]2(i-1)b, (2i-1)b]$, for $i = 1, ..., t$. Consider the set S_1. It is impossible that $\Sigma_{j \in S_t} a_j > b$, due to the definition of S_1; the case $\Sigma_{j \in S_1} a < b$ cannot occur either, since this would lead to machine idle time in $[b, 2b]$. It follows that $\Sigma_{j \in S_1} a_j = b$, and our assumption about the size of $a_j (j \in S)$ implies that $|S_1| = 3$. This argument is easily extended to an inductive proof that $\{S_1, ..., S_t\}$ constitutes a solution to 3-PARTITION.

Now it should be clear that the $P2|res111, chain, p_j = 1|C_{max}$ problem can be transformed into an instance of the RCPSP and that the optimal solution to one of those problems can be transformed to the other problem. As a result, the RCSPSP is clearly NP-hard in the strong sense. $\square$

3 STOCHASTIC RESOURCE CONSTRAINED PROJECT SCHEDULING

In chapter 2 we introduced the deterministic resource constrained project scheduling problem as an introduction to scheduling under resource constraints.

This problem is very similar to an instance of the EASP to which we are trying to find a solution. The difference between those two problems is that the processing times of the single tasks in the RCPS are deterministic. E.g. the duration of all the single tasks is known in advance.

In the case of the EASP this is not true because a test may either succeed or fail at different stages of execution and therefore directly influence the duration of the activity. A scheduling problem which is able to account for this circumstance is the stochastic RCPS which will be described in detail in this chapter.

3.1 DEALING WITH UNCERTAINTY

When scheduling several tasks, one has to know in advance how long a certain task will take. Especially when performing *unique* projects where there is no historic data which could be used as reference. The naive approach to handling uncertainty of processing times is to just use *average* activity-durations as single-time estimates. But this completely ignores the chance that some tasks may deviate from the average. For example, an activity is expected to take 10 days processing time and might vary from 9 to 11 days. Another activity is also expected to take 10 days, but might vary from 3 to 17 days. This example illustrates why single-time estimates fail to represent the characteristics of an activity with uncertain task durations. Since projects in the real life are seldom purely deterministic, the need for more adequate handling of stochastic processing times arose. *PERT* (Project Evaluation and Review Technique) is a statistical tool introduced in the 1950s which was designed to cope with this issue [15].

Malcolm et al. [35] proposed to use not one, but three times for estimating the duration of a task (optimistic, most likely and pessimistic). They then modeled every activity duration as a random variable which follows a beta distribution and proposed simple method for approximating the expectation and variance of the network event times. With the advent of fuzzy sets [48], this stochastic approach was quickly rejected and replaced by *fuzzy* models [15]. These models still play an important role when there is no historic information of previous activity durations available. When the scheduling task is not unique and there is historic data available, it can be used to create much more accurate models [15].

3.2 FORMAL DEFINITION

The stochastic resource constrained project scheduling problem (*SR-CPSP*) (or $m, 1|cpm, d_j|E(C_{max}])$ is quite similar to its deterministic counterpart. Here, the duration D_i of each activity $i \in N$ is not a fixed value but rather a random variable (see eq. 1). The resulting random vector $(D_0, D_1, ..., D_n)$ is defined as D. Further, we define that the duration of the dummy activities still is zero: $i = 0, n$, $P[D_i = 0] = 1$ (with $P(e)$ being the probability of event e). While the duration of the remaining activities is always greater than zero: $i \in N \backslash \{0, n\}$, $P(D_i < 0) = 0$. The solution of an instance of the SRCPSP is no longer a schedule, but rather a *scheduling policy* which is gradually produces the schedule as time progresses. E.g. the problem can be seen as a multi-stage stochastic decision process [20] [15]: A decision rule determines which task to execute at certain decision points in time. These decision times are at time 0 (project start time) and the completion times of tasks. It is important that the decision made at point t can only use information which has become available before or at time t. Therefore, the schedule is constructed gradually as time progresses. A realization of D is denoted as d. The decision rule can also be interpreted as a function [42]: The policy Π is a function $\mathbb{R}^{n+1}_{\geqslant} \mapsto \mathbb{R}^{n+1}_{\geqslant}$ which maps given samples of activity durations d to schedules $s = \Pi(d) \in \mathbb{R}^{n+1}$. The makespan of a given scenario d and the policy Π is denoted as $[\Pi(d)]_n$. The bracket notation $[\cdot]_i$ represents the $(i + 1)$-th component of the vector between the brackets, since in our case the indexing starts from 0. Thus, the the project completion time $[\Pi(D)]_n$ is a random variable and the objective of the SRCPS can be be defined as: Find a policy Π^* which minimizes $E[[\Pi(D)]_n]$ within a specific class $\mathcal{C}$(with $E[\cdot]$ being the expectation operator with respect to D).

3.3 SOLUTION APPROACHES

As described in section 3.2 the solution of an instance of the stochastic resource constrained project scheduling problem (*SRCSP*) is no schedule, but rather a scheduling policy. Therefore, this section will introduce different scheduling policies which have been proposed in literature. All approaches introduced in this section are heuristic solutions which cannot guarantee to deliver an optimal solution.

3.3.1 *Resource Based (RB) Policies*

One of the simplest scheduling policies is that of Resource-Based (*RB*) policies. An RB Policy can be seen as a function $\Pi : A \mapsto \mathbb{N}$ which assigns priorities to activities. At each decision point t, iterates over the priority list and starts as many activities as possible (respecting precedence and resource constraints) [36]. It has been shown that there are problem instances for which this approach yields no optimal solution [15]. Besides, these policies have the drawback that so called *Graham Anomalies* may occur: Several actions which would expectably decrease the makespan, can lead to an increase. These actions are: Decreasing the activity durations, adding additional resources and also removing precedence constraints [22].

3.3.2 *Early Start (ES) Policies*

ES policies were first proposed by *Radermacher* in 1985 [40]. The scheduling policy in this case is a set of *minimal forbidden sets* (*mfs*). A minimal forbidden set (i, j) is an ordered pair of activities $\{i, j\} \in (N \times N) \backslash A$ which should not be executed at the same time because of resource constraints and are not precedence related. An instance of the ES policy takes a set of mfs X as parameter. The idea is to "break" minimal forbidden sets by introducing new precedence relations such that the partial order set A extends to $A \cup X$. In the ideal case this allows us to ignore the resource constraints [15] [3]. Computational results for the SRCSPSP are scarce. Stork [42] proposes to use branch-and-bound algorithms for finding the optimal solution for the shortest makespan. However, this policy class becomes unfeasible when the number of tasks becomes too big (approximately 50 tasks or more) [3].

3.3.3 *Activity Based (AB) Policies*

First introduced by *Stork* [42], AB policies also assign priorities to all activities in N. But, in contrast to the RB policies, there is another restriction in place: AB policies do not start a activity as long as there is another activity present with a greater priority. Also, this policy does not suffer from graham anomalies [36].

Part II

IMPLEMENTATION AND RESULTS

4 REFERENCE ALGORITHM AND BENCHMARK DATASET OVERVIEW

4.1 THE REFERENCE ALGORITHM

The existing algorithm interprets the problem as a multi-stage stochastic decision process and schedules the activities by means of a resource based scheduling policy. The software offers the possibility to assign coarsely grained priorities to the activities which influences the ordering of the tasks in the priority list. However, the software does not influence the ordering of the activity list in order to optimize the makespan.

Handling of different resources in reality is more difficult than described in the formal definition in chapter 2. The problem is that resources are possibly not only member of one device class $k \in K$, but can be member of several device classes at the same time $\{k_1, k_2, ..., k_n\} \in K$. In the scenario of hardware testing this happens relatively often: imagine a multimeter which is not only capable of measuring currents but also voltage. This is even more true when we consider the precision tolerances of a test device: imagine we have explicit device classes for voltmeters which can measure with a tolerance 1, 5 and 10 percent. Every device which is in the 1 percent device class is therefore automatically also in the other two. When there are several devices which would be capable of carrying out an activity, the reference implementation chooses the device with the least total amount of capabilities: minimize $|\{k_1, k_2, .., k_n\}|, k \in K$. This heuristic has been adopted in our implementation as well, since the algorithm proposed does only consider resource availability during scheduling.

4.2 DATA OVERVIEW

For reasons of confidentiality, we unfortunately are not able to publish detailed information about the data set used to benchmark the algorithms since it is taken directly from a customer of exept. However, we can give an overview over the broad structure.

The data was provided as a very large number of XML files which reference each other with unique identifiers. In there we identified 44 different activities. Each activity stores the information which resources it needs, what tasks need to be executed, how often it has been run and the mean and variance of the duration of the execution. Furthermore we identified files which are historical records of test execution. These contain a reference to the test, the resources assigned to the tasks and the actual execution times. Ideally, the number of executions reported by the test file would be identical with the number of historical records. Unfortunately, this is not the case because the product automatically deletes historical records which are older than 30 days. As can be seen in Fig. 4, the number of observed events is rather small compared to the claimed number of executions. This means that that the values reported for variance and mean can be different from those which can be actually verified by historical data.

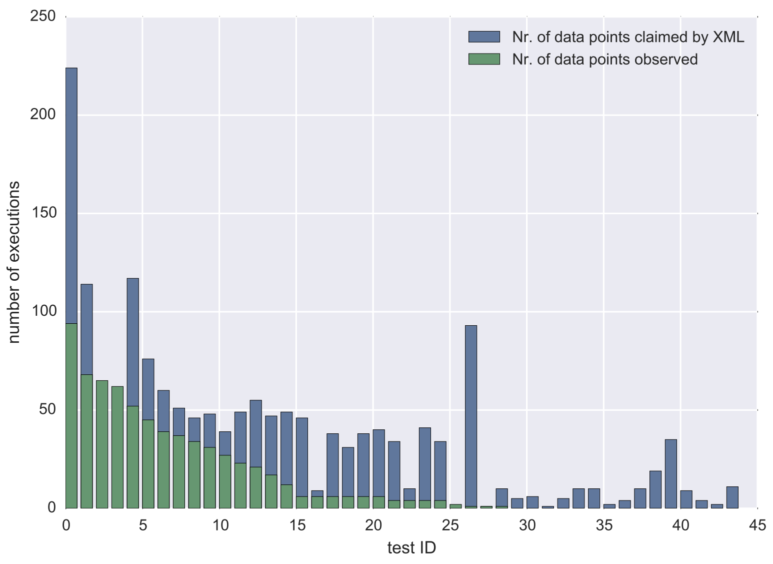

Figure 4: Number of executions reported vs. seen

Please note the special cases of IDs 2 & 3 in Fig. 4, where the number of data points observed is higher than the number of executions claimed by the XML file. This is probably due to an incident where the customer had to restore files from a backup and thus creating a inconsistent state. Overall, the dataset contains 421 different resources within 103 resource classes. Out of the 44 tasks 4 require more than 15 resources and 38 more than 10. The maximum number of resources required by a single task is 18; the minimum is 1. The dataset does not contain any precedence constraints between tasks.

5 MODELING OF EXECUTION TIMES

In order to be able to make statistically significant claims about the goodness of a certain scheduling rule, the algorithm has to be tested with a very large number of different scenarios. Unfortunately, in our case the number of historically available data which could be used for simulation is rather small. Therefore we resort to the creation of a mathematical model from the few data points we have available. This data is later used in a simulation environment to create an arbitrarily, large set of scenarios for the verification of the scheduling algorithms examined.

For the creation of a infinitely large set of test scenarios we have to approximate the processing times d_i for all task $i \in N$. As described in the previous section, the data available is not very consistent and therefore we had to make a few assumptions which will be described here in detail.

5.1 CASE 1: ENOUGH HISTORICAL DATA AVAILABLE

In the trivial case we can approximate the execution Time d_i by performing inverse transform sampling on the given data. Please note that this is only done for cases where we have at least 15 data points available. Inverse transform sampling is done as follows [16]:

1. Let X be the random variable of which the distribution function should be sampled. Its cumulative distribution function F can be approximated using the historic data available.
2. Draw a random number u from the uniform distribution on the interval [0,1]
3. Compute the value x such that $F(x) = u$
4. Take x as the random number drawn from F

5.2 CASE 2: NOT ENOUGH HISTORICAL DATA AVAILABLE

In many cases we didn't have enough historical records to use the method described in section 5.1. In these cases we have to rely on the values for mean and standard deviation which are reported by the tasks themselves. We used them to perform a two moment fit of phase type distributions to model the execution times [18]. This approach often finds application when modeling the waiting time of services in queuing theory [9], and we assume that it is applicable in our case as well since there is no literature on service time of hardware and software tests available. The kind of phase type distribution to use for the two moment fit is determined by the coefficient of variation of the random variable X:

$$c_X = \frac{\mu_X}{\sigma_X} \tag{4}$$

with μ and σ being the mean and standard deviation of X. In total, we used three different phase type distributions, depending on the domain of c_X^2 [34]:

$c_X^2 \geqslant 1$: For large values of c_X^2 the distribution is approximated with a two phase hyperexponential distribution with *balanced means*. Its density function is given by [43]:

$$\begin{aligned} &f(x) = p\lambda_1 e^{-\lambda_1 x} + (1-p)\lambda_2 e^{-\lambda_2 x} \\ &\text{with:} \\ &p = \frac{1}{2}\Big(1 + \sqrt{\frac{c_X^2 - 1}{c_X^2 + 1}}\Big), \\ &\lambda_1 = \frac{2p}{\mu}, \\ &\lambda_2 = \frac{2(1-p)}{\mu} \end{aligned} \tag{5}$$

By means of integration, we can obtain the distribution function of Eq. 5:

$$F(x) = \int f(x)dx = 1 - pe^{-\lambda_1 x} + (1-p)e^{-\lambda_2 x} \tag{6}$$

This distribution function can then again be sampled with inverse transform sampling.

$0.5 < c_X^2 < 1$: We approximate c_X^2 values between 0.5 and 1.0 with a Erlangian $E_{k-1,k}$ distribution, which is defined as a mixture of E_{k-1} and E_k distributions [43]. Its probability density function is given by:

$$f(x) = p\mu^{k-1}\frac{x^{k-2}}{(k-2)!}e^{-\mu x} + (1-p)\mu^k\frac{x^{k-1}}{(k-1)!}e^{-\mu x}, x \geqslant 0 \tag{7}$$

The cumulative distribution function (*CDF*) was obtained by integration:

$$\begin{aligned}
&F(t) = \frac{k(-p+1)\Gamma(k)\gamma(k,\mu t)}{(k-1)!\Gamma(k+1)} \\
&+ \frac{\mu p}{(k-2)!}\Big(\frac{k\Gamma(k-1)}{\mu\Gamma(k)}\gamma(k-1,\mu t) - \frac{\Gamma(k-1)}{\mu\Gamma(k)}\gamma(k-1,\mu t)\Big) \\
&\text{with:} \\
&\gamma(s,x) = \int_0^x t^{s-1}e^{-t}dt \\
&\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx
\end{aligned} \tag{8}$$

with Γ being the gamma function and γ the lower incomplete gamma function.

$c_X^2 < 0.5$: Small coefficients of variation are approximated with a uniform distribution with the probability density function:

$$\begin{aligned}
&f(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b]\ , \\ 0 & \text{otherwise} \end{cases} \\
&\text{with:} \\
&a = 0, b = 2\mu
\end{aligned} \tag{9}$$

TRUNCATION OF DISTRIBUTIONS The three different ranges of the coefficient of variation are not the only criterion used to decide which distribution function to use. Additionally, the XML data model offers the possibility to specify an upper limit for the execution time after which a still running task is automatically eliminated. This truncation has to be included into the mathematical model in order to be correct. The truncation is given by [19]:

$$F_{a,b}(x) = \begin{cases} 0 & \text{for } x \leqslant a,, \\ \frac{F(x)-F(a)}{F(b)-F(a)} & \text{for } a < x \leqslant b, \\ 1 & x > b, a < b. \end{cases} \tag{10}$$

With F being the distribution function as defined in the previous section, $a = 0$ and b as defined in the XML model. Therefore, the model employed differentiates between six different cases.

6 A GENETIC ALGORITHM FOR THE EXEPT SCHEDULING PROBLEM

Chapter 2 has shown that even the deterministic version of the resource constrained project scheduling problem is NP-hard in the strong sense. All known solution approaches to the SRCPSP are of heuristic nature. We thus have to face the fact that the EASP is also NP-Hard and that we cannot provide an algorithm which is able to generate an optimal solution to a real world sized instance of the EASP with manageable computational effort. *Ashtiani et al.* [2] manage to combine promising approaches to the SRCPSP to create a new class of scheduling policies which fits well to the use case of the EASP. In their approach, the computationally expensive parts can be *precomputed* and reused. This fits the EASP since regression tests are run periodically and do not change very often. Another important factor is, that the algorithm proposed also incorporates an RB-Policy which has already been implemented in the software of exept AG and therefore the whole approach is easier to integrate. The overall approach has been adopted, however the algorithm differs in details. Whenever we deviate from the original approach, it is clearly marked in the text.

6.1 THEORETICAL BACKGROUND: PREPROCESSOR POLICIES

As described in chapter 3 the solution of a RCPSP is a scheduling policy which determines the next activity to execute at every point in time of a multi-stage decision process. Several of these policies were introduced in section 3.3. The work of *Ashtiani et al.* introduced a new class of scheduling algorithms which they call pre-processor policies or *PP policies*. In their approach they combine $\mathcal{C}^{ES}$ and $\mathcal{C}^{RB}$ to gain a new approach for project scheduling. *Wu et al.* [47] argue that the questions of "when and how to make which decisions" are among the most important factors for planning a schedule when the execution of tasks may be interrupted by unforeseen activities. This means that the scheduling decisions made at the beginning of the planning horizon have utmost importance for the overall schedule performance. Following that insight, the class of pre-processor policies identifies a critical subset of scheduling decisions which are crucial good performance and relegates the remaining scheduling decisions to future points in time [3]. In the introductory part of this work, we explained that customers who face the EASP often run regression tests, e. g.the same set of activities is run periodically. Thus, every time the same set of activities is executed, one can take advantage of the (computationally expensive) pre-processing.

6.1.1 *PP policies: A combination of ES- and RB-policies*

As described in section 3.3.2, ES policies try to resolve *all* minimum forbidden sets before the actual execution starts, leaving a scheduling problem with no possible resource conflicts. The new class $\mathcal{C}^{PP}$ incorporates a combination of the real-time dispatching capabilities of $\mathcal{C}^{RB}$ with a *partial* solution of $\mathcal{C}^{ES}$. An instance of the PP-policy $\Pi \in \mathcal{C}^{PP}$ is defined by an activity list L together with a set of activity pairs $X \subset N \times N$. The resulting policy Π is only feasible if the precedence graph $G(N, A \cup X)$ is acyclic. With X being the partial solution of an instance of $\mathcal{C}^{ES}$, Π is now able to resolve *some* of the resource conflicts which could be encountered. The remaining resource conflicts are resolved with the help of the RB policy and the activity list L as its parameter.

6.1.2 *Combination of ES- and RB-Policies*

During this work we have outlined that RB policies suffer from shortcomings such as Graham anomalies. Therefore, one might be intrigued to combine $\mathcal{C}^{ES}$ and $\mathcal{C}^{AB}$ instead of $\mathcal{C}^{ES}$ and $\mathcal{C}^{RB}$. In their work, *Ashtiani et al.* conjecture that the stability issues are irrelevant for practical decision making when the the expected makespan is appropriately low. Further, they argue that their computational results deliver better results in the *average* case when using solutions from $\mathcal{C}^{RB}$ instead of $\mathcal{C}^{AB}$. These findings were confirmed by our own experiments.

6.2 GLOBAL STRUCTURE OF THE ALGORITHM

For finding good policy parameters L and X the implementation of the algorithm makes heavy use of genetic algorithms (*GA*). First proposed by *Holland* [26] as a heuristic approach for tackling optimization problems. As described in section 2.3.2.3 these algorithms have been successfully employed in the scheduling environment for several years. The algorithm proposed by *Ashtiani et al.* is a two-phase genetic algorithm. The overall structure of the algorithm we derived from the work of *Ashtiani et al.* is elucidated in listing 1. Phase one consists of a genetic algorithm (**ListGA**) which is responsible for finding a good order list L. The result of the first phase is then passed to the second algorithm which creates an initial list of activity pairs X which is then optimized using the second genetic algorithm (**ArcGA**). Our algorithm deviates in one point from the reference: *Ashtiani et al.* proposed that **ListGA** should not return a *single best* solution, but rather the n best which are then separately passed into the second

Listing 1: Global Algorithm Overview

```
        L = ListGA()
        X = ArcGA(L)
        return L, X
```

phase. We opted to deviate from this solution because we found that the n best solutions are often very similar (in terms of activity ordering) and therefore large deviations in goodness between the n activity lists are rather uncommon. This change dramatically decreases the computational effort because **ArcGA** is executed only once and not n times.

6.3 PHASE I: LISTGA

This section gives a detailed overview of the **ListGA** algorithm. As outlined in previous sections of this work, genetic algorithms are a search heuristic which mimics evolutionary processes. Often these algorithms are composed out of the following components:

- **Individuals**: To be able to perform genetic operations one needs an entity to which these can be applied to. Often these individuals are string or list representations of a possible solution to the problem the algorithm is trying to solve.
- **Evolutionary Operations**: These operations are applied to the single individuals over the course of several generations in order to select beneficial solutions and further evolve them. Evolutionary Operators which are commonly applied are: selection, mutation, crossover and inheritance.
- **Fitness function**: After the evolutionary operations have been applied to an individual, one needs to decide whether the modification was beneficial or not. This is done using a fitness function. It takes a single individual as parameter and computes a numerical value which represents how good or bad the individual performed.

6.3.1 *Algorithm Discussion*

These components are also present in the implementation of **ListGA**. Specifically, the following section will describe every part of the algorithm implemented in detail. An overview of the algorithm is given in listing 2.

INDIVIDUALS Every individual in the algorithm is a precedence-feasible list of activities which should be executed.

Listing 2: Details of ListGA Algorithm

```
1        bestList = createInitialPopulation()
2        while not TerminationCriterion do:
3                population = ListCrossMut(bestList)
4                population.append(bestList)
5
6                for L in population:
7                        compute fitness with $\Pi(D;L)$
8                end for
9                bestList = select $n_{list}$ best inidivduals from
                     population
10       end while
11       return best from bestList
```

INITIAL POPULATION Before the genetic algorithm can become operational, it needs an initial population to which the genetic operations can be applied to (see Listing 2, line 1). Here, regret-based biased random sampling (RBRS) [17] is employed. The idea is to select an activity j out of the set E of precedence and resource eligible activities. This is done with probability π_j, which is determined by the average duration of the task j compared to the total execution time of all tasks in E:

$$\begin{aligned} \pi_j &= \frac{(p_j+1)^\alpha}{\Sigma_{k\in E}(p_k+1)^\alpha} \\ \text{with: } p_j &= max_{k\in E} \nu(k) - \nu(j) \end{aligned} \tag{11}$$

α has been fixed to 1, ν to the average execution time.

CROSSOVER The first genetic operator applied to an individual is the crossover operator. With the probability p_{cx} a two-point crossover [25] operator is applied. This operator combines two lists into a pair of new lists. This is done as follows:

- Select two individuals as parents (father and mother)
- Draw two random integers r_1 and r_2 with $1 \leqslant r_1 \leqslant r_2 \leqslant n$
- construct the son (daughter) by copying the first r_1 positions from the father (mother)
- copy positions r_1 to r_2 from the father (mother) to the son (daughter)
- finally, copy the remaining positions from the mother (father)

MUTATION The second genetic operator applied is a standard mutation operator [25]. For a given individual $L = (l_0, l_1, ..., l_m)$ it changes the ordering of the activities at position $i = 1, .., n-2$ as follows: Exchange activities l_i and l_{i+1} wit probability p_{mut}.

FITNESS FUNCTION The fitness of each individual is obtained via simulation using a RB-Policy which tries to schedule as many eligible tasks as possible at each decision point. At this point *Ashtiani et al.* employ the makespan $[\Pi(E[D], L)]_n$ with expected processing times as the fitness of the individual L. However, it has been shown that when either the number of tasks is large, or the tasks feature a great variability, it is more beneficial to not use expectational values for processing times during simulation [5]. Our experiments have shown that this seems to be true for the EASP. Instead of using the mean execution time of each task during simulation we are sampling the probability distributions introduced in chapter 5. Doing so alone introduced a new problem with the standard implementation: The plain algorithm by *Ashtiani et al.* computed the fitness of each individual only once. So it was possible that a badly performing individual was assigned a good fitness, because the fitness does not only depend on the performance of the individual but also on the realization of the vector D. To circumvent this, we changed the algorithm so that selection of best individuals is benchmarked together with the current population (see listing 2, line 4) every iteration of the algorithm. This increases the number of simulations required, but ensures that only individuals which deliver a good overall performance independent of the realization of D stay in the list of best candidates.

SELECTION After the fitness function has been applied to all individuals in the population (this includes the best n_{list} from the last generation and their offspring). It has been shown [25] that simple *"select best"* operators outperform more complex selection algorithms (e. g.tournament, proportional selection). The trivial select best operator selects the n_{list} best individuals and discards the rest.

6.4 PHASE II: ARCGA

The second phase of the algorithm consists of the Arc Genetic Algorithm (*ArcGA*) which tries to break some MFSS by adding new precedence constraints to the problem.

The ArcGA features the same building blocks as the ListGA. The only difference here lies in the implementation of the different parts of the algorithm. The overall algorithm structure is shown in listing 3.

Listing 3: Details of ArcGA Algorithm

```
bestList = createInitialPopulation()
while not TerminationCriterion do:
        population = TupleCrossMut(bestList)
        population.append(bestList)

        for X in population:
                compute fitness with $\Pi(D_i;L;X) - \Pi(D_i;L;\emptyset)$
        end for
        bestList = select $n_{list}$ best inidivduals from
            population
end while
return best from bestList
```

INDIVIDUALS In case of the ArcGa, every individual $X = x_1, .., x_m$ is an unordered set of ordered activity pairs $x_i = (j, k)$ with $i = 1, ..., m : x_i \notin A$. Each of these pairs describe a precedence relation, e.g. the first activity of the pair has to be executed before the second.

INITIAL POPULATION In the original implementation of *Ashtiani et al.* the initial population is constructed by randomly choosing n_{pairs} activity pairs from the list of all tasks. In our implementation, we tried to improve this simple scheme by using domain knowledge: The data model developed by exept distinguishes between two different types of resource requirements: The first type only specifies certain characteristics which have to be met by the resource which should be used for execution. The second type is fixed assignment of a certain device to the task. This can be seen as a special resource type $k_i \in K$ with an availability of 1. The idea is to gather the set of activities B which specifically require the resource type k_i. Therefore, the activities in B form a mfss since they would introduced a resource conflict when scheduled together. Then, this mfss is broken up by randomly selecting two activities $(a_i, a_j) \in B$ as part of the initial population. This set of precedence constraints is only feasible if the resulting graph $G(N, A \cup X)$ is acyclic.

CROSSOVER The crossover operation used has been proposed by *Ashtiani et al.*. The *uniform crossover* works as follows: Subdivide the current population into pairs of parents (mother X_M & father X_F). With the probability p_{corss}, assign each $x_i \in X_F$ to a son individual X_S, otherwise assign it to the daughter X_D. All elements $x_i \in X_M$ are assigned to X_S and X_D analogously.

MUTATION The activity pairs of each individual X are affected by the mutation operator with probability p_{mut}. Let C be the set of candidate pairs for the first generation. Each $x \in X$ is removed from X with probability p_X. Similarly, a new activity pair $x \in C \setminus X$ is added with probability $1 - p_X$.

FITNESS FUNCTION The reference implementation of *Ashtiani et al.* proposes to use a RB-policy with expected durations and the condition $[\Pi(E[D], L, X)]_n < [\Pi(E[D], L, \emptyset]_n$ to test for makespan improvement. We again chose to not depend on expected processing times. We implemented our simulator in such manner, that it is possible to use one manifestation D_i of D for several simulations. Therefore we are able to compute $[[\Pi(D_i, L, \emptyset)]_n - \Pi(D_i, L, X)]_n$. This term directly indicates how much time was saved by applying the precedence constraints X. In order to compensate for the impact of different realizations of D_i, the same method of reevaluating every individual (best and population) is applied.

SELECTION Selection of the most promising individuals is also done with the simple select best operator.

7 EXPERIMENTAL RESULTS

After the algorithms have been implemented as introduced in the last chapter, the impact of each parameter on the performance of the algorithms has been research thoroughly.

7.1 EXPERIMENTAL SETUP & IMPLEMENTATION DETAILS

In order to be able to handle the large number of simulations required to be able to make statistically significant claims about the quality of a parameter set, two computers were used to conduct the experiments. One was a personal computer with four processor cores operating at 4200 MHz and 16GB of RAM. The other was a server equipped with two 8-core processors operating at 2900 MHz and 32GB of RAM. Implementation of the algorithm was done in python, mainly using the scientific python ecosystem [27].

We modeled the execution of the tasks with help of the discrete event simulation [6] framework SimPy [37]. In those systems the operation of systems is modeled as a discrete sequence of events. This means that the state of the the whole system can only change at certain points during the simulation. In our case those points are the beginning (allocation of resources) and termination of a task (release of allocated resources). Therefore, the simulated time can directly jump from one event to the next. The duration of each task is determined at the task start time with the help of the mathematical model provided in chapter 5.

All parts of the software which do not expose intellectual property of exept software AG will be made public[1].

7.2 NOTE ON COMPUTATIONAL TIME

In order to account for different computing environments we restrain from providing exact computational times for each simulation in this chapter. We rather user the accepted [25] method of providing the number of schedules which have been generated during the creation of a scheduling policy. This should help to establish a comparability between different scheduling policies. As a point of reference: The personal computer specified in section 7.1 is capable of computing 120.000 schedules per hour.

1 https://github.com/juliusf/Genetic-SRCPSP

7.3 META-OPTIMIZATION OF ALGORITHM

In order to achieve the best possible performance, genetic algorithm parameters have to be tuned correctly. There are several meta-optimization approaches [24] [28] which can be used to automatically find an optimal or near optimal parameter set. Employing these techniques can be very time consuming since a large portion of the parameter space has to be covered. Therefore, we opted for manually optimizing each parameter iteratively. To be able to make statistically relevant claims about the quality of a parameter set, each set was used to create 200 different policies. Each of those policies was then used to create one schedule of which the makespan was recorded.

7.3.1 *Computational Results of ListGA*

The ListGA as depicted in Figure 2 has several parameters which can be optimized:

- Termination criterion; in our case: number of generations: n_{gen}
- Number of individuals in the bestList: n_{list}
- Crossover probability: p_{cx}
- Mutation probability: p_{mut}

The population size n_{pop} is derived from n_{list} since all individuals in the population are the result of n_{list} crossovers / mutations. However, not every genetic operation always produces a new offspring. Therefore, absolute numbers cannot be provided. The number of schedules which have to be computed for the creation of a policy using ListGA can be estimated as follows:

$$\begin{aligned} n_{schedules} &= n_{gen}(n_{list} + n_{pop}), \\ n_{pop} &\approx p_{cx} n_{list} + p_{mut} n_{list} \end{aligned} \tag{12}$$

The following paragraphs describe which values have been selected for the algorithm:

7.3.1.1 *Optimization of* n_{gen}

Figure 5 displays a box plot which compares the impact of different n_{gen} on the makespan of schedules created with scheduling policies created with ListGA. For every parameter set, 200 scheduling policies have been created and used to simulate one schedule. In the box plots, the box encompasses the inter-quartile range (*IQR*) with the median marked as a bar. The whiskers mark the lowest datum still within $1.5 \times$ IQR of the lower quartile, and the highest datum still

n_{gen}	MEAN (μ)	STD. DEV. (σ)	MEDIAN	# SCHEDULES
1000	86524	12896	82883	21000
500	86478	12627	83305	10500
250	93127	17285	88314	5250
100	101593	20278	95607	1950
5	125571	20452	122352	105

Table 2: Overview: Impact ofn_{gen} on ListGA Performance

within 1.5 × IQR of the upper quartile. Data points which fall above or below that range are considered outliers and are marked with a dot. The box plots are ordered by median. One can clearly see, that a larger n_{gen} results in a smaller median value of execution times and smaller spread of the data points within the inter quartile range. The mean execution time as a function of $_{gen}$ converges against a value close to the mean of $n_{gen} = 1000$. The amount of outliers seems not be affected by n_{gen}, but the total range is likely to decrease with higher values. Optimization of the remaining parameters was done using $n_{gen} = 250$ since it offers a good trade-off between computational effort and quality of result. Detailed information about the experiments are given in table 5. Please note that the last column displays the approximate number of schedules which have been generated during the generation of the scheduling policy. The following parameters were used to generate the figure: $p_{cx} = 0.9$, $p_{mut} = 0.2$, $n_{list} = 10$, 200 data points per parameter set.

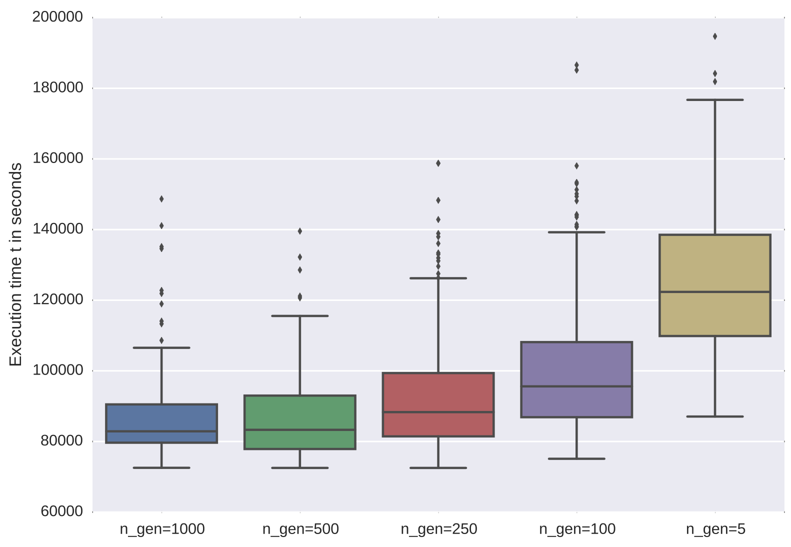

Figure 5: Box plot of 200 ListGA execution times with different n_{gen}

n_{gen}	MEAN (μ)	STD. DEV. (σ)	MEDIAN	# SCHEDULES
50	98895	21961	93270	5250
10	109253	21729	109253	1050
5	121012	22997	117472	525

Table 3: Overview: Impact ofn_{list} on ListGA Performance

7.3.1.2 *Optimization of* n_{list}

Figure 6 displays the impact of different n_{list} on the quality of the algorithm. As expected, a larger parameter size increases the algorithm's performance. This is logical because a higher n_{list} leads to a larger number of individuals which are examined. Since computational effort matters in our scenario, we have to make a trade off between the number n_{gen} of generations and the number n_{list} of best individuals selected each generation.

We found that it is more beneficial to choose larger n_{gen} over n_{list}, since the genetic algorithm has more time to evolve good solutions. For example, during the execution of $n_{gen} = 250$ from Fig. 5 approximately $250(10 + 0.9 \cdot 10 + 0.2 \cdot 10) = 5250$ schedules have been evaluated. The median execution time of 200 algorithm runs at this configuration is 88314 seconds.

The execution of $n_{list} = 50$ in Fig. 6 was done using $n_{gen} = 50$ and therefore also simulated approximately $50(50 + 0.9 \cdot 50 + 0.2 \cdot 50) = 5250$ schedules. However, the median execution time in this scenario was 93270 and therefore worse.

We therefore propose to use a n_{list} value of 10. Table 3 gives more details about the experiments. The experiments for this figure were conducted using the following settings: $p_{cx} = 0.9$, $p_{mut} = 0.2$, $n_{gen} = 50$, 200 data points per parameter set.

7.3.1.3 *Optimization of* p_{cx}

Figure 7 shows the impact different p_{cx} values have on the execution time of schedules generated with ListGA. The figure shows that higher crossover probabilities deliver better results. This can be explained with the way the ListGA is implemented: At the beginning of each generation, the last generation's n_{list} individuals are *copied* and used as a population, to which the genetic operators are applied. After calculating the fitness of the n_{list} best individuals of the last generation and the current population, the new n_{list} best individuals of *all* individuals is selected. This means that a larger p_{cx} results in a larger number of *different* individuals examined. Thus we propose to use $p_{cx} = 1$. More detailed experiment results are depicted in table 4. The experiments for this figure were conducted using the

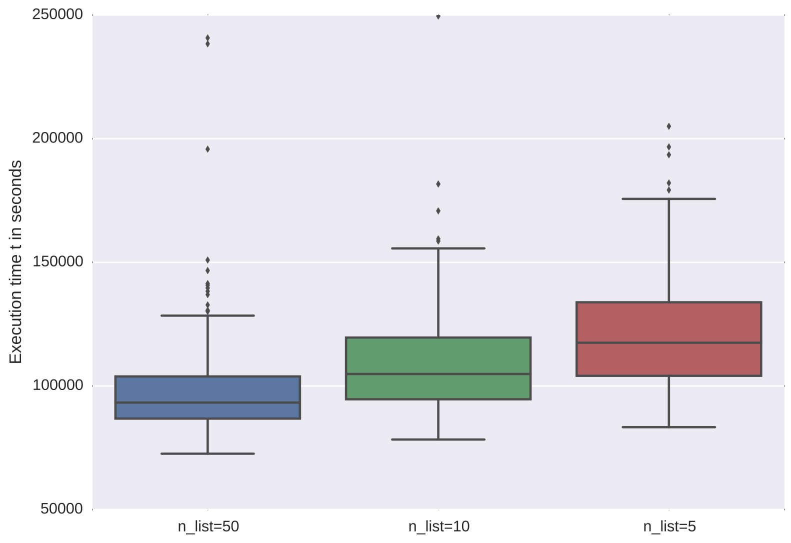

Figure 6: Box plot of 200 ListGA execution times with different n_{list}. Box plots to be interpreted as described in 7.3.1.1

p_{cx}	MEAN (μ)	STD. DEV. (σ)	MEDIAN	# SCHEDULES
1.0	89642	14864	84826	5500
0.9	93056	18578	88771	5250
0.6	96615	20406	92444	4500
0.3	101943	20343	97629	3750
0.1	107982	21064	103218	3250

Table 4: Overview: Impact ofp_{cx} on ListGA Performance.

following parameters: $p_{mut} = 0.2$, $n_{gen} = 250$, $n_{list} = 10$, 200 data points per parameter set.

7.3.1.4 *Optimization of* p_{mut}

The impact of p_{mut} is displayed in Fig. 8. It can be seen that the parameter has to be balanced correctly: Large values of p_{mut} (i.e. 1.0) introduce a greater variety in the population but they also tend to destroy "good" parts of the activity list every generation; e.g. They change too many of the individual as compared to the last generation. If the value is chosen too small (i.e. $0.01, 0.05$), the variation achieved by the crossover operator alone is smaller and some good solutions are missed. We found that the best results are achieved by using a p_{mut} value of 0.5. Additional information for this experiment is depicted in table 5. The simulations for this figure were conducted

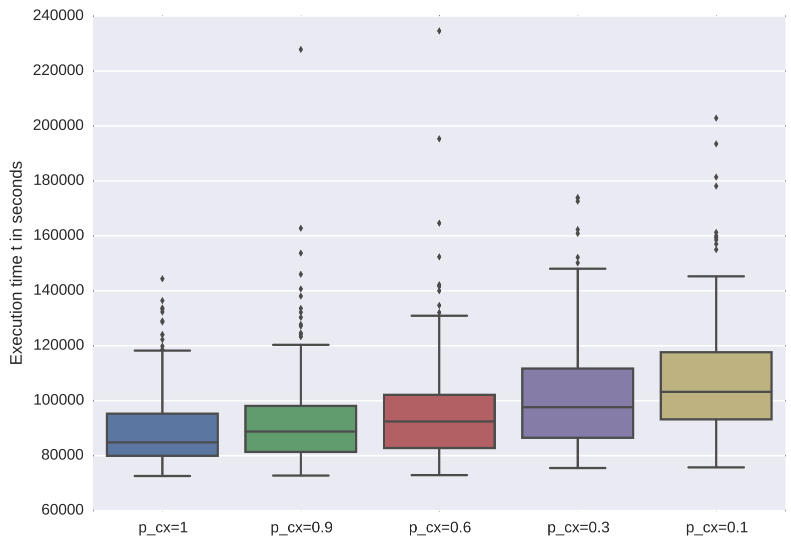

Figure 7: Box plot of 200 ListGA execution times with different p_{cx}. Box plots to be interpreted as described in 7.3.1.1.

p_{mut}	MEAN (μ)	STD. DEV. (σ)	MEDIAN	# SCHEDULES
0.5	90064	18618	85421	6250
0.2	91561	20370	85853	5500
0.7	90192	14868	87745	6750
0.1	95476	22074	89262	5250
0.05	96998	18837	91360	5125
0.01	102314	21830	96282	5025
1.00	110291	23416	105891	7500

Table 5: Overview: Impact ofp_{mut} on ListGA Performance

using the following parameters: $p_{cx} = 1.0$, $n_{gen} = 250$, $n_{list} = 10$, 200 data points per parameter set.

7.3.1.5 *Fitness over Time*

Figure 9 shows the minimum and maximum fitness values of the n_{list} best individuals over 250 generations. A genetic algorithm operating on a deterministic fitness function normally features a monotonic fitness over time. But the fitness function we proposed incorporates different manifestations of the random vector D and is therefore not deterministic. Thus it is possible that the same best individual can feature different fitnesses and when plotting the fitness over time, the curve is not monotonic. The simulations for this figure were

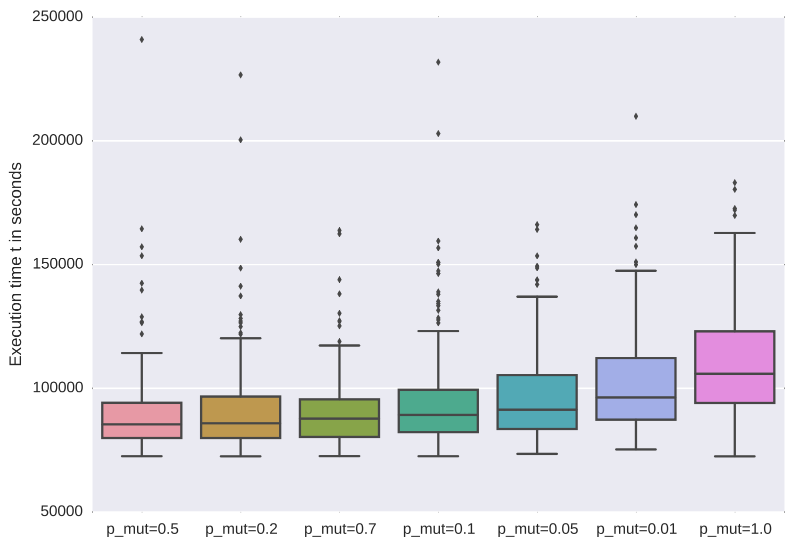

Figure 8: Box plot of 200 ListGA execution times with different p_{mut}. Box plots to be interpreted as described in 7.3.1.1

conducted using the following parameters: $p_{cx} = 1.0$, $p_{mut} = 0.5$, $n_{list} = 10$, $n_{gen} = 250$.

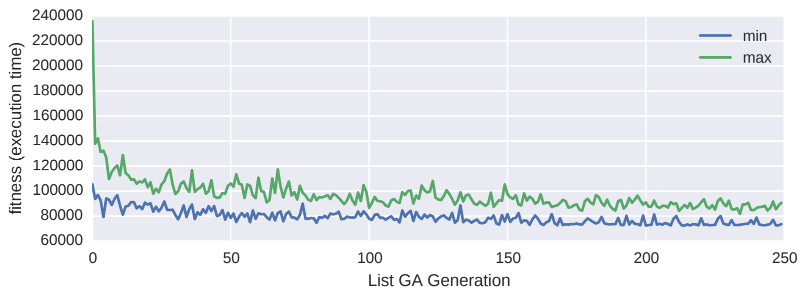

Figure 9: Min. and Max. fitness values of best individuals over 250 generations of ListGA.

7.3.2 *Computational results of ArcGA*

Since the ArcGA has a very similar structure as the ListGA it features the same optimization parameters. However, the fitness function is more complex, since it has to compute every realization D_i of D with and without consideration of X. Therefore, the number of schedules generated per generation is twice as high as compared to ListGA:

$$\begin{aligned} n_{schedules} &= 2n_{gen}(n_{list} + n_{pop}), \\ n_{pop} &\approx p_{cx}n_{list} + p_{mut}n_{list} \end{aligned} \tag{13}$$

7.3.2.1 *Convergence Problems*

After implementing the ArcGA, several tests have revealed that the proposed algorithm with stochastic activity times does not converge. Figure 10 shows the fitness over time with stochastic processing times and an unoptimized activity list. Please note that the ArcGa is a *maximizing* optimization problem since the fitness function is the difference between the execution times achieved without and with the precedence list X.

It can be seen that the minimum and maximum fitness values can display a large variation between two generations. This variation can also be observed in the ordering of individuals in the best list: An individual which is at the last position of the best list in generation n can be ranked best in generation $n+1$. I. e., the performance of an individual is strongly dependent on the realization D_i of D.

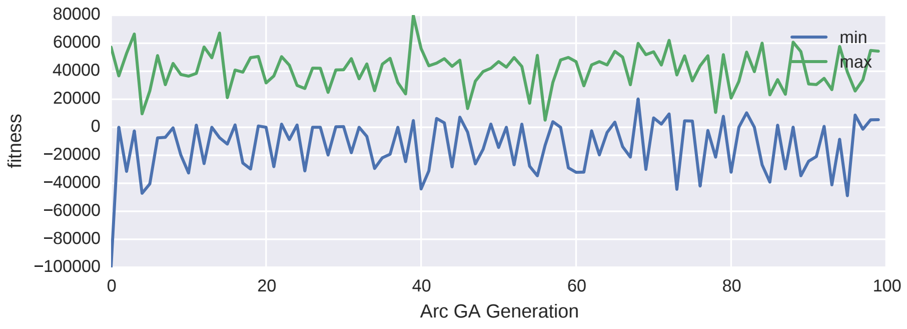

Figure 10: Min. and Max. fitness values of best individuals over 100 generations of ArcGA. Unoptimized activity list.

During the experiments process we noticed a great difference in the algorithms variability of fitness over generations depending on the quality of the activity list L which is used in conjunction with the ArcGA. Figure 11 displays the fitness of ArcGa over 100 generations with an activity list which has been optimized by ListGA. The maximum fitness now immediately converges at a fitness value of 0 (i.e. no impact on overall performance) or slightly above. The variability is much smaller as compared to fig 10, which is probably caused by the fact that the optimized activity list starts the algorithm in a local optimum.

7.3.3 *Algorithmic Variations*

Several variations of the algorithm have been implemented in order to achieve convergence. These variations together with their results are introduced shortly:

NON-OPTIMAL OPERATOR PARAMETERS The same kind of experiments as in 7.3.1 were conducted. This includes simulations with up

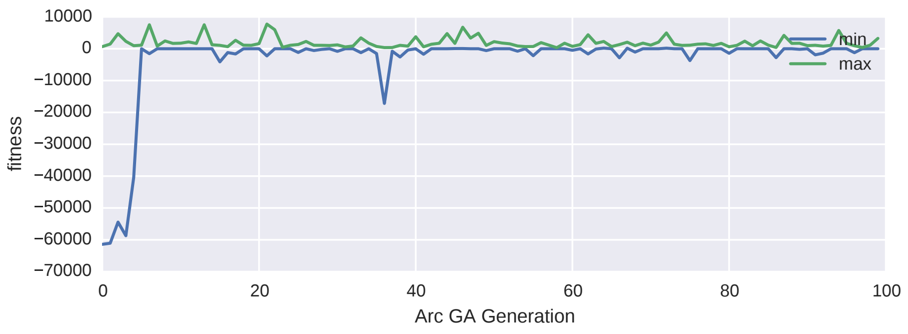

Figure 11: Min. and Max. fitness values of best individuals over 100 generations of ArcGA. Optimized activity list.

to 500 generations of ArcGA. No parameter set was able to significantly improve the results generated by ListGA.

NON-STOCHASTIC PROCESSING TIMES Another variation that was implemented utilizes deterministic processing times in the fitness function $[\Pi(E[D], L, X)]_n$, as suggested in the original approach. In this case the algorithm did converge when the activity list was unoptimized. Using an activity list optimized with ListGA did not. Instead, the overall performance dropped compared to ListGA only.

INTRODUCTION OF EUGENIC We have seen that the ArcGA implementation, as proposed by us, fails to find activity pairs which reduce the overall makespan of a schedule by consecutively judging every individual. We therefore tried to introduce a eugenic which guarantees the survival of individuals which performed well under certain conditions. We found that the overall makespan was increased by this measure, since individuals which are beneficial in one scenario unnecessarily delay the execution of activities in another.

7.3.4 *Comparison of Reference Problem Instances*

We have reviewed our implementation of ArcGA excessively and can exclude implementation mistakes. We therefore continue to compare the characteristics of the problem instances used by *Ashtiani et al.* to benchmark their algorithm.

In their work, they show that $(\mathcal{C}^{RB} \cup \mathcal{C}^{ES}) \subset \mathcal{C}^{PP}$ by supplying an example instance of the SRCPSP with several precedence constraints which is solved using the three mentioned scheduling policies. They found that the solution generated with $\mathcal{C}^{PP}$ features the shortest makespan. Their findings are reinforced by computational results gathered from solving reference problems (J120 from mpsplib[2]). J120 is a set of different RCPSP-instances with 120 non-dummy activities. Inspection of these instances revealed that they commonly

2 http://www.mpsplib.com/, accessed 20. April 2015

feature only a few resource types and a large number of precedence constraints (e. g.4 resource types, > 150 precedence constraints). Contrary, our instance of the EASP features a large number of resource types and no precedence constraints. We therefore argue, that when no precedence constraints are in place, $\mathcal{C}^{RB}$ is capable of delivering the same results as $\mathcal{C}^{PP}$ and thus restrain to only use the ListGA algorithm for solving the EASP, as long as there are no precedence constraints involved. This does not affect the algorithms advantage of being able to precompute the computationally expensive parts.

7.4 COMPARISON WITH REFERENCE IMPLEMENTATION

Figure 12 depicts a box plot comparing ListGA and the reference algorithm. 200 data points have been computed in each case. ListGA has been executed with the same parameters as described in section 7.3.1. In the experiments conducted, ListGA features a median execution time which 17% shorter (cf. table 6). The standard deviation is 33% smaller which dramatically decreases the IQR and therefore helps to make the scheduling process more predictable and easier to plan in a business environment. Figure 13 displays the mean of the datasets as bar charts. The standard error of each dataset is marked in green. An independent two-sample t-test yields a p-value of 7.514×10^{-15} for the null-hypothesis that the two samples originate from the same distribution. We can therefore say that the improvements observed are statistically significant.

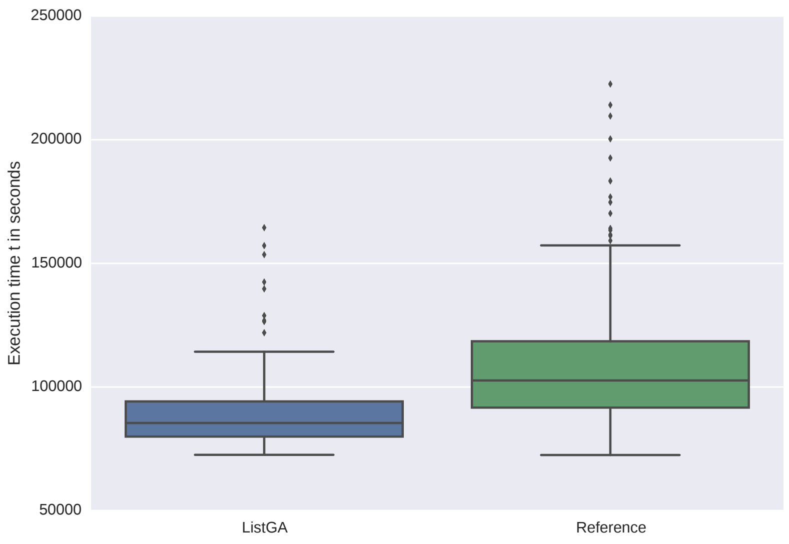

Figure 12: Box plot comparing ListGA and Reference Algorithm. Box plots to be interpreted as described in 7.3.1.1

ALGORITHM NAME	MEAN (μ)	STD. DEV. (σ)	MEDIAN
ListGA	90064	18618	85421
Reference	109446	27622	102636

Table 6: Overview: Comparison of ListGA and Reference Algorithm Performance

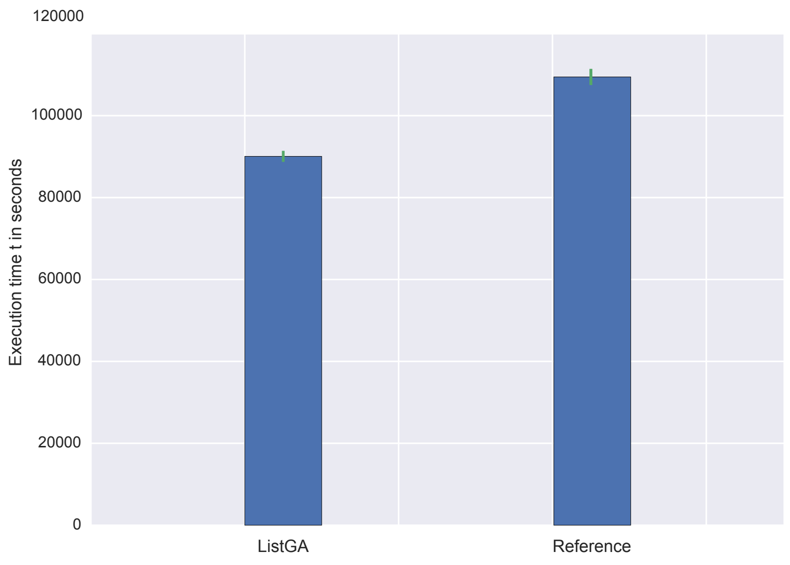

Figure 13: Comparison of ListGA and Reference means with standard error

Figure 14 and figure 15 in the appendix display Gantt Charts of schedules created with the reference algorithm and the ListGA. When comparing two different schedules of the same scheduling job, key performance criterion is how many tasks are completed in parallel and how many in serial. The two Gantt Charts depicted reveal that ListGA is capable of scheduling a large number of tasks concurrently even at later stages of execution.

8 CONCLUSIONS & OUTLOOK

8.1 CONCLUSIONS

In this work we introduced the exept AG scheduling problem as a representative of the stochastic resource constrained project scheduling problem. The foundation of this work was laid by first providing background information on resource constrained project scheduling with deterministic processing times. This foundation was later extended to stochastic processing times. In section two of this work we introduced an exemplary instance of the exept AG scheduling problem which is based on real world data. We later described how this data is used to create probability distributions which are used to model activity durations in a simulation environment. We proceeded to adapt a two-phase genetic algorithm for solving stochastic resource constrained project scheduling problems to the use case of the exept AG scheduling problem. Benchmarking and parameter optimization of the algorithm implementation revealed that the second phase of the genetic algorithm is not necessary when the dataset features no precedence constraints. With the first phase of the algorithm alone, our solution was able to achieve 17% better median execution times as compared to the reference implementation. The standard deviation was 33% smaller. The results were statistically significant.

8.2 OUTLOOK AND FUTURE WORK

Before implementing the algorithm proposed in the real application, there are a few facts which have to be considered first: The algorithm has to be tested with several instances of the exept AG scheduling problem with different characteristics. This includes other task duration distributions (e.g. software tests vs. hardware tests) and most importantly several amounts of precedence constraints between activities. Also, the results have to be verified again with more historic data eliminating the probability distributions of the activity durations as a possible error source.

We did not work on every interesting aspect of the problem. For instance, further possible steps would be to study the applicability of the software created for resource planning and bottleneck identification during production.

Also, our current implementation uses a simple heuristic for deciding which resource to assign to a task when it resolves all resource requirements. Further work could be to include this step in the planning horizon. Our current knowledge of the field suggests that this has not been done before.

The two genetic algorithms implemented also offer the potential for further research: It would be interesting to study the impact of different selection algorithms (e.g. tournament selection, fitness proportionate selection, etc.) on the ability of arcGA to converge to better solutions.

Finally, we suggest to evaluate the algorithm implemented in conjunction with different fitness functions: This could not only optimize the makespan, but machine workload, minimization of cost or strain on human resources.

Part III

APPENDIX

A ADDITIONAL FIGURES

This chapter contains additional full-page gantt diagrams depicting schedules created with the reference algorithm and ListGA.

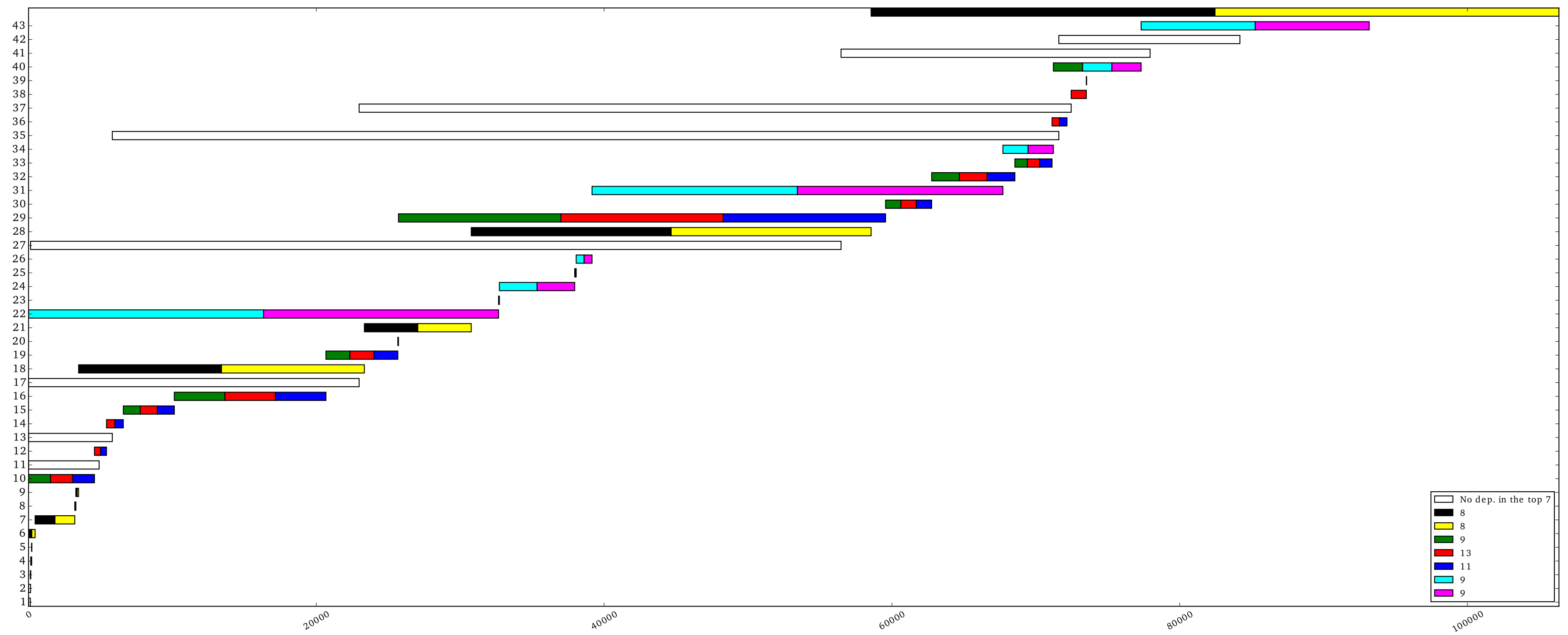

Figure 14: Gantt chart of schedule created with Reference Algorithm. X-Axis represents time in seconds. Y-Axis the task id (determined by point in time when the execution is finished). The colored markings represent the top 7 resources assigned during scheduling (i. e.a colored section on a task means that the corresponding resource was assigned to it). The numbers in the box behind the color indicate how often a resource has been assigned during scheduling.

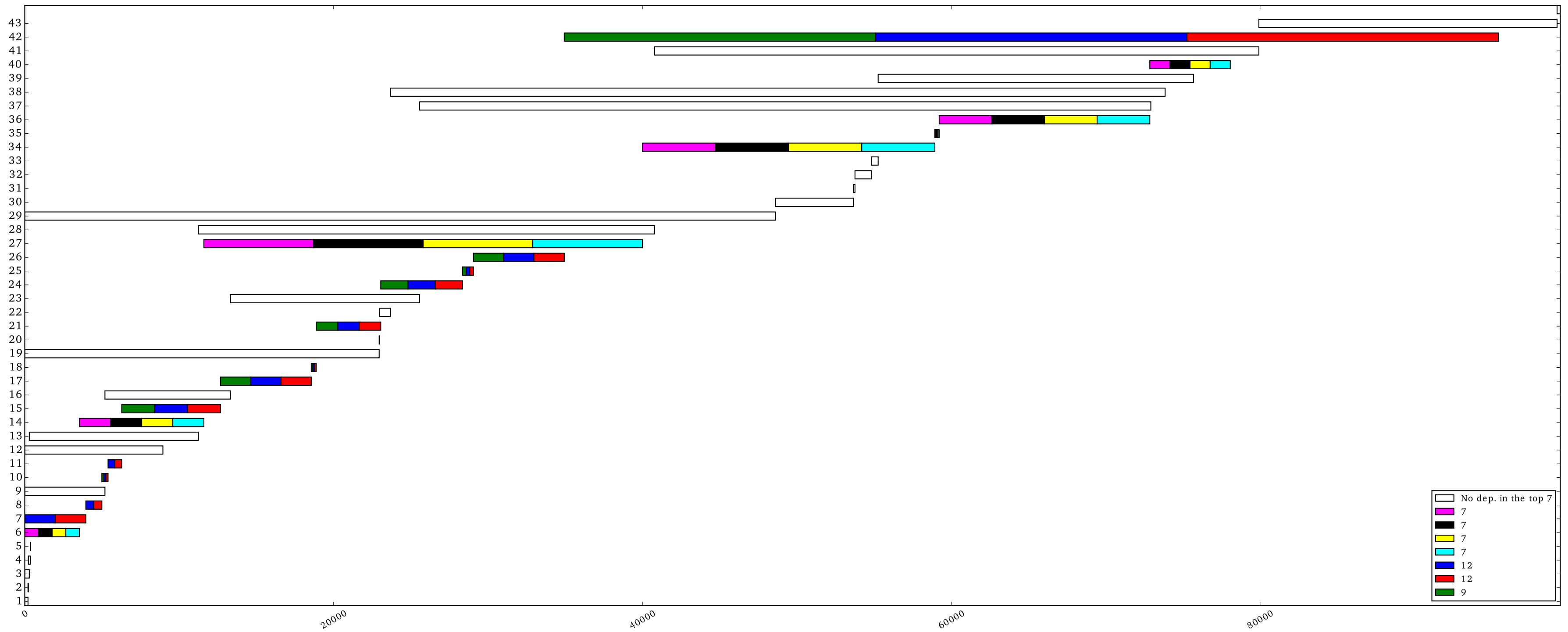

Figure 15: Gantt chart of schedule created with scheduling policy created with ListGA. X-Axis represents time in seconds. Y-Axis the task id (determined by point in time when the execution is finished).The colored markings represent the top 7 resources assigned during scheduling (i. e.a colored section on a task means that the corresponding resource was assigned to it). The numbers in the box behind the color indicate how often a resource has been assigned during scheduling.

BIBLIOGRAPHY

[1] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.

[2] Behzad Ashtiani, Roel Leus, and Mir-Bahador Aryanezhad. A novel class of scheduling policies for the stochastic resource-constrained project scheduling problem. *Available at SSRN 1368714*, 2008.

[3] Behzad Ashtiani, Roel Leus, and Mir-Bahador Aryanezhad. New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing. *Journal of Scheduling*, 14(2):157–171, 2011.

[4] Adedeji Bodunde Badiru. *Project management tools for engineering and management professionals*. Industrial Engineering and Management Press, Institute of Industrial Engineers, 1991.

[5] Francisco Ballestín. When it is worthwhile to work with the stochastic rcpsp? *Journal of Scheduling*, 10(3):153–166, 2007.

[6] J Banks, JS Carson, and BL Nelson. *DM Nicol, Discrete-Event System Simulation*. Prentice hall Englewood Cliffs, NJ, USA, 2000.

[7] Önder Halis Bettemir and Rifat Sonmez. Hybrid genetic algorithm with simulated annealing for resource-constrained project scheduling. *Journal of Management in Engineering*, 2014.

[8] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[9] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S Trivedi. Queueing networks and markov chains, 2000.

[10] KLEIN Bouleimen and HOUSNI Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149(2):268–281, 2003.

[11] Ralf Bruns. Direct chromosome representation and advanced genetic operators for production scheduling. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 352–359. Morgan Kaufmann Publishers Inc., 1993.

[12] J-H Cho and Y-D Kim. A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, pages 736–744, 1997.

[13] Edward W Davis and George E Heidorn. An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, 17(12):B–803, 1971.

[14] Lawrence Davis. Job shop scheduling with genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, volume 140. Carnegie-Mellon University Pittsburgh, PA, 1985.

[15] Erik Leuven Demeulemeester. *Project scheduling: a research handbook*, volume 102. Springer Science & Business Media, 2002.

[16] Luc Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pages 260–265. ACM, 1986.

[17] Andreas Drexl. Scheduling of project networks by job assignment. *Management Science*, 37(12):1590–1602, 1991.

[18] Salah E Elmaghraby, Rachid Benmansour, Abdelhakim Artiba, and Hamid Allaoui. On the approximation of arbitrary distributions by phase-type distributions. In *3rd International conference on information systems, logistics and supply chain ILS 2010*, 2010.

[19] Willliam Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.

[20] A.A. Fernandez. The optimal solution to the resource-constrained project scheduling problem with stochastic task durations. Unpublished Doctoral Dissertation, University of Central Florida., 1995.

[21] José Fernando Gonçalves, Jorge JM Mendes, and Mauricio GC Resende. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190, 2008.

[22] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[23] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[24] John J Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128, 1986.

[25] Sönke Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, 45 (7):733–750, 1998.

[26] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[27] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 2015-04-13].

[28] Andy J Keane. Genetic algorithm optimization of multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9(2):75–83, 1995.

[29] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5-6):975–986, 1984.

[30] Rainer Kolisch. *Project scheduling under resource constraints: efficient heuristics for several problem classes*. Springer Verlag, 1995.

[31] Rainer Kolisch and Sönke Hartmann. *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis*. Springer, 1999.

[32] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37, 2006.

[33] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[34] Svenja Lagershausen. *Performance Analysis of Closed Queueing Networks*, volume 663. Springer Science & Business Media, 2012.

[35] Donald G Malcolm, John H Roseboom, Charles E Clark, and Willard Fazar. Application of a technique for research and development program evaluation. *Operations research*, 7(5):646–669, 1959.

[36] KS Mountakis. *Stochastic Scheduling of Train Maintenance Projects*. PhD thesis, TU Delft, Delft University of Technology, 2013.

[37] Klaus Müller and Tony Vignaux. Simpy framework, 2002. https://simpy.readthedocs.org/en/3.0.5/.

[38] Klaus Neumann. *Stochastic project networks: Temporal analysis, scheduling and cost minimization*, volume 344. Springer Science & Business Media, 1990.

[39] Linet Ozdamar. A genetic algorithm approach to a general category project scheduling problem. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 29(1):44–59, 1999.

[40] Franz Joseph Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4(1):227–252, 1985.

[41] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[42] Frederik Stork. *Stochastic resource-constrained project scheduling*. PhD thesis, Technische Universität Berlin, 2001.

[43] Henk C Tijms. *A first course in stochastic models*. John Wiley and Sons, 2003.

[44] Peter JM Van Laarhoven, Emile HL Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations research*, 40(1):113–125, 1992.

[45] Matthew Bartschi Wall. *A genetic algorithm for resource-constrained scheduling*. PhD thesis, Massachusetts Institute of Technology, 1996.

[46] James M Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430–437, 2003.

[47] S David Wu, Eui-Seok Byeon, and Robert H Storer. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999.

[48] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

COLOPHON

This document was typeset using LaTeXand the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*.
Proudly made with free software.

Final Version as of April 29, 2015 (`classicthesis` - Final).