Universiteit Utrecht

Department of Information and Computing Sciences

# Combining local search and heuristics for solving robust parallel machine scheduling

Master's Thesis
ICA-3692701

*Author:*
G.J.P.N. Passage

*Supervisors:*
dr. ir. J.M. van den Akker
dr. J.A. Hoogeveen

May 2016

# Abstract

In the context of parallel machine scheduling with precedence relations, it is often desirable to obtain schedules which are robust against disturbances. In this thesis, we apply local search on problem instances of parallel machine scheduling where processing times of jobs are retrieved from a probability distribution and we attempt to minimize the expected makespan in order to maximize robustness of schedules.

We compare two types of objective functions. The first type of objective functions build upon a previously introduced combination of simulation and optimization, where expected makespan is approximated by computing the average makespan of a set of samples from a single schedule. The second type of objective functions build upon a previously introduced notion of robustness, where we attempt to incorporate properties of stochastic schedules on a single, deterministic schedule in order to approximate expected makespan by applying probability theory.

The objective functions are run within our presented local search framework on various problem instances, where the best solutions found within a given time frame are examined in order to find the best algorithms for various contexts. We have found that one of the objective functions on a single schedule introduced in this thesis is significantly more effective for minimizing the expected makespan than both fixation maximization and result sampling for nearly all of the tested problem instances.

# Acknowledgements

I would first like to thank my thesis supervisors, dr. ir. J.M. van den Akker and dr. J.A. Hoogeveen, for their exceptional support, insights, enthusiasm and time investment. Their excellent and frequent feedback has been of great value to me and allowed me to learn a lot throughout the process of writing this thesis. Our meetings have always been very pleasant and instructive. I would also like to thank prof. dr. H.L. Bodlaender as the second reader of my thesis and for helping me to find this subject.

Furthermore, I would like to thank my fellow student Bert Massop for his great unconditional support, valuable feedback and frequently helping me to develop new ideas. I would also like to thank my parents for their continuous moral support and encouragement. Finally, I would like to thank the attendees of my thesis defense for their interest, attention and feedback.

# Contents


**1 Introduction** **1**
- 1.1 Problem definition . . . 2
  - 1.1.1 Graph representation . . . 3
- 1.2 Related work . . . 4
- 1.3 Outline . . . 6

**2 Local Search** **7**
- 2.1 Neighbourhoods . . . 7
- 2.2 Valid operations . . . 8
- 2.3 Local search methods . . . 11
  - 2.3.1 Variable neighbourhood descent . . . 11
  - 2.3.2 Iterative local search . . . 12
  - 2.3.3 Conclusion . . . 15

**3 Result sampling** **16**
- 3.1 Introduction . . . 16
- 3.2 A new ILS perturbation step . . . 17
- 3.3 Reducing the neighbourhood . . . 17
- 3.4 Finding the best reinsertion . . . 18
  - 3.4.1 Computing critical path lengths using head and tail times . . . 19
  - 3.4.2 Runtime analysis . . . 20
- 3.5 Neighbourhood operator implementations . . . 21
  - 3.5.1 *1-move* and *2-move* implementation . . . 21
  - 3.5.2 *2-swap* implementation . . . 21

**4 Approximating $E(C_{max})$ based on expected processing times** **23**
- 4.1 Approximating the maximum of two normal distributions . . . 24
  - 4.1.1 Computing $E(X)$ and $\sigma^2(X)$ for known $\rho$ . . . 24
  - 4.1.2 Computing $E(X)$ and approximating $\sigma^2(X)$ for unknown $\rho$ . . . 24

4.2 Expected makespan without precedence relations . . . . . 25
4.2.1 Aggregated machine load ($AML$) . . . . . 25
4.2.2 Gaussian makespan (GM) . . . . . 26
4.2.3 Iterative Gaussian makespan (IGM) . . . . . 29
4.3 Incorporating slack from precedence relations (AD) . . . . . 30
4.4 Approximating completion times using dynamic programming (DM) . . . . . 33
4.4.1 Shortcomings of arctan delay . . . . . 33
4.4.2 Dynamic makespan framework . . . . . 34
4.4.3 Computation of $max(S_i + q_{ij}, S_j^{k-1})$ when $\theta(i) = \theta(j)$ 36
4.4.4 Calculating $Pr(\delta > 0)$ and $E(\delta|\delta > 0)$ when $\delta$ is not normally distributed . . . . . 37
4.4.5 An illustrative example . . . . . 39

**5 Experiments and results** **41**
5.1 Problem instances and general setup . . . . . 41
5.1.1 Distributions . . . . . 41
5.1.2 General set-up . . . . . 42
5.2 Parameter experiments . . . . . 42
5.2.1 Replacement of samples in result sampling . . . . . 43
5.2.2 VND stop criterion . . . . . 45
5.2.3 Perturbation step types . . . . . 48
5.2.4 Amount of perturbations . . . . . 49
5.2.5 Experiments for makespan without precedence relations 50
5.2.6 Choosing $\alpha$ and $\beta$ for the arctan delay function . . . . 52
5.3 Final experiments . . . . . 52
5.3.1 Experimental set-up . . . . . 53
5.3.2 Comparison of the approaches . . . . . 55
5.3.3 Run time . . . . . 56
5.3.4 Result deviations . . . . . 57
5.3.5 Comparing objective function evaluations . . . . . 57

**6 Conclusion and further research** **59**
6.1 Summary . . . . . 59
6.2 Conclusion . . . . . 60
6.3 Further research . . . . . 61
6.3.1 Improving efficiency . . . . . 61
6.3.2 Algorithm improvement . . . . . 62
6.3.3 Other machine scheduling problems and objectives . . 62
6.3.4 Other notions of robustness . . . . . 63

**Appendices** **64**

**A Final experiment results and run times** **65**

**B Limited run time experiments** **70**

**C Final experiment result and run time deviations** **73**

**D Comparison against fixation maximization** **75**

**E Experimental results for problem instances with large $q_{ij}$** **77**

**F Experimental results for function evaluations** **80**

**Bibliography** **84**

# Chapter 1

# Introduction

In the past decades, scheduling has been a popular research topic. Scheduling has applications in many fields, such as manufacturing, education, sport competitions, public transport and load balancing in processors. In general, scheduling incorporates assigning work to limited resources that can complete the work. In machine scheduling, the work is represented by jobs and the resources are represented by machines. In parallel machine scheduling, multiple machines operate at the same time and each machine can handle at most one job at once without pre-emption.

The majority of the current research in machine scheduling is about deterministic machine scheduling, where all data are known beforehand. However in reality, the actual data often differs from the expected data. The robustness of a schedule defines how well it can handle failures or disturbances. In this research, we investigate a specific problem category of parallel machine scheduling where processing times are taken stochastically to represent uncertainty in real-life models. For clarity, we will call this problem category robust parallel machine scheduling (RPMS). In the context of RPMS, we aim to find an efficient algorithm for generating schedules which maintain effectiveness with processing time disturbances. We also analyse which components of a schedule need to be optimized to generate robust schedules, and how these components can be optimized efficiently.

We use the RPMS problem in this research since it incorporates many properties of real-life scheduling problems such as release dates, precedence relations and availability of multiple machines. Therefore, many of the algorithms presented in this research can be used or incorporated to solve real-life scheduling problems. Examples include batch production in manufacturing and finding schedules in public transport taking possible delays into account and preventing delay propagations. The notion of robustness is especially relevant in real-life problems where dependency between jobs

should be minimized.

## 1.1 Problem definition

The RPMS problem consists of a set $J$ of $n$ jobs and a set $M$ of $m$ machines, where $n \geq 1$ and $m \geq 2$. Each job must be executed exactly once on a single machine and it can be scheduled on any machine. Each $j \in J$ has a release date $r_j$, stating that a job cannot be started before $r_j$, and a predefined processing time $p_j$, stating that a job needs $p_j$ time to be executed since it is scheduled on a machine. $p_j$ is independent of the machine the job is processed on. Execution of a job on a machine cannot be interrupted until it is completed.

To express disturbances in schedules, we assume that processing times are stochastic. A deterministic parallel machine scheduling problem instance can be turned into a RPMS instance by generating stochastic processing times $P_j$ from a probability distribution $D$ with average $p_j$ for all $j \in J$. In this thesis, $P_j$ follows the same probability distribution class for all jobs, where only the $p_j$ values differ. We enforce that $P_j = 0$ when a negative $P_j$ is generated. In the context of a stochastic schedule, $S_j$ and $C_j$ respectively denote the starting time and completion time of $j \in J$. Since execution cannot be interrupted, it must be that $C_j = S_j + P_j$ for all $j \in J$. All given times are in $\mathbb{R}_{\geq 0}$. The operator $\theta(j) = m_k$ denotes that $j$ is scheduled on machine $m_k \in M$ and the ordered set $J_{m_k}$ denotes $\{j|\theta(j) = m_k\}$ ordered by $S_j$. A schedule $S$ of a problem instance can be defined by setting $S_j$ and $\theta(j)$ for each job $j$.

Additionally, jobs have precedence relations, each stating that a job $j$ cannot be started until a certain time after another job $i$ is started. More precisely, let $P$ be a set of $r$ precedence relations. A precedence relation $(i, j) \in P$ for jobs $i, j \in J$ expresses the constraint that $S_j - S_i \geq q_{ij}$, where $q_{ij} \geq 0$, which means that $j$ should start at least $q_{ij}$ time units after $i$.

For each job $j$, we denote with $pp(j)$ ($ps(j)$) the sets of precedence predecessors (successors), that is, $i \in pp(j)$ if $(i, j) \in P$ and $i \in ps(j)$ if $(j, i) \in P$. Furthermore, for a schedule $S$, we denote that $i$ is a machine predecessor (successor) of $j$, shorthand $i = mp(j)$ ($i = ms(j)$), if $i$ is scheduled directly before (after) $j$ on the same machine. If $j$ is the first job on machine $m_k \in M$, we denote that $m_k = mp(j)$. If $j$ is the last job on a machine, $ms(j)$ is undefined. A job $j_1$ is an ancestor (descendent) of $j_k$ if there exists a possibly empty sequence of jobs $j_2, \ldots, j_{k-1}$ such that each $j_i$ is a precedence predecessor (successor) of $j_{i+1}$ for $1 \leq i < k$. Note that $i$ is also a precedence ancestor (descendent) of $j$ when $i$ is a precedence predecessor (successor), but an ancestor (descendant) is not always a precedence

predecessor (successor).

Graham et al. (1979) introduce a commonly used notation to define machine scheduling problems. Their notation consists of three parts: the machine environment, the job characteristics and the objective function. A commonly used objective function is the makespan, which denotes the latest completion time of any job. Since processing times are defined stochastically in this thesis, the makespan also becomes stochastic. Therefore, the robust machine scheduling problem with makespan optimization can be defined using the notation of Graham et al. (1979) as follows:

$$P|r_j, \text{stoch. } p_j, \text{prec}|E(C_{max})$$

Here, $C_{max}$ denotes the probability distribution of the makespan, $E(C_{max})$ denotes the expected makespan and $P$ indicates that we look at parallel machine scheduling, that is, problems with $m$ machines, where $m \geq 2$. The makespan time in a deterministic schedule, i.e. $P_j = p_j$, is denoted as $c_{max}$.

### 1.1.1 Graph representation

We introduce a graph representation to be able to express some important properties of RPMS. To express a given schedule $S$, we consider a graph $G(S) = (V, A)$, where $V$ is a set of vertices and $A$ is a set of directed arcs, distributed in three characteristic subsets $A_P$, $A_M$ and $A_D$. When $S$ is clear from context, it is omitted from the graph notation. They are defined as follows:

$$V = J \cup M \cup \{s, t\}$$

$$A_P = \{(i, j)|(i, j) \in P\}$$

$$A_M = \{(i, j)|mp(j) = i\}$$

$$A_D = \{(s, j)|j \in J\} \cup \{(j, t)|j \in J\}$$

The start and finish of the schedule are represented by the dummy nodes $s$ and $t$ respectively. Each arc has a weight. The arc set $A_P$ represents precedence relations, where $(i, j) \in A_P$ has weight $q_{ij}$. The arc set $A_M$ represents execution order on machines, where $(i, j) \in A_M$ has weight $P_i$ if $i \in J$ or 0 otherwise. The arc set $A_D$ represents release dates and completion times, where each arc $(s, j)$ has weight $r_j$ and each arc $(j, t)$ has weight $P_j$. This formulation enables us to express the makespan of a schedule $S$ by the length of longest path in $G(S)$. Note that within a problem instance $I$, the arcs in $A_M$ can be removed or added to obtain various different schedules and all other arcs remain constant. We also use vertices representing the

machines in order to express $mp(j) = m_k$ when $j$ is the first job on $m_k$.

This graph representation is similar to the disjunctive graph model used for job shop scheduling, as presented by Roy and Sussmann (1964). However, in the disjunctive graph model, jobs which can potentially be executed consecutively have an edge drawn between them. In a schedule, these edges are directed to represent execution order. Recall that for RPMS, jobs can be scheduled on any machine, contrary to job shop scheduling where operations are fixed on predefined machines. To represent potential execution orders of jobs, we would need $O(n^2)$ edges, where $n = |J|$, since any job can be scheduled before any other job unless it violates precedence constraints. Therefore, we represent the execution order of jobs on machines in a schedule by the existence of arcs, so there exists no edge between two jobs if they are not scheduled consecutively on a machine.

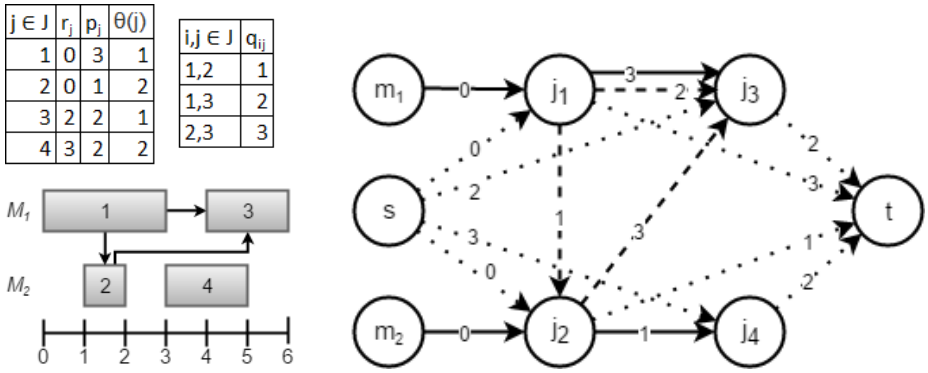
With this representation, an arc of length $l$ between $i$ and $j$, where $i, j \in J$, denotes that $j$ cannot start less than $l$ time units after $i$. This enforces that for $j \in J$, $S_j$ can not be smaller than the length of the longest path from $s$ to $j$ in a valid schedule. Therefore, since each $j \in J$ is also connected to $t$ with weight $p_j$, the makespan is defined by the longest path from $s$ to $t$. Since the remainder of this thesis is written in the context of makespan minimization, we define $S_j$ to be the longest path from $s$ to $j$, i.e. the earliest possible starting time of $j$ in a valid schedule. As a consequence, a schedule $S$ can be defined uniquely by the sets $J_{m_k}$ for each $m_k \in M$.

Note that $G$ may not always represent a valid solution. If $G$ contains a cycle, it cannot represent a schedule since the execution order of the jobs on the cycle cannot be defined. Since $S_j$ is defined as the longest path between $s$ and $j$, jobs will not overlap and no precedence relations will be violated in any schedule $S$.

In Figure 1.1, a graph representation is drawn for a problem instance with two machines, four jobs and three precedence relations.

## 1.2 Related work

Recall that a schedule is called robust when the effect of failures and disturbances in the schedule is minimal. A delay of a single job $j$ could delay many more jobs if not enough idle time is planned between successive jobs of $j$, which in turn could also delay execution of their successors. Hoppenbrouwer (2011) proposed a measure for robustness by minimizing the number of different machines on which the successors for each job are scheduled. More precisely, for any job $j$ and any machine $m$, let $\gamma_{jm}$ be 1 if any successor of $j$ is scheduled on $m$, otherwise $\gamma_{jm} = 0$. Hoppenbrouwer attempts to maximize robustness by minimizing $\sum_{j \in J} \sum_{m \in M} \gamma_{jm}$, from now on abbreviated




(a) Example schedule with data (b) Graph representation of the example schedule

Figure 1.1: A simple example schedule with the corresponding graph representation. Arcs from $A_D$ are dotted, arcs from $A_P$ are dashed and arcs from $A_M$ are solid. Arc weights are represented by the numbers on the arcs.

as $\sum \gamma$, with the additional condition that the makespan is minimized. Unfortunately, it seems hard to find optimal or near-optimal solutions for larger problem instances. Therefore, Van Roermund (2013) suggested to maximize robustness by maximizing the number of *fixations*. A precedence relation is fixated if the corresponding jobs are executed on the same machine. More precisely, for each $(i, j) \in P$, let $f_{i,j}$ be 1 if jobs $i$ and $j$ are scheduled on different machines, otherwise, $f_{i,j} = 0$. Now, robustness is maximized by minimizing $\sum_{(i,j)\in P} f_{i,j}$. Note that $\sum \gamma$ cannot be larger than $\sum_{(i,j)\in P} f_{i,j}$ and fixating a precedence relation cannot increase $\sum \gamma$. Van Roermund (2013) maximizes $\sum_{(i,j)\in P} f_{i,j}$ for schedules where $C_{max} \leq d$ for some general deadline $d$. Additionally, Van Roermund (2013) specifies a deadline $\bar{d}_j$, i.e. it must be that $c_j \leq \bar{d}_j$ for each $j \in J$. The approach of Van Roermund (2013) scales up better than the approach of Hoppenbrouwer (2011), quickly solving problems with up to 60 jobs, 60 relations and 8 machines to optimality and problems with 60 jobs, 150 precedence relations and 4 or 8 machines to near-optimality.

Van den Akker et al. (2013) attempted to solve stochastic job shop scheduling with stochastic processing times by combining local search and simulation, similar to the expected makespan minimization algorithm presented in Chapter 3. A set of random stochastic realisations are drawn from a schedule $S$. These samples are realized using discrete event simulation. From this sample set, the average of the $C_{max}$ values is used to approximated the expected makespan. For the local search part, they used simulated annealing and they introduced the critical path block swap and waiting left shift neighbourhoods, both reversing the order of two subse-

quent operations. This approach outperforms the classical methods, where stochastic values are replaced by deterministic values with some extra slack time included, depending on the mean value and underlying distribution, to represent potential delays.

## 1.3 Outline

For complex machine scheduling problems, it is usually impractical to use an exact algorithm to solve the problem to optimality. Local search algorithms are commonly applied to find a good approximation of the optimum in various problems. In Chapter 2, an in-depth explanation is given about how local search can be applied to solve RPMS.
Makespan minimization is one of the most commonly used objective functions in machine scheduling. Therefore, in this thesis, we investigate only expected makespan minimization. In Chapter 3, we attempt to minimize the expected makespan by sampling schedules with different $P_j$ realizations from the same $p_j$ for each $j \in J$ and minimizing the makespans of these schedules, similar to the approach of Van den Akker et al. (2013). In Chapter 4, we introduce various objective functions on a single deterministic schedule, with the goal of approximating the expected makespan, similar to the approach of Van Roermund (2013). In Chapter 5, we investigate parameter setting and present the experimental setup and results. In Chapter 6, we discuss the found results, present the conclusion and add pointers for further research.

# Chapter 2

# Local Search

The problem P$||C_{max}$ is NP-hard, as proven by Lenstra et al. (1977). Since it is a simplification of RPMS with expected makespan minimization, it must be that RPMS is also NP-hard. Additionally, there may be precedence relations and the processing times are stochastic in RPMS. As a consequence, it is difficult to find an optimal solution even for small problem instances. Therefore, we use local search algorithms to find good approximations of the optimum. The goal of this chapter is to define a local search framework to solve RPMS problems as effectively as possible, exploiting the nature of RPMS. The implementation details of this framework are specified in later chapters for solving specific RPMS problems.

## 2.1 Neighbourhoods

In local search, a better solution may be found by modifying parts of a given solution. The operator performing this modification is called the neighbourhood operator $N$. A schedule $S'$ is called a neighbour of a schedule $S$ if $S' \in N(S)$. The set of all $S' \in N(S)$ is called the neighbourhood of $S$. For RPMS, we define the following two neighbourhood operators:

1. The *k-move* operator. A move selects a single job and selects a new machine predecessor not equal to its old machine predecessor, or puts it as the first job on a machine. Note that this new machine predecessor can be on any machine, including the machine of the old machine predecessor. The *k-move* operator applies a sequence of $k$ successive moves.

2. The *k-swap* operator. A swap selects two jobs $i$ and $j \neq i$ and switches their positions in the schedule. The *k-swap* operator applies

a circular series of swaps between $k$ vertices. More precisely, $k$ vertices $j_{s_1}, \cdots, j_{s_k} \in J$ are selected and $j_{s_i}$ is put behind $mp(j_{s_{i+1}})$ for $1 \leq i < k$ and $j_{s_k}$ is put behind $mp(j_{s_1})$. Note that $k$ should be at least 2. When $k = 2$, a simple swap is performed.

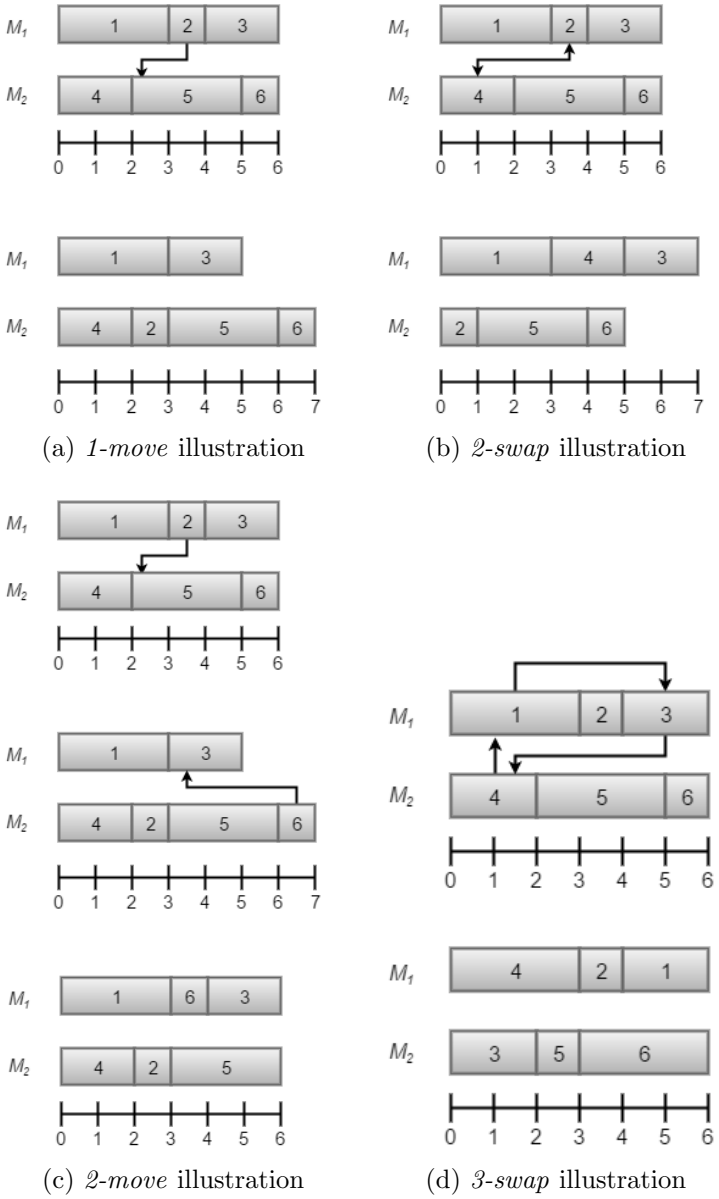
In this research, we only use the *1-move*, *2-move* and *2-swap* operators for local search neighbourhoods. We use the *5-swap* operator as a perturbation operator in iterative local search, as explained later in this chapter. An illustration of these neighbourhood operators is given in Figure 2.1.

The *k-move* operator has a neighbourhood of size $O(n^{2k})$, since $O(n)$ jobs and $O(n)$ predecessors can be selected for each move. Since the *k-swap* operator selects $k$ jobs, it has a neighbourhood of size $O(n^k)$. Note that in practice, the size of the neighbourhoods could be smaller since some operations may violate precedence constraints. The simplest operators (*1-move* and *2-swap*) are the fastest, since operation time linearly depends on $k$. Moreover, their neighbourhoods usually have the largest improvement proportion, since partial moves of larger neighbourhood operators often majorly disrupt the schedule. For makespan minimization, it is often impossible for the *1-move* operator to find an improvement when all machines are nearly equally occupied, since any move could disrupt this balance. On the other hand, the *2-swap* operator never changes the number of jobs on each machine, which could be problematic when this distribution is not balanced. Therefore, it may be a good idea to use both *1-move* and *2-swap* operators in a local search algorithm to cancel out their drawbacks. The *2-move* operator also cancels out the drawbacks of the *1-move* and *2-swap* operators, since it is a generalisation of both operators. However, it is often a less effective operator, since it contains a relatively large proportion of strongly disruptive operations. Nevertheless, it may still find improvements when the *1-move* and *2-swap* neighbourhoods are exhausted, since the *2-move* neighbourhood is much larger.

## 2.2 Valid operations

Recall that a move is invalid if it creates a cycle in the graph representation of the schedule. The *k-move* step will not create a cycle when all individual moves in a *k-move* operation do not create a cycle. Therefore, we can narrow the scope of finding valid *k-move* operations to finding valid *1-move* operations.

A cycle is created when there exists a path from $j \in J$ to some $j_p \in pp(j)$ after moving $j$ in graph $G(S)$. Similarly, a cycle is created when there exists a path from $j_s \in ps(j)$ to $j$ after moving $j$. Recall that when a job $j$ is the




(a) *1-move* illustration (b) *2-swap* illustration

(c) *2-move* illustration (d) *3-swap* illustration

Figure 2.1: Example of neighbouring solutions in different neighbourhoods for a schedule with 2 machines and 6 jobs

first job on machine $m_i$, $m_i$ is the machine predecessor of $j$ in $G$. When a cycle is created by moving $j$ behind a machine predecessor $f \in J \cup M$, $f$ is denoted as a forbidden machine predecessor. The set of all forbidden machine predecessors of $j$ in $G(S)$ is denoted as $F_j(S)$.

Let $pp(j)^<$ be the set of jobs for which a path from $j_{<p} \in pp(j)^<$ to any $j_p \in pp(j)$ exists and let $pp(j)^\leq = pp(j)^< \cup pp(j)$. If $j$ is placed *before* any job in $\leq pp(j)$, then a cycle is created, since there exists a path from $j$ to some $j_{\leq p} \in \leq pp(j)$, a path from $j_{\leq p}$ to some $j_p \in pp(j)$ and a path from $j_p$ to $j$. Therefore, $f \in F_j(S)$ if $f \in pp(j)^\leq$ *except* when $f$ is the latest job in $pp(j)^\leq$ on its machine, since placing $j$ after $f$ will not create a cycle then. Additionally, $f \in F_j(S)$ if there exists a path from any $j_s \in ps(j)$ to $f$ or when $f \in ps(j)$. All $f \in F_j(S)$ can be found by recursively back-tracing the arcs from all $j_p$ and forward-tracing the arcs of all $j_s$. This can be done in $O(m + n + r)$ time when appropriate caching is used. Then, valid *1-move* predecessors for $j$ are represented by $(J \cup M) \setminus F_j(S)$.

Consider the example schedule in Figure 2.2. We have $pp(j)^\leq = \{M_1, M_2, j_1, j_2, j_4\}$, since $j_4 \in pp(j_5)$ and for the remaining machine predecessor candidates, there exists a path to $j_4$. A cycle is created in $G(S)$ when $j$ is placed before $j_1$ or $j_4$ and consequently, $j_1$ and $j_4$ are not included in $F_{j_5}(S)$. Since $j_6 \in ps(j_5)$, we have that $j_6 \in F_{j_5}(S)$ and since there exists a path from $j_6$ to both $j_7$ and $j_8$, they are contained in $F_{j_5}(S)$ as well.

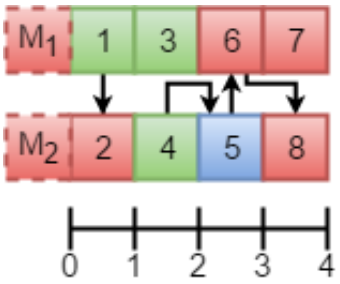



Figure 2.2: An example schedule with 8 jobs, 2 machines and 4 precedence relations $(j_1, j_2)$, $(j_4, j_5)$, $(j_5, j_6)$ and $(j_6, j_8)$. All jobs have $p_j = 1$ and $r_j = 0$ and all precedence relations have $q_{ij} = 0$. Forbidden machine predecessors of $j_5$ are red and allowed machine predecessors are green.

The valid operations for the *k-swap* operator can be defined similarly. $k$ jobs $j_1 \dots j_k$ can be moved by the *k-swap* operator if $mp(j_{i+1}) \notin F_i(S)$ for $1 \leq i < k$ and $mp(j_1) \notin F_k(S)$.

## 2.3 Local search methods

There exists a large variety of local search methods. The quality of a local search method is largely dependent on its parameters and the problem it is applied to. For different problems, different local search methods may prove to be the most effective. We found that an iterative local search algorithm using a variable neighbourhood descent is effective for finding RPMS schedules with a low expected makespan. In this section, we explain the implementation of this algorithm and why we found it to be effective.

### 2.3.1 Variable neighbourhood descent

In Section 2.1, we have defined various neighbourhood operators, each with their own benefits and drawbacks. We try to combine these neighbourhoods in such a way that we cancel out their drawbacks and combine their benefits. Variable neighbourhood descent (VND), introduced by Mladenović and Hansen (1997), is the simplest local search algorithm using multiple neighbourhoods. Suppose $N_1 \dots N_{k_{max}}$ is a sequence of $k_{max}$ neighbourhood operators. When a large neighbourhood is fully investigated, the algorithm could take a long time to complete. It may be beneficial to stop investigating this neighbourhood and try a different neighbourhood or to stop the algorithm. For this purpose, we introduce a parameter $l$, specifying the maximum number of neighbours investigated in a single neighbourhood. A VND iteration uses a subroutine called first-improvement, which repeatedly iterates over $N(S)$ until an improvement is found and stops when $l$ neighbours have been investigated or $N(S)$ is exhausted. It returns $S$ if it finds no improvement. Now, the VND algorithm is defined as follows:

- Generate an initial solution $S$ and let $k = 1$
- Repeat the following steps until the stopping criterion is met:
  - Apply first-improvement on $S$ using the neighbourhood $N_k$ and investigating at most $l$ neighbours in a random order, returning a potentially better solution $S'$.
  - If $S'$ is better than $S$, continue with $S = S'$ and $k = 1$.
  - Otherwise, continue with $k = k + 1$, or stop and return $S$ if $k = k_{max}$.

Note that the first-improvement subroutine could be replaced by the best-improvement subroutine, where the neighbourhood is searched exhaustively and the best neighbour is returned, but this process takes more CPU

time. The stopping criterion is met if all neighbourhoods are investigated without finding an improvement, or after a certain amount of CPU time.

VND is a simplification of the variable neighbourhood search algorithm (VNS). In VNS, the first-improvement subroutine is replaced by a local search subroutine. If the local search subroutine does not find a better solution, a random neighbour is drawn from the current best solution with the current neighbourhood. Then, VNS continues with the next neighbourhood applied on this solution. Drawing a random neighbour is called the *shaking* step of VNS. A more detailed explanation of VNS and its variants is given by Mladenović and Hansen (1997). For RPMS, the *shaking* step is often so disruptive that it takes several steps in the next neighbourhood to find an improvement. Then, it finds a local optimum which is significantly different from the previous best found solution. Therefore, the power of the neighbourhoods is not combined as it is in VND, where all neighbourhoods are applied on a local optimum so that the final local optimum is of maximal quality. For this reason, we have found that VND generally produces better results for minimizing $E(C_{max})$.

For local search algorithms with variable neighbourhoods, it is important to consider the order of $N_1 \dots N_{k_{max}}$. The most effective neighbourhoods should be used first, whereas the less effective, often larger neighbourhoods should only be used if the more effective neighbourhoods have been exhausted. Therefore, it is usually most effective to use the *1-move* and *2-swap* operators before performing the *2-move* operator.

### 2.3.2 Iterative local search

Hill climbing is the simplest local search algorithm, where first-improvement or best-improvement is repeated until no better solution could be found. A straightforward improvement of hill climbing is multi start local search (MSLS), where local search is repeated from different starting solutions and the best found local optimum is returned. MSLS generally finds very diverse local optima. However, a better local optimum can often be found by slightly adjusting the current local optimum and starting a local search algorithm from this modified local optimum. Iterative local search (ILS) is based on this principle. ILS uses local search to find a local optimum and repeatedly restarts local search from a modified local optimum. These modifications are called perturbations. After perturbing a solution and applying the local search algorithm to find a new solution, an acceptance criterion is necessary to decide whether we want to continue the process from this new solution or the previous solution. Using these principles, a general ILS framework is set up as follows:

**Algorithm 1** Iterative local search

**Precondition:** Initial solution $S$

1: $S^* \leftarrow$ local-search($S$)
2: **repeat**
3: $S' \leftarrow$ perturb($S^*$)
4: $S^{*\prime} \leftarrow$ local-search($S'$)
5: $S^* \leftarrow$ acceptance-criterion($S^{*\prime}$,$S^*$)
6: **until** Termination criterion met

To realize the full potential of ILS, we need to specify the following parameters:

- The algorithm generating an initial solution.
- The local search sub-procedure.
- The size and type of the perturbation steps.
- The acceptance criterion for accepting either the previously best found solution or the current solution.

Lourenço et al. (2003) extensively discuss ILS, presenting these parameters and how they can be optimized. Here, we present how these parameters can be specified for RPMS specifically.

**Initial solution** The most straightforward way to generate an initial solution is by generating a random valid solution. However, it is often better to generate a greedy initial solution. It may take a few local search steps from a random solution to generate a solution of comparable quality to a greedy solution. Therefore, greedy initial solutions are especially effective when a good solution should be found within a few local search steps, for example when the allowed CPU time is low or the problem instance is very large. Moreover, a greedy initial solution could contain beneficial properties which might remain in the local search solutions, thereby generating better local optima. A useful property of greedy initial solutions in RPMS is that the jobs with the earliest release dates are planned first. However in RPMS, we have found by initial experiments that ILS is not improved significantly by starting from a greedy solution. The saved CPU time is negligible, since in most cases, a better schedule is found after ten iterations from a random solution. Moreover, local search has a tendency to find schedules with the same beneficial properties as a greedy schedule. Nevertheless, since a

random initial solution does not provide any benefits, we still use a greedy initial solution. A greedy initial solution is composed by repeatedly selecting the job with the lowest possible starting time from all available jobs and putting it on the least occupied machine. A job is available when all of its predecessors from precedence relations are completed.

**Local search sub-procedure** ILS performs better when the local search sub-procedure performs better. Hill climbing is the fastest and most straightforward sub-procedure. Alternatively, more sophisticated local search algorithms like simulated annealing and tabu search can be used. However, for RPMS we have found that a VND subroutine generates more promising results compared to the attempted simulated annealing and tabu search subroutines. This could be explained by the fact that local optima are strongly clustered so that it takes a relatively long time to explore a plateau.

Moreover, it is important to consider the right neighbourhood for the local search sub-procedure. Therefore, we use the VND algorithm to combine the benefits of the presented neighbourhoods. The order of the neighbourhoods in the VND algorithm will be specified in the next chapters.

**Perturbation steps** ILS performs best with a good balance between intensification and diversification. If perturbation steps are too small, the local search sub-procedure finds a solution $S^{*\prime}$ identical or very similar to $S^*$, causing no improvement. If perturbation steps are too large, ILS behaves like MSLS. As a consequence, the probability of finding a better solution is decreased and the local search sub-procedure needs more steps to find a new local optimum. Small perturbation steps enhance intensification, whereas large perturbation steps enhance diversification.

It is also important to consider the type of perturbation. A good perturbation step is hard to undo by the local search sub-procedure. The *k-swap* operator with $k > 2$ needs $k$ consecutive *1-move* or *2-swap* steps to undo, making it a good candidate for perturbation steps. Various values of $k$ for *k-swap* perturbations are compared in the experiments.

To enhance diversification, one may choose to combine multi-start local search with ILS by restarting the ILS algorithm from a new initial solution after a certain amount of perturbation steps. Alternatively, a large perturbation step can be performed instead of a complete restart. Since we only perform up to 8 perturbations for the entire local search process in the experiments, we choose not to restart or perform large perturbations.

**Acceptance criterion** The acceptance criterion also influences the balance between intensification and diversification. Always accepting $S^{*\prime}$ en-

hances diversification, whereas accepting $S^*$ unless $S^{*\prime}$ is better enhances intensification. As an intermediate solution, one could accept $S^{*\prime}$ with a probability, dependent on the quality difference between $S^*$ and $S^{*\prime}$. However, since we only perform a few perturbations before the stopping criterion is met, we only accept $S^{*\prime}$ when it is better than $S^*$.

### 2.3.3 Conclusion

To summarize, the implementation of the ILS algorithm used in this research is given in Algorithm 2.

**Algorithm 2** ILS implementation

**Precondition:** Problem instance $P$, perturbation amount $pa$, perturbation size $k$, objective function $f$

1: $S \leftarrow$ Greedy-initialize($P$)
2: $S^* \leftarrow$ VND($S$)
3: $i \leftarrow 0$
4: **while** $i < pa$ **do**
5: $\quad S' \leftarrow k\text{-}swap(S^*)$
6: $\quad S^{*\prime} \leftarrow$ VND($S'$)
7: $\quad$ **if** $f(S^{*\prime}) < f(S^*)$ **then**
8: $\quad\quad S^* \leftarrow S^{*\prime}$
9: $\quad i \leftarrow i + 1$

Here, $f$ is an objective function which generates a large (e.g. 10000) amount of samples and returns the average makespan of these samples for the given solution, in order to accurately approximate $E(C_{max})$. Furthermore, the perturbation amount $pa$ denotes the maximum amount of performed perturbations.

# Chapter 3

# Result sampling

Recall from Section 1.1.1 that one could calculate the longest path in the graph representation $G$ to calculate the makespan $C_{max}$ of a deterministic RPMS schedule, that is, where each $P_j = p_j$. When each $P_j$ is stochastically generated from $p_j$, the makespan also becomes stochastic. This introduces the challenge of finding a good approximation of the expected makespan $E(C_{max})$. On the one hand, this approximation should be as accurate as possible, that is, when a schedule $S$ has a better $E(C_{max})$ value than another schedule $S'$, the approximation function of $E(C_{max})$ should conclude the same. On the other hand, the approximation function should be easy to compute, so that more local search steps can be done in the same time.

## 3.1 Introduction

One could argue that $E(C_{max})$ of a schedule $S$ can be approximated by drawing some random stochastic realisations of $S$ and taking the average of all their $C_{max}$ values. Throughout this thesis, we refer to this method as (result) sampling.

When more samples are used, the objective function becomes more accurate but slower, so a trade-off needs to be made to decide on a good amount of samples. Additionally, the accuracy of the objective function depends on the variance of the underlying probability distribution $D$ of processing times. For example, more samples are required to accurately approximate $E(C_{max})$ when $D$ is exponentially distributed compared to a uniform distribution U$[0.9p, 1.1p]$.

Result sampling has originally been presented for job shop scheduling by Van den Akker et al. (2013). The implementation details of our approach differ from the result sampling approach of Van den Akker et al. (2013), since we apply sampling for RMPS instead of job shop scheduling. Our local

search implementation is different from the local search implementation of Van den Akker et al. (2013). As the performed experiments in this research demonstrate, replacing the sample set every iteration is less effective for RPMS, since it often happens that the selected solution is better for the current set of samples, but worse for the next set of samples. Consequently, the local search will not converge to a good local optimum. Moreover, using only 5 or 10 samples generates very poor results for RPMS.

Van den Akker et al. (2013) present a technique called cutoff sampling, where the samples with the largest and smallest critical path lengths are removed from the set of samples. As demonstrated by the experiments, adding cutoff sampling in addition to sample replacement after every iteration is still less effective than keeping the same sample set. We have also found that the local search subroutine does not converge if cutoff sampling is performed while keeping the sample set every iteration, since the selected samples by cutoff sampling may be different for each iteration. For the remainder of this section, we therefore assume that the same sample set is used throughout the entire VND subroutine and that cutoff sampling is not used.

## 3.2 A new ILS perturbation step

When the used samples are always the same throughout the ILS procedure, the found local minima will be over-fit on these samples, where the approximated $E(C_{max})$ may be much lower than the actual $E(C_{max})$. However, the local search may never converge when samples are replaced too often, since it often performs a step which is effective for the current group of samples, but counter-productive for the next group of samples. As an intermediate approach, one can replace some samples when a local optimum is found by the local search subroutine of ILS as a perturbation step. Then, the local search subroutine can find a new local optimum which is specifically fit on the new samples, while still keeping some beneficial properties from the previous sets of samples. The perturbation step size is now the amount of samples being replaced. This perturbation step is compared to the *k-swap* operator in the experiments.

## 3.3 Reducing the neighbourhood

Recall that $j \in J$ can only be moved behind a new machine predecessor when a cycle is not created in the graph representation. Van den Akker et al. (2013) state that for the job shop scheduling problem, the neighbourhood

can be restricted to jobs on the critical path. We prove that this is also the case for the RPMS problem.

**Theorem 3.3.1.** *The critical path length can decrease only by reinserting a job that is currently located on the critical path.*

*Proof.* Suppose there is a job $j$ that is not on the critical path and $j$ is removed from $G$. Since $j$ was not on the critical path, the longest path from $s$ to any job on the critical path should remain the same and therefore, the critical path should remain the same. Furthermore, reinserting $j$ after removing it from $G$ could never cause the critical path length to decrease. □

Theorem 3.3.1 implies that for a single sample of a schedule, the *1-move* operator should only reinsert jobs on the critical path to decrease the makespan. Since the makespan is optimized for many samples at once, it suffices to pick only the jobs which are on at least one of the critical paths in the samples. For a large sample size and underlying distributions with large variance, there can be various critical paths, filtering only very few to none of the jobs. However, jobs on the critical paths of many samples are more likely to generate a better solution when reinserted, so we prioritize job candidate selection by the amount of samples for which the job appears on the critical path.

## 3.4 Finding the best reinsertion

The most straightforward *1-move* implementation for sampling would repeatedly pick a random job, then reinsert this job behind a valid machine predecessor and then approximate $E(C_{max})$ from the samples. In this subsection, we present an efficient technique to find the best reinsertion of a chosen job, based on the reinsertion technique of Vaessens (1995) for generalized job shop scheduling.

Consider a graph $G$ for a schedule $S$. Reinserting a job $j$ is done in two steps:

1. Let $i = mp(j)$. Delete $j$ from its machine by deleting $(i, j)$ from $A_M$. If there is a job $k$ such that $(j, k) \in A_M$, delete $(j, k)$ from $A_M$ and add $(i, k)$ to $A_M$.

2. Find a new allowed machine predecessor $i$ and insert $j$ behind $i$. A new edge $(i, j)$ is added. If there is a job $k$ such that $mp(k) = i$, replace the edge $(i, k)$ by $(j, k)$ in $A_M$.

For the remainder of this section, we assume $j$ is the job to be moved and $i$ is the new allowed machine predecessor of $j$ after step 2. The resulting

graphs after step 1 and 2 are denoted by $G^-$ and $G^i$ respectively. Note that $j$ is still connected to all its precedence predecessors and successors in $G^-$ and $G^i$.

For $G$, we denote the distance of the longest path from $s$ to $j$ excluding $j$ as the head time $h_j$ and the distance of the longest path from $j$ to $t$ including $j$ as the tail time $t_j$. The head time can be calculated recursively from the head time of the machine predecessor and precedence predecessors, and the tail time can be calculated recursively from the tail time of the machine successor and precedence successors as follows:

$$h_j = max\{r_j, h_{mp(j)} + P_{mp(j)}, max\{h_{j_p} + q_{j_pj} | j_p \in pp(j)\}\}$$

$$t_j = max\{t_{ms(j)} + P_j, max\{t_{j_s} + q_{jj_s} | j_s \in ps(j)\}\}$$

For convenience, when any of the partial expressions are not defined in these equations, they are set to 0. For example, $t_{ms(j)} + P_j = P_j$ when $j$ is the last job on a machine and $max\{h_{j_p} + q_{j_pj} | j_p \in pp(j)\} = 0$ when $pp(j)$ is empty. Head (tail) time for $G^-$ and $G^i$ are denoted as $h_j^-$ ($t_j^-$) and $h_j^i$ ($t_j^i$) respectively, where $j$ is omitted when $j$ is clear from context.

### 3.4.1 Computing critical path lengths using head and tail times

To find the best reinsertion, we need to find a new machine predecessor $i$ such that the critical path length of $G^i$ is minimal. The critical path length of $G^i$ is equal to the maximum of the longest path length from $s$ to $t$ without $j$ and the longest path length from $s$ to $t$ with $j$. Therefore, in order to find the best reinsertion of $j$, we need to compute the length of the longest path without $j$ and the longest path with $j$ for all $G^i$ given that $i$ is a valid machine predecessor for $j$. In order to find a linear time algorithm for best insertion, these measures should both be computed in linear time. The following theorem helps establishing a linear time algorithm for computing the longest path length for all reinsertions.

**Theorem 3.4.1.** *The longest path length from $s$ to $t$ containing $j$ in $G^i$ can be computed in constant time, given the head and tail times for all jobs in $G^-$.*

*Proof.* Recall that the length of the longest path from $s$ to $t$ with $j$ can be computed using the head times of all $j_p \in pp(j)$, the tail times of all $j_s \in ps(j)$ and $h_{mp(j)}$ and $t_{ms(j)}$. Since $i$ will be the new machine predecessor and $ms(i)$ will be the new machine successor of $j$ in $G^i$, we need to use $h_i^i$ and $t_{ms(i)}^i$ instead of $h_{mp(j)}$ and $t_{ms(j)}$. By definition of head time, if a path from

$j$ to another job $k$ in $G^-$ does not exist, then $h^i_k = h^-_k$. Similarly, if a path from $k$ to $j$ in $G^-$ does not exist, then $t^i_k = t^-_k$. Since $G^i$ must be acyclic, there exist no paths from $j$ to any $j_p \in pp(j)$ or from any $j_s \in ps(j)$ to $j$. Consequently, we have that $\forall j_p \in \{pp(j) \cup \{mp(j)\}\} : h^i_{j_p} = h^-_{j_p}$ and $\forall j_s \in \{ps(j) \cup \{ms(j)\}\} : t^i_{j_s} = t^-_{j_s}$. In order for $i$ to be a valid machine predecessor for $j$, there should not exist a path from $j$ to $i$ in $G^-$. Additionally, there should not exist a path from $ms(i)$ to $j$ in $G^-$. Therefore, we find that $h^i_i = h^-_i$ and $t^i_{ms(i)} = t^-_{ms(i)}$. Now, if we compute $max_{j_p \in pp(j)}(h^-_{j_p})$ and $max_{j_s \in ps(j)}(t^-_{j_s})$ beforehand, we can compute $h^i_j$ and $t^i_j$ in constant time by directly using the head and tail times of $G^-$. Consequently, the longest path from $s$ to $t$ with $j$ in $G^i$ can be computed in constant time. □

### 3.4.2 Runtime analysis

Recall from Section 2.2 that finding valid machine predecessors for $j$ can be done in $O(m + n + r)$ time. Suppose head and tail times are computed in (reverse) topological ordering so that head (times) times of all predecessors (successors) of each job are known. Now, computing the head and tail times for a single job takes $O(1 + |pp(j)|)$ or $O(1 + |ps(j)|)$ time. Therefore, computing all head and tail times in $G^-$ takes $O(n + r)$ time, since $\sum_{j \in J}(|pp(j)| + |ps(j)|) = 2r$. The longest path from $s$ to $t$ without $j$ can easily be computed in $O(n + m + r)$ time using the head and tail times of $G^-$. Since the longest path from $s$ to $t$ with $j$ in $G^i$ can be computed in constant time using the head and tail time sof $G^-$, the longest paths from $s$ to $t$ in all $G^i$ with $j$ can be computed in $O(n)$ time. The best reinsertion for $j$ is behind job $i$ for which $G^i$ has the smallest critical path length. Adding up the time required to find this job $i$, we need $O(n + m + r)$ time. In comparison, $O(m + n + r)$ time is required to evaluate the solution quality after a random insertion in the worst case. Therefore, the runtime of the best reinsertion algorithm only differs in constant time factor compared to random insertion. For this reason, we apply best insertion instead of random insertion for the result sampling implementations.

Gambardella and Mastrolilli (1996) present an even more efficient implementation, using sophisticated algorithms to further reduce the candidate set and often finding a best reinsertion in $O(\log n)$ time for the flexible job shop problem. These algorithms can also be implemented to make sampling for RPMS faster. However, since the difference in computation speed will be insignificant in practice for small $n$ and to keep the research simple, we use a simpler implementation. It should thus be noted that result sampling may be optimized by applying the algorithms of Gambardella and Mastrolilli (1996).

## 3.5 Neighbourhood operator implementations

Now that the best reinsertion of a given job $j$ can be found in $O(m+n+r)$ time, the *1-move* operator can be optimized. To approximate $E(C_{max})$, we need to calculate the critical path lengths of all $G^i$ in all samples. This takes $O(s \cdot (m+n+r))$ time, where $s$ is the amount of samples. The best machine predecessor $i$ should now have the smallest sum of critical path lengths over $G^i$ in all samples.

### 3.5.1 *1-move* and *2-move* implementation

The best *1-move* operation is one of the best reinsertions for all $j \in J$. Therefore, it takes $O(n \cdot s \cdot (m+n+r))$ time to evaluate the whole neighbourhood of the *1-move* operator.

Recall that the *2-move* operation performs two successive *1-move* operations. Suppose that $j_1$ and $j_2$ are reinserted in the first and final move, respectively. After the first move, the best reinsertion for the final move will be the best reinsertion of the *2-move*. However, it is possible that the first move of the best *2-move* operation results in a (temporarily) worse schedule than the original schedule and consequently, it is also possible that $j_1$ is not on the critical path in the original schedule. Nevertheless, it is likely that the best *2-move* operation includes a good first move. For the experiments, we only try a limited amount of *2-move* operations for the same schedule before halting the local search due to time constraints. Within this limit, we generally do not perform more than $n^2$ operations. Additionally, we note that it is generally more effective to try the best first moves. Therefore, we also use the best reinsertion technique for finding the first move, which reduces the neighbourhood size to $O(n^2)$. Now, exhaustively running the *2-move* operator takes $O(n^2 \cdot s \cdot (m+n+r))$ time. Finally, to save computation time, we do not allow to reinsert $j$ to its original position for the first move, since then, the *2-move* operation may reduce to a *1-move* operation.

### 3.5.2 *2-swap* implementation

The best reinsertion technique cannot be applied for the *2-swap* operator, since it involves moving two jobs and the fast best reinsertion technique can only be applied for a single move. Without this technique, finding the best reinsertion for a single job requires computing the critical path length of $n$ schedules. As a consequence, it is no longer more effective than random insertion. A single random insertion step of the *2-swap* operator requires $O(s \cdot (m+n+r))$ time. Since $O(n^2)$ jobs can be swapped, exhaustively running the *2-swap* operator takes $O(n^2 \cdot s \cdot (m+n+r))$ time, similar to the

*2-move* operator. The *2-move* operator is generally more effective than the *2-swap* operator for result sampling, since it performs two successive best reinsertions instead of random swaps within almost the same time. For this reason, it is questionable whether the *2-swap* operator should be included in the VND subroutine or that instead, the limit for the amount of *2-move* attempts should increase. Small-scale experiments have shown that the *2-swap* still coincidently finds better schedules than the *2-move* operator. Therefore, we have decided to include the *2-swap* operator as well in the VND subroutine for sampling, but only after the *2-move* attempt limit has been reached. The *2-move* operator is used only when the *1-move* operator is exhausted. Now, when the *2-swap* attempt limit has been reached, the VND subroutine is aborted.

# Chapter 4

# Approximating $E(C_{max})$ based on expected processing times

Many samples may be needed to approximate $E(C_{max})$ accurately for RPMS using simulation. Therefore, it may be more effective to approximate $E(C_{max})$ using an objective function on a single, deterministic instance of a schedule, which should capture properties of stochastic schedules. In this chapter, we present various ways to approximate $E(C_{max})$ using a single sample where $P_j = p_j$. In the first part of this chapter, we approximate $E(C_{max})$ by first approximating the expected makespan without precedence relations in Section 4.2 and then adding the approximated delay caused by precedence relations in Section 4.3. In Section 4.4, we approximate the probability density functions (PDFs) of all $C_j$ and then approximate $E(C_{max})$ using the completion times of the final jobs on each machine, using the technique described in Section 4.2.3. Since the computation of $X = max(D_1, D_2)$ for two normal distributions $D_1$ and $D_2$ is an important building block for the objective functions presented in Section 4.2.3 and Section 4.4, we will present two ways to compute of $E(X)$ and $\sigma^2(X)$ first in Section 4.1.

Note that the absolute difference between the outcomes of the approximation objective function and the expected makespan is not important. For example, when schedule $S$ has a lower deterministic makespan than another schedule $S'$, the objective function $c_{max}/10$ and the objective function $c_{max}$ will both conclude that $S$ is a better schedule, despite the fact that the latter function is significantly closer to the expected makespan in terms of absolute difference. Instead, the most accurate objective functions should have the strongest correlation with a 'perfect' expected makespan objective function when comparing schedules.

## 4.1 Approximating the maximum of two normal distributions

Before we present the objective functions for approximating $E(C_{max})$, we explain how we can approximate $X = max(D_1, D_2)$ for $D_1 = \mathcal{N}(\mu_1, \sigma_1^2)$ and $D_2 = \mathcal{N}(\mu_2, \sigma_2^2)$. Note that the maximum of two normal distributions is not normally distributed. Nevertheless, in order to reuse $X$ in the context of maximization of normal distributions, we assume that $X$ is normally distributed. Therefore, we are interested in computing $E(X)$ and $\sigma^2(X)$. When the correlation $\rho$ between $D_1$ and $D_2$ is known, $E(X)$ and $\sigma^2(X)$ can be computed exactly and quickly. This computation is presented in Section 4.1.1. When $\rho$ is unknown, $E(X)$ can still be computed exactly though $\sigma^2(X)$ needs to be approximated. This approximation is presented in Section 4.1.2.

### 4.1.1 Computing $E(X)$ and $\sigma^2(X)$ for known $\rho$

Let $\theta = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2}$. Furthermore, let $\Phi(x)$ and $\phi(x)$ denote the CDF and the PDF of the standard normal distribution respectively. As stated by Nadarajah and Kotz (2008), $E(X)$ and $E(X^2)$ can be computed as follows:

$$E(X) = \mu_1\Phi(\frac{\mu_1 - \mu_2}{\theta}) + \mu_2\Phi(\frac{\mu_2 - \mu_1}{\theta}) + \theta\phi(\frac{\mu_1 - \mu_2}{\theta}) \tag{4.1}$$

$$\begin{aligned} E(X^2) =& (\sigma_1^2 + \mu_1^2)\Phi(\frac{\mu_1 - \mu_2}{\theta}) + (\sigma_2^2 + \mu_2^2)\Phi(\frac{\mu_2 - \mu_1}{\theta}) \\ &+ (\mu_1 + \mu_2)\theta\phi(\frac{\mu_1 - \mu_2}{\theta}) \end{aligned} \tag{4.2}$$

Now, from probability theory, we have that $\sigma^2(X) = E(X^2) - E(X)^2$.

### 4.1.2 Computing $E(X)$ and approximating $\sigma^2(X)$ for unknown $\rho$

Let $\delta = D_2 - D_1$. We have that $E(max(D_1, D_2)) = E(D_1) + E(max(0, \delta))$. Let $\Delta = E(max(0, \delta))$ so we have $\mu(X) = \mu(D_1) + \Delta$. If $\delta > 0$ then $max(D_1, D_2) = D_2$ and otherwise, $max(D_1, D_2) = D_1$. Since $\sigma^2(X)$ depends on which of $D_1$ and $D_2$ is the greatest, we obtain:

$$\sigma^2(X) \approx Pr(\delta > 0) \cdot \sigma_2^2 + Pr(\delta \leq 0) \cdot \sigma_1^2$$

Note that when $D_1 = D_2$, we have that $\sigma^2(X) \leq \sigma^2(D_1)$ and $\sigma^2(X) \leq$

$\sigma^2(D_2)$. When $D_1$ and $D_2$ are not similar, $\sigma^2(X)$ is still slightly overestimated in this computation. Finally, it remains to compute $\Delta$ and $Pr(\delta > 0)$ (where $Pr(\delta \leq 0) = 1 - Pr(\delta > 0)$). Using probability theory, we obtain:

$$\begin{aligned}\Delta =& E(max(0, \delta)) \\ =& Pr(\delta \leq 0) \cdot E(max(0, \delta) | \delta \leq 0) \\ &+ Pr(\delta > 0) \cdot E(max(0, \delta) | \delta > 0) \\ =& Pr(\delta > 0) \cdot E(\delta | \delta > 0)\end{aligned}$$

To ensure that both values in the maximum computation are normal distributions, we replace 0 by $\mathcal{N}(0, 0)$ in the computation. When $\delta = \mathcal{N}(\mu_\delta, \sigma_\delta^2)$, we have:

$$Pr(\delta > 0) = 1 - Pr(\delta \leq 0) = 1 - \Phi(\frac{-\mu_\delta}{\sigma_\delta})$$

Let $\alpha = -\mu_\delta / \sigma_\delta$ and let $\lambda(\alpha) = \frac{\phi(\alpha)}{1 - \Phi(\alpha)}$. As explained by Greene (2003), we have

$$E(\delta | \delta > 0) = \mu_\delta + \sigma_\delta \cdot \lambda(\alpha)$$

## 4.2 Expected makespan without precedence relations

The machine causing the deterministic makespan is not the only forecaster of the expected makespan. Especially when the underlying probability distribution $D$ of $P_j$ has a large variance, the makespan in a stochastic realisation may be caused by a different machine. In this section, we present some methods of approximating the expected makespan without precedence relations by examining the probability distributions of the completion times of all machines.

### 4.2.1 Aggregated machine load (*AML*)

Let $C_{max}(m_k)$ and $c_{max}(m_k)$ denote the completion time of the final job on $m_k \in M$ in a stochastic and deterministic (i.e. $P_j = p_j$) schedule, respectively. $C_{max}(m_k)$ will be denoted as the machine load of $m_k$. Intuitively, the larger $c_{max}(m_k)$ compared to other machine loads, the greater the probability that $m_k$ causes the makespan in a stochastic instance. Therefore, it is more effective to remove jobs from machines for which $c_{max}(m_k)$ is relatively

large, since it is more likely that the expected makespan in stochastic instances decreases as a consequence. To approximate the expected makespan, we define a value $l_k$ for each $m_k$ to approximate the likeliness of $m_k$ to cause the makespan in a stochastic schedule. We assume that $l_k$ is twice as large as $l_i$ for machines $k$ and $i$ if $c_{max}(m_i) < c_{max}(m_k)$ and there are no other machines $x$ for which $c_{max}(m_i) < c_{max}(m_x) < c_{max}(m_k)$. Formally, let $l_k = 2^{-MCi(m_k)}$, where $MC$ denotes the array of machine loads in nonincreasing order and $MCi(m_k)$ denote the position of $m_k$ in $MC$, i.e. $MCi(m_k)$ is 1 if $m_k$ has the largest load and $m$ if $m_k$ has the smallest load. Now, a simple but more accurate function than the deterministic makespan for approximating $E(C_{max})$ is formulated as follows:

$$E(C_{max}) \approx \frac{\sum_{m_k \in M} l_k \cdot c_{max}(m_k)}{\sum_{m_k \in M} l_k}$$

Note that the outcome of this function is independent of machine ordering in $MC$ for machines with equal loads. Therefore, machines with equal loads are ordered arbitrarily in $MC$. We denote this objective function as the aggregated machine load ($AML$). This objective function is relatively easy to compute, but it is only a very rough approximation since the value of $l_k$ depends only on the ordering of $MCi(m_k)$ and not on the actual differences between the loads.

**Illustrative example** Consider the example presented in Figure 4.1. We have $c_{max}(m_1) = 5$ and $c_{max}(m_2) = 6$ if all jobs are scheduled as early as possible. After sorting the machine loads, we have that $MCi(m_1) = 2$ and $MCi(m_2) = 1$ and thus $c_1 = 2^{-2}$, $c_2 = 2^{-1}$ and $\sum_{m_k \in M}(c_k) = \frac{3}{4}$. We can now compute

$$E(C_{max}) \approx \frac{\frac{1}{4} \cdot 5 + \frac{1}{2} \cdot 6}{\frac{3}{4}} = 5\frac{2}{3}$$

### 4.2.2 Gaussian makespan (GM)

To precisely calculate the expected makespan, we need to calculate the PDFs of the machine loads, followed by calculating the expected maximum of these PDFs. Unfortunately, it is impractical to calculate both of these measures precisely, so we need to resort to approximation functions.
The PDF of a machine load $C_{max}(m_k)$ can be computed by aggregating the PDFs of the processing times of the jobs on $m_k$. To do this precisely, we need to compute the convolutions of these PDFs, but this is inconvenient

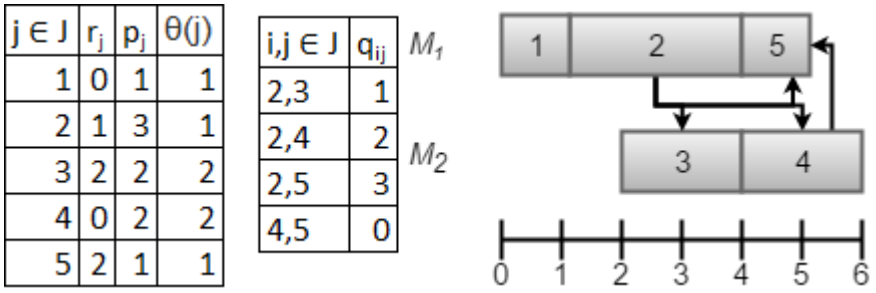



Figure 4.1: Example schedule with data

when the number of jobs per machine becomes big, where this computation becomes very time-consuming. Therefore, we approximate the PDF of $C_{max}(m_k)$ with a normal distribution, using the central limit theorem. Now, $C_{max}(m_k)$ can be approximated quickly by converting the PDFs of each $P_j$ to a normal distribution and aggregating the PDFs of each $P_j$ where $\theta(j) = k$. Let $\sigma^2(D)$ denote the variance of the underlying distribution of $P_j$, i.e. the variance of $P_j$ when $p_j = 1$. For example, $\sigma^2(D) = 1$ when $D$ is an exponential distribution or $\sigma^2(D) = 0.25$ when $D$ is a 4-Erlang distribution. Now, $\sigma(P_j) = \sigma(D) \cdot p_j$. Since we assume that there are no precedence relations involved in this section, we have that each $C_{max}(m_k)$ is independent. Therefore, the PDF of $C_{max}(m_k) \approx N(\mu(m_k), \sigma^2(m_k))$ is approximated as follows:

$$\mu(m_k) = c_{max}(m_k) \tag{4.3}$$

$$\sigma^2(m_k) = \sum_{j \in J_{m_k}} \sigma^2(P_j) = \sigma^2(D) \cdot \sum_{j \in J_{m_k}} p_j^2 \tag{4.4}$$

Here, $J_{m_k}$ is defined as the set of jobs being scheduled on machine $m_k$. Note that $c_{max}(m_k) \neq \sum_{j \in J_{m_k}} p_j$ when there is machine idle time in the deterministic schedule. Additionally, note that $\sigma^2(m_k)$ is an approximation, since precedence relations could also introduce or reduce variance on the PDF of $C_{max}(m_k)$. To approximate $E(C_{max})$ from the PDFs of the machine loads, we approximate the PDF of the maximum of the machine loads and take its mean. The mean of the maximum of machine loads can be approximated by dividing the distribution in intervals and computing the probability that $C_{max}$ lies in a given interval. Then, for each interval, we multiply the probability of the interval occurrence by the mean of the interval and finally, we sum these outcomes. Formally, consider a set of $s$ uniformly distributed values $x_1, \ldots, x_s$ representing the limits of these in-

tervals, sorted in ascending order. In Section 5.2.5, it is discussed how these values are chosen. The probability that $C_{max}$ is between two values $x_i$ and $x_{i+1}$ equals $Pr(C_{max} \leq x_{i+1}) - Pr(C_{max} \leq x_i)$ and the mean of this interval is $(x_{i+1} + x_i)/2$. Suppose $Pr(C_{max} \leq x_1) = a$ and $Pr(C_{max} \geq x_s) = b$. Now, $E(C_{max})$ is approximated as follows:

$$\begin{aligned} E(C_{max}) &= \max_{m_k \in M} C_{max}(m_k) \\ &\approx \frac{\sum_{i=1}^{s-1} (Pr(C_{max} \leq x_{i+1}) - Pr(C_{max} \leq x_i)) \cdot \frac{x_{i+1}+x_i}{2}}{1-(a+b)} \end{aligned} \tag{4.5}$$

The equation is divided by $1 - (a + b)$, since the intervals below $x_1$ and above $x_s$ are disregarded and consequently, $1 - (a + b)$ is the sum of the interval probabilities. The objective function from Equation (4.5) will be denoted as the Gaussian makespan (GM) function. Note that the *GM* function becomes more accurate for large $s$, but also more difficult to compute. Furthermore, it is important to find a good balance between the values of $s$, $a$ and $b$. If $a$ and $b$ are too small, the width of the intervals becomes relatively large which causes inaccuracy. However, if $a$ and $b$ are too large, an important part of the PDF may be disregarded. Finally, the values $a$ and $b$ should be as close as possible in order to prevent underestimation or overestimation of the average of the PDF.

In order to compute $Pr(C_{max} \leq x_i)$, we note that the following holds:

$$\begin{aligned} Pr(C_{max} \leq x) &= Pr(\forall m_k \in M : C_{max}(m_k) \leq x) \\ &= \prod_{m_k \in M} Pr(C_{max}(m_k) \leq x) \\ &= \prod_{m_k \in M} \Phi(\frac{x - \mu(m_k)}{\sigma(m_k)}) \end{aligned} \tag{4.6}$$

Here, $\Phi(x)$ denotes the CDF of the standard normal distribution, which is defined as $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \, \mathrm{d}t$.

**Illustrative example** Consider the schedule presented in Figure 4.1, where we ignore precedence relations. Let $P_j$ be distributed from a 4-Erlang distribution so that $\sigma^2(P_j) = 0.25 \cdot p_j$. We have $\mu(m_1) = 5$ and $\mu(m_2) = 6$. For the variances of $C_{max}(m_k)$, we have

$$\sigma^2(m_1) = 0.25 \cdot 1^2 + 0.25 \cdot 3^2 + 0.25 \cdot 1^2 = 2.75$$

$$\sigma^2(m_2) = 0.25 \cdot 2^2 + 0.25 \cdot 2^2 = 2$$

Now, consider 4 load values $\{x_1 = 5, x_2 = 6, x_3 = 7, x_4 = 8\}$. We have

$$Pr(C_{max} \leq 5) = \Phi(\frac{5-5}{\sqrt{2.75}}) \cdot \Phi(\frac{5-6}{\sqrt{2}}) \approx 0.5 \cdot 0.24 = 0.12$$

$$Pr(C_{max} \leq 6) = \Phi(\frac{6-5}{\sqrt{2.75}}) \cdot \Phi(\frac{6-6}{\sqrt{2}}) \approx 0.727 \cdot 0.5 = 0.363$$

$$Pr(C_{max} \leq 7) = \Phi(\frac{7-5}{\sqrt{2.75}}) \cdot \Phi(\frac{7-6}{\sqrt{2}}) \approx 0.886 \cdot 0.76 = 0.674$$

$$Pr(C_{max} \leq 8) = \Phi(\frac{8-5}{\sqrt{2.75}}) \cdot \Phi(\frac{8-6}{\sqrt{2}}) \approx 0.965 \cdot 0.921 = 0.889$$

Additionally, $a = 0.12$ and $b = 1 - 0.889 = 0.111$. Now, we can compute $E(C_{max})$ as follows:

$$\begin{aligned} E(C_{max}) &\approx \frac{(0.363 - 0.12) \cdot 5.5 + (0.674 - 0.363) \cdot 6.5 + (0.889 - 0.674) \cdot 7.5}{1 - (0.12 + 0.111)} \\ &\approx \frac{1.337 + 2.022 + 1.613}{0.769} \\ &\approx 6.492 \end{aligned} \tag{4.7}$$

Note that the value of $E(C_{max})$ measured using a million samples for this example without precedence constraint is approximately 6.507.

### 4.2.3 Iterative Gaussian makespan (IGM)

The most computationally expensive part of the *GM* function is the approximation of the maximum of machine loads when the PDFs of the machine loads have been approximated. In this subsection, we present a method to approximate the maximum of machine loads significantly faster and nearly as accurate. Similar to the *GM* function, we assume that the PDFs of the machine loads are normally distributed using Equation (4.3) and Equation (4.4). For readability, let $C_{max}(m_{\leq k})$ denote the PDF of the makespan when only regarding machines $m_1$ to $m_k$, i.e. $max_{1 \leq i \leq k}(C_{max}(m_i))$. We iteratively compute $C_{max}(m_{\leq k})$ from $C_{max}(m_{\leq k-1})$ or equivalently $C_{max}(m_{<k})$. Due to the iterative nature of this objective function, it is denoted as the iterative Gaussian makespan (*IGM*). Preserving the ordering of machines, the *IGM* function is computed as follows:

In order to compute the *IGM* function, $max(C_{max}(m_{<k}), C_{max}(m_k))$ needs to be approximated. As a simplification, we assume $C_{max}(m_{\leq k})$ is

**Algorithm 3** Iterative Gaussian makespan

**Precondition:** $C_{max}(m_1), \ldots, C_{max}(m_m)$

1: $C_{max}(m_{\leq 1}) \leftarrow C_{max}(m_1)$
2: **for** $k \in \{2, \ldots, m\}$ **do**
3: $\quad C_{max}(m_{\leq k}) \leftarrow max(C_{max}(m_{<k}), C_{max}(m_k))$
**return** $E(C_{max}(m_{\leq m}))$

normally distributed in order to save computation time. Note that for large $m$, the actual PDF of the expected maximum of machine loads differs more strongly from a normal distribution and the *IGM* function becomes less accurate as a consequence.

Since precedence relations are disregarded, all $C_{max}(m_k)$ are independent. Therefore, we can use the approximation of $max(D_1, D_2)$ presented in Section 4.1.1 for $D_1 = C_{max}(m_{<k})$, $D_2 = C_{max}(m_k)$ and $X = C_{max}(m_{\leq k})$ with $\rho = 0$.

**Illustrative example** Consider the schedule presented in Figure 4.1 with $\sigma^2(D) = 0.25$. We have $C_{max}(m_{<2}) = C_{max}(m_1) = N(\mu, \sigma^2)$ with $\mu = 5$ and $\sigma^2 = 2.75$, similar to the example of the *GM* function. Now, to compute the expected makespan, we need to compute $C_{max}(m_{\leq 2})$. Let $C_{max}(m_{<2}) = D_1 = \mathcal{N}(\mu_1, \sigma_1^2)$, $C_{max}(m_2) = D_2 = \mathcal{N}(\mu_2, \sigma_2^2)$ and $X = max(D_1, D_2) = C_{max}(m_{\leq 2}) = E(C_{max})$. Since $X = E(C_{max})$, only $E(X)$ is relevant to compute. Using $\rho = 0$, we have $\theta = \sqrt{\sigma_1^2 + \sigma_2^2 - 0} = \sqrt{2.75 + 2} \approx 2.179$ and $\frac{\mu_1 - \mu_2}{\theta} \approx \frac{-1}{2.179} = -0.459$. Now, we obtain

$$\begin{aligned} E(C_{max}) = E(X) &\approx 5 \cdot \Phi(-0.459) + 6 \cdot \Phi(0.459) + 2.179 \cdot \phi(-0.459) \\ &\approx 5 \cdot 0.323 + 6 \cdot 0.677 + 2.179 \cdot 0.359 = 6.459 \end{aligned} \tag{4.8}$$

## 4.3 Incorporating slack from precedence relations (AD)

To complete the $E(C_{max})$ approximation, we add the expected delays caused by precedence relations. Consider the realisation of a schedule $S$ where $P_j = p_j$, where $s_j$ denotes the earliest possible starting time of $S$, i.e. the longest path from $s$ to $j$ in $G$. In order to approximate expected delays caused by precedence relations, we first define the notion of expected delay:

**Definition 4.3.1.** *The* expected delay *of a job $j$ equals $E(S_j) - s_j$, i.e. the expected increase of starting time of $j$ when the processing times become stochastic.*

To illustrate how expected delay may be caused, consider $S_5$ and the precedence relation $(j_4, j_5)$ in Figure 4.1, ignoring the precedence relation $(j_2, j_5)$ for simplicity. It is clear that in a stochastic schedule, $S_5$ is either equal to $C_2$ or $S_4$, depending on which of these times is greater. In other words, $S_5 = max(C_2, S_4)$. Now, the following holds:

$$E(S_5) = E(max(C_2, S_4)) > E(C_2) = s_5$$

To conclude, we have $E(S_5) > s_5$ and consequently, expected delay occurs. In general, expected delay of $j$ occurs when at least two machine and/or precedence predecessor relations are involved for $j$, i.e. when $j$ has at least two precedence predecessors or a machine predecessor and a single precedence predecessor. Note that when a job $j$ gets delayed, it may in turn also delay descendants of $j$. Also note that expected delay is never negative.

Recall that Van Roermund (2013) introduced the notion of fixations, where two jobs $i$ and $j$ are fixated if $(i, j) \in P$ and $\theta(i) = \theta(j)$. When jobs are fixated, they cannot delay jobs on other machines and therefore, the total expected delay will be lower. Van Roermund (2013) minimizes expected delay by maximizing the amount of fixations. In this section, we attempt to define a more accurate notion of expected delay.

If $(i, j) \in P$ and $\theta(i) \neq \theta(j)$, it is important to consider the difference between $s_j$ and $s_i$. The expected delay depends on the probability that delay occurs and the average delay impact. The delay probability depends on $s_j - s_i - q_{ij}$ and $\sigma(D)$, where delay probability is maximal when $s_j - s_i - q_{ij}$ is minimal and delay probability decreases more steeply for low $\sigma(D)$. To approximate the delay probability, we need a function that steeply decreases for small $s_j - s_i - q_{ij}$ and approaches 0 for large $s_j - s_i - q_{ij}$. Based on the cost function for idle times between two consecutive flights introduced by Diepen et al. (2013) in the Schiphol gate assignment problem, we use the function $f(x) = \frac{\pi}{2} - arctan(x)$. Finally, we need to approximate delay impact, which is correlated with $\sigma^2(D)$, since large variance causes large expected delays. Let $P_f = \{(i, j) \in P | \theta(i) \neq \theta(j)\}$ denote the set of non-fixated precedence relations. Recall that this set of precedence relations cause probability of delay propagation. Therefore, the expected delay $d(i, j)$ of $j$ caused by a single predecessor $i \in pp(j)$ is defined as follows:

$$d(i, j) = \frac{\pi}{2} - arctan(\frac{s_j - s_i - q_{ij}}{\sigma(D)} \cdot \beta)) \tag{4.9}$$

Here, $\beta$ scales the slope of the arctan function. An example plot is drawn in Figure 4.2. Note that the effect of delay propagation does not linearly depend on the amount of predecessors of a job, since the probability that new predecessors cause delay decreases when there are other predecessors that may cause delay. Therefore, we assume that the total delay depends on the square root of the sum of squares of delay. Now, the delay function is defined as follows:

$$\sigma(D) \cdot \alpha \cdot \sum_{j \in J} \sqrt{\sum_{(i,j) \in P_f \cap pp(j)} d(i,j)^2} \tag{4.10}$$

This function will be denoted as the arctan delay (AD) function. Here, $\alpha$ scales the maximum penalty. Note that a delayed job early in the schedule has a bigger delay propagation probability, whereas the delay of job later in the schedule is often larger since the starting time uncertainty of its predecessors is greater. Since these drawbacks cancel out, we have chosen not to involve starting time in the slack function.

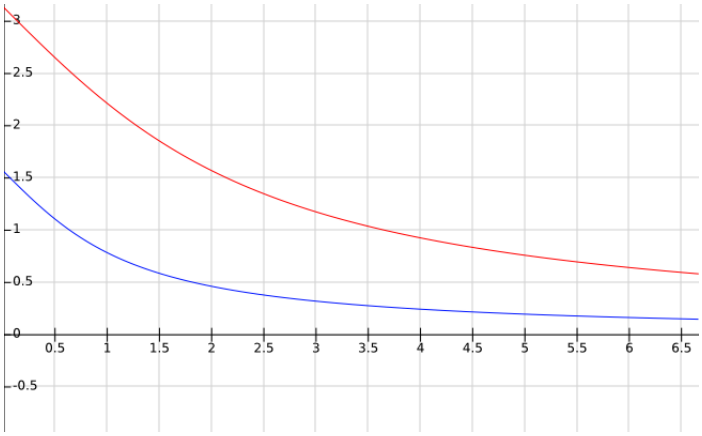



Figure 4.2: The $AD$ function for $\alpha = 1$, $\beta = 1$ and $\sigma(D) = 2$ (red) and $\sigma(D) = 1$ (blue). The x-axis represents $s_j - s_i - q_{ij}$.

Finally, we note that fixated precedence relations may also cause expected delay. For example, the precedence relation $(j_2, j_5)$ causes idle time on $M_1$ when $P_2 < q_{2,5}$ in Figure 4.1. However, this effect is generally less significant than expected delay caused by non-fixated precedence relations and is therefore not included in the $AD$ function.

**Illustrative example** Consider the schedule presented in Figure 4.1. Let $\alpha = 0.3$, $\beta = 0.3$ and $\sigma(D) = 0.5$. First, we compute $arctan(\frac{s_j - s_i - q_{ij}}{\sigma(D)} \cdot \beta)$ for each non-fixated precedence relation $(i,j) \in P_f$:

$$(2,3) : arctan(\frac{2-1-1}{0.5} \cdot 0.5) = 0$$
$$(2,4) : arctan(\frac{4-1-2}{0.5} \cdot 0.5) \approx 0.54$$
$$(4,5) : arctan(\frac{4-4-0}{0.5} \cdot 0.5) = 0$$

Now, the $AD$ function equals

$$0.5 \cdot 0.3 \cdot (\sqrt{(\frac{\pi}{2} - 0)^2 + (\frac{\pi}{2} - 0.54)^2} + \sqrt{(\frac{\pi}{2} - 0)^2}) \approx 0.15 \cdot 3.967 \approx 0.595$$

When the outcome of the $AD$ function is added to the outcome of the $GM$ function, we get an approximation of the expected makespan with an approximation of delay caused by precedence relations included. The final approximated value of the expected makespan becomes $6.492 + 0.595 \approx 7.087$. The value of $E(C_{max})$ measured using a million samples is approximately 6.735 so in this case, the $AD$ function overestimates the expected delay caused by precedence relations. One could decrease the value of $\alpha$ for a more accurate approximation for this schedule, though this may cause underestimation of $E(C_{max})$ for other schedules when the same value of $\alpha$ is used.

## 4.4 Approximating completion times using dynamic programming (DM)

A large number of notations are used and introduced in this section. In order to clarify the notations used to define the objective function introduced in this section, a summary of the notations is given in Table 4.1.

### 4.4.1 Shortcomings of arctan delay

Though the $AD$ function incorporates more elements of expected delay caused by precedence relations than the maximization of fixations, it is still a rather rough estimate. In Figure 4.3, two example schedules are illustrated which are equally evaluated by the $AD$ function, since $s_j - s_i - q_{ij}$ is equal in both schedules for all precedence relations. However, $E(C_{max})$ differs significantly for these schedules. In Figure 4.3a, $S_4$ is equal to $C_2$ if $S_3 \leq C_2$, whereas $S_4$ is always equal to $S_3$ in Figure 4.3b. Therefore, expected delay occurs exclusively in Figure 4.3a.

| Terminology | Definition and explanation |
|---|---|
| $S_j^k$ | An approximation of $j$ using $mp(j)$ and the first $k$ precedence relations in the given ordering of $pp(j)$. |
| $\delta$ | $S_i + q_{ij} - S_j^{k-1}$ for precedence relation $(i, j)$; used in order to approximate $S_j^k$. Measures the difference between $S_i + q_{ij}$ and the previous approximation of $S_j$. |
| $\Delta$ | $Pr(\delta > 0) \cdot E(\delta \mid \delta > 0)$; measures expected delay caused by $i \in pp(j)$ relative to $S_j^{k-1}$. |
| $D(x)$ | A PDF following the distribution of $D$ with mean $x$, e.g. $D(4)$ is equal to $P_j$ for a job $j$ such that $p_j = 4$. |
| $D(i \rightarrow j)$ | $D(\mu(S_j^{k-1}) - \mu(S_i))$, i.e. the PDF of the time between the start of $j$ and the start of $i$. |
| $D^{<x}$ | The partial distribution of $D$ where the domain of $D$ is restricted such that $Pr(D^{<x} \geq x) = 0$. In other words, the PDF of $D$ truncated after $x$. |

Table 4.1: Notations used to define the $DM$ function

The $AD$ function does not take precedence relations on the same machine into account. However, in both figures, the precedence relation $(1, 3)$ causes an expected delay of $j_3$, since $S_3 = max(1, C_1)$ now and $E(S_3) > 1$ as a consequence. Additionally, in Figure 4.3a, it is more likely that $j_3$ causes delay of $j_5$ compared to Figure 4.3b. Moreover, the expected delay of $j_3$ and $j_4$ in Figure 4.3a causes extra expected delay of $j_5$ when delay is propagated. We also observe that delay propagation of $j_3$ has more effect in Figure 4.3a than in Figure 4.3b, since there is idle time between $j_3$ and $j_5$ in Figure 4.3b. Finally, we observe that the expected delay of $j_5$ will be greater than the expected delay of $j_3$, since more uncertainty is involved in the starting and completion time of the predecessors of $j_5$. We can now conclude that in order to accurately approximate slack caused by a precedence relation $(i, j) \in P$, the expected delay of $S_i$, the expected completion time of $mp(j)$ and the PDFs of $C_{mp(j)}$ and $S_i$ should also be taken into account and it may also be important to take precedence relations on the same machine into account.

### 4.4.2 Dynamic makespan framework

If we assume that the PDF of each job is normally distributed, similar to the assumption of the $GM$ and $IGM$ functions, we can approximate the PDF of $S_j$ and $C_j$ by aggregating approximated PDFs of the predecessors of $j$

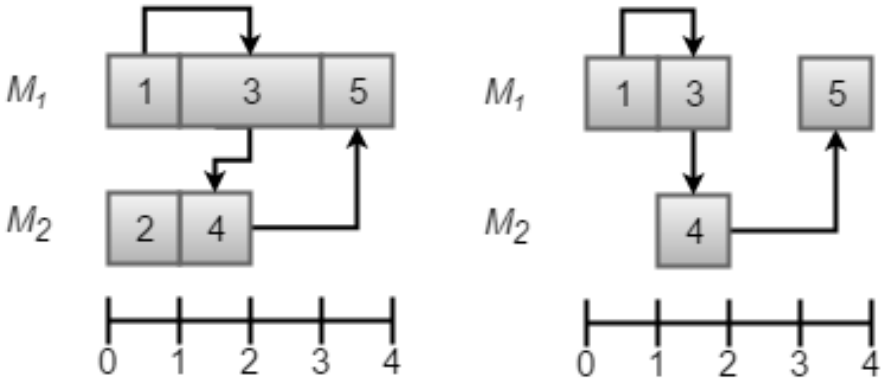



Figure 4.3: Two deterministic instances of example schedules with 5 jobs, 2 machines and 3 precedence relations $(j_1, j_3)$ with $q_{1,3} = 1$, $(j_3, j_4)$ with $q_{3,4} = 0$ and $(j_4, j_5)$ with $q_{4,5} = 2$. In schedule (b), job 2 is removed and $p_3 = 1$, whereas $p_3 = 2$ in (a). All jobs have $r_j = 0$.

using the central limit theorem. Since the PDF of $S_j$ depends on $r_j$, $C_{mp(j)}$ and $S_i$ for all $i \in pp(j)$, it can be computed as follows:

$$S_j = max(r_j, C_{mp(j)}, max_{i \in pp(j)}(S_i + q_{ij}))$$

Obviously, we have $C_j = S_j + P_j$. Non-existent values are replaced by 0 in the equation (e.g. $C_{mp(j)} = 0$ when $j$ is the first job on the machine). In order to compute each $S_j$ and $C_j$, the values for $C_{mp(j)}$ and all $S_i$ for $i \in pp(j)$ should be known. Therefore, $S_j$ and $C_j$ are computed for jobs in topological ordering from the graph representation introduced in Section 1.1.1. When the PDFs of all jobs are known, we can compute their expected maximum with the functions introduced in Section 4.2. Since dynamic programming is used in order to compute $S_j$ and $C_j$, we denote this objective function as the dynamic makespan ($DM$) function.

Since we have assumed that all these PDFs are normally distributed, we can approximate $S_j$ similar to the approximation of $E(C_{max})$ with the $IGM$ function:

For readability, let $i = i_k$. We order the precedence relations such that $i_1 = mp(j)$ if $(mp(j), j) \in pp(j)$ and the precedence relations $(i, j)$ for which $\theta(i) = \theta(j)$ are handled before the precedence relations for which $\theta(i) \neq \theta(j)$. An explanation for this specific ordering is given in Section 4.4.3 Now, when we assume all $S_j^k$ are normally distributed, the approximation of $S_j$ becomes a series of maximum approximations between pairs of normal distributions. For $max(r_j, S_j^{|pp(j)|})$, we have that $r_j = \mathcal{N}(r_j, 0)$ is independent of $S_j^{|pp(j)|}$.

**Algorithm 4** $S_j$ computation for dynamic makespan

**Precondition:** $C_{mp(j)}$ (optional); ordered set of $i \in pp(j)$: $\{i_1, \ldots, i_{|pp(j)|}\}$, $\forall i \in pp(j) : \mu(S_i), \sigma^2(S_i)$

1: **if** $j$ is the first job on its machine **then**
2: $\quad S_j^0 \leftarrow 0$
3: **else**
4: $\quad S_j^0 \leftarrow C_{mp(j)}$
5: **for** $k \in \{1, \ldots, |pp(j)|\}$ **do**
6: $\quad S_j^k \leftarrow max(S_j^{k-1}, S_{i_k} + q_{i_k j})$
**return** $max(r_j, S_j^{|pp(j)|})$

Furthermore, when $\theta(i) \neq \theta(j)$, we assume that $S_i$ and $S_j^{k-1}$ are independent in order to save computation time. Therefore, we can use the approximation of $max(D_1, D_2)$ presented in Section 4.1.1 for these two cases using $\rho = 0$. Note that the accuracy of the $DM$ function can be improved by reasoning that $S_i$ and $S_j^{k-1}$ are dependent in some way, at the expense of run time. Since dependency assumptions of $S_i$ and $S_j^{k-1}$ can effectively be made when $\theta(i) = \theta(j)$, the computation of $max(S_i + q_{ij}, S_j^{k-1})$ becomes more complicated in that case.

### 4.4.3 Computation of $max(S_i + q_{ij}, S_j^{k-1})$ when $\theta(i) = \theta(j)$

Since the correlation between $S_i$ and $S_j^{k-1}$ is unknown when $\theta(i) = \theta(j)$, we will use the $max(D_1, D_2)$ approximation presented in Section 4.1.2. In order to do this, it remains to compute $\delta$. Recall that all $S_i$ and $C_{mp(j)}$ are known since all $S_j$ are computed in topological ordering of the graph representation. For this computation, we distinguish two cases:

- $mp(j) = i$. Since $i_1 = mp(j)$, we have $S_j^{k-1} = C_{mp(j)}$ and $C_{mp(j)} - S_i = P_i$ so that $\delta_j^k = q_{ij} - P_i$ and hence, $\delta_j^k$ is of the same probability distribution class as $D$. Note that this assumption cannot be made when $mp(j) \neq i_1$ since then, the PDF of $S_j^{k-1}$ depends on more than $C_{mp(j)}$ and $S_j^{k-1} - S_i \neq P_i$. Since $\delta$ may not be normally distributed now, we may have to compute $Pr(\delta > 0)$ and $E(\delta | \delta > 0)$ differently. The computation of these values when $D$ is not normally distributed is given in Section 4.4.4.

- $mp(j) \neq i$. Since $S_j^k$ computations for $\theta i = \theta j$ are performed first, we assume that $S_j^{k-1} - S_i = P_i + \ldots + P_{mp(j)}$ so that the variance of $\delta_{ij}$ only depends on the variance of processing times of jobs between $i$ and

$j$. Note that when $S_j^{k-1}$ is computed using precedence predecessors on different machines, the dependence between $S_j^{k-1}$ and $S_i$ becomes much weaker in general. Now, we can approximate $\delta$ in two different ways.

If we assume that $P_i + \ldots + P_{mp(j)}$ is normally distributed, we have $\mu(\delta_{ij}) = \mu(S_i) + q_{ij} - \mu(S_j^{k-1})$ and $\sigma^2(\delta_j^k) = \sigma^2(P_i) + \ldots + \sigma^2(P_{mp(j)})$.

Alternatively, $\delta_j^k$ could be expressed using the same PDF class as $D$. Let $D(x)$ be the PDF generated from $D$ with mean $x$, e.g. $P_j = D(p_j)$. Let $D(i \to j) = D(\mu(S_j^{k-1}) - \mu(S_i))$, i.e. the aggregated PDF of the jobs between $i$ and $j$ Now, if we assume that $P_i + \ldots + P_{mp(j)} = D(i \to j)$, then $S_j^{k-1} - S_i = D(i \to j)$ so that $\delta = q_{ij} - D(i \to j)$.

Since summing PDFs of jobs reduces the variance relative to the mean, $D(i \to j)$ will have a significantly greater variance than the real PDF of $P_i + \ldots + P_{mp(j)}$. As a consequence, assuming $C_{mp(j)} - S_i = D(i \to j)$ may be more inaccurate than assuming $P_i + \ldots + P_{mp(j)}$ is normally distributed, especially when $C_{mp(j)} - C_i$ is large. Therefore, we choose the expression where $\delta$ is normally distributed here.

### 4.4.4 Calculating $Pr(\delta > 0)$ and $E(\delta|\delta > 0)$ when $\delta$ is not normally distributed

Let $\delta = q_{ij} - P_i$. First, we rewrite the expression $Pr(\delta > 0)$.

$$\begin{aligned} Pr(\delta > 0) &= Pr(q_{ij} - P_i > 0) \\ &= Pr(q_{ij} > P_i) \\ &= Pr(P_i < q_{ij}) \end{aligned} \tag{4.11}$$

In order to compute $Pr(\delta > 0)$ and $E(\delta|\delta > 0)$, we compute $Pr(P_i < q_{ij})$ and $E(P_i|P_i < q_{ij})$.

In the experiments we assume that $D$ is a uniform distribution, a normal distribution, an exponential distribution or an Erlang distribution. If $D$ is normally distributed, $E(\delta|\delta > 0)$ can be directly computed as described in Section 4.1.2, using $\mu_\delta = q_{ij} - p_i$ and $\sigma_\delta^2 = \sigma^2(P_i)$. For the other distributions, $Pr(P_i < q_{ij})$ can be computed using the CDF of $D$ parametrized with $q_{ij}$. Let $E(P_i^{<q_{ij}}) = E(P_i|P_i < q_{ij})$ for convenience. We explain how $E(P_i^{<q_{ij}})$ is calculated using $P_i^{<q_{ij}}$ for each distribution.

**Uniform distribution** Consider the case that $P_i$ is uniformly distributed with lower bound $a$ and upper bound $b$, e.g. $a = 0.8 \cdot p_i$ and $b = 1.2 \cdot p_i$. In order to compute $E(P_i^{<q_{ij}})$, we distinguish three cases:

- $a \geq q_{ij}$: now, the entire range from $a$ to $b$ is not included in $P_i^{<q_{ij}}$. Therefore, $Pr(P_i < q_{ij}) = 0$ and the value of $E(P_i^{<q_{ij}})$ becomes irrelevant.

- $a < q_{ij}$ and $b \geq q_{ij}$: since $P_i$ is truncated after $q_{ij}$, $\mu(P_i^{<q_{ij}})$ is the mean of $a$ and $q_{ij}$, i.e. $\frac{a+q_{ij}}{2}$.

- $a < q_{ij}$ and $b < q_{ij}$: now, the mean of $P_i^{<q_{ij}}$ is equal to the mean of $P_i$ which is $\frac{a+b}{2}$.

**Exponential distribution** Let $\lambda$ denote the mean of $P_i$. As stated and proven by Olive (2008), if $Y \sim TEXP(\lambda, b = k\lambda)$ for truncation point $b$, then $E(Y) = \lambda(\frac{1-(k+e)e^{-k}}{1-e^{-k}})$. Therefore, if $k = q_{ij}/\lambda$, we have

$$E(P_i^{<q_{ij}}) = \lambda(\frac{1-(k+1)e^{-k}}{1-e^{-k}})$$

**Erlang distribution** Unfortunately, we have not found a way to easily compute $E(P_i^{<q_{ij}})$ exactly. A naive way to approximate this value is by resorting to a caching table $T$ containing several pairs of $(x, E(P_i^{<x})$ and finding the entry in $T$ such that $x$ is as near as possible to $q_{ij}$. A huge drawback of this approach is that a new table would need to be generated for every distinct $P_i$. Let $y = q_{ij}/p_i$. In order to improve this approach, we note that $E(P_i^{<q_{ij}}) = p_i \cdot E(D(1)^{<y})$. Now, we can resort to a caching table $T$ containing several pairs of $(x, E(D(1)^{<x}))$ instead, only requiring a distinct table $T$ for each value of $k$ in an Erlang-$k$ distribution. Furthermore, to improve precision, we look up the greatest smaller entry and the smallest greater entry in $T$ relative to $y$ instead of looking up the entry $x$ such that $x$ is as near as possible to $y$. To be precise, we look up two entries $(a, E(D(1)^{<a})) \in T$ and $(b, E(D(1)^{<b})) \in T$ such that $a \leq y \leq b$ and there is no entry $(x, E(D(1)^{<x})) \in T$ such that $a < x < y$ or $y < x < b$. First, suppose there exists no such $b$, that is, $y$ is greater than the greatest key of $T$. In this case, we assume $E(D(1)^{<y}) \approx 1$, since then, the truncated mean of the Erlang distribution approaches the mean of the whole Erlang distribution. Otherwise, we assume that $E(D(1)^{<y})$ is a linear combination of $E(D(1)^{<a})$ and $E(D(1)^{<b})$. Suppose $y = c \cdot a + (1-c) \cdot b$ for some $0 \leq c \leq 1$. Now, we have

$$\begin{aligned} E(P_i^{<q_{ij}}) &= p_i \cdot E(D(1)^{<y}) \\ &\approx p_i \cdot (c \cdot E(D(1)^{<a}) + (1-c) \cdot E(D(1)^{<b})) \end{aligned} \tag{4.12}$$

When the amount of entries in $T$ and the upper bound of $b$ is large enough, this approach should give a very efficient and accurate approximation of $E(P_i^{<q_{ij}})$ when $D$ is an Erlang distribution.

### 4.4.5 An illustrative example

Consider the schedule presented in Figure 4.1. Suppose $D$ is a 4-Erlang distribution so that $\sigma(D) = 0.5$. In this example, we only compute the approximation of $S_5^2$ and assume the required approximated PDFs of $S_2, C_2$ and $S_4$ have been precomputed as such:

$$\mu(S_2) \approx 1.199; \sigma^2(S_2) = 0.292$$
$$\mu(C_2) \approx 4.199; \sigma^2(C_2) = 2.335$$
$$\mu(S_4) \approx 4.335; \sigma^2(S_4) = 0.789$$

First, we order the precedence relations such that $i_1 = j_2$ and $i_2 = j_4$. We start with $S_5^0 = C_2 \approx \mathcal{N}(4.199, 2.335)$. Since $j_2 = mp(5)$, we approximate $S_5^1$ using Section 4.1.2 and $\delta = q_{25} - P_2 = 3 - D(3)$. We have $Pr(\delta > 0) = Pr(P_i < q_{ij}) = Pr(D(3) < 3) \approx 0.567$. To approximate $\mu(P_i^{<q_{ij}})$, we have $y = q_{ij}/\mu(P_i) = 1$. Since $D$ is an Erlang distribution, there is no easy way to compute $\mu(D(i \longrightarrow j))$ exactly. Therefore, we look up the entries $(a, \mu(D(1)^{<a})) \in T$ and $(b, \mu(D(1)^{<b})) \in T$ in the caching table $T$ so that $a \leq y \leq b$ and there exists no entry $(x, \mu(D(1)^{<x})) \in T$ such that $a < x < y$ or $y < x < b$. Suppose $a = 0.99$ and $b = 1.02$, so we find the entries $(0.99, 0.6516)$ and $(1.02, 0.6656)$. We have $y = \frac{2}{3}a + (1 - \frac{2}{3})b$, therefore

$$\mu(D(3)^{<3}) = \mu(D(3)) \cdot \mu(D(1)^{<1}) \approx 3 \cdot (\frac{2}{3} \cdot 0.6516 + \frac{1}{3} \cdot 0.6656) = 1.969$$

It follows that $E(\delta|\delta > 0) \approx 3 - 1.969 \approx 1.031$ and therefore, $\Delta = Pr(\delta > 0) \cdot E(\delta|\delta > 0) \approx 0.567 \cdot 1.031 \approx 0.585$ so that

$$\begin{aligned} \mu(S_5^1) &= \mu(S_5^0) + \Delta \\ &\approx 4.199 + 0.585 \approx 4.784 \end{aligned} \tag{4.13}$$

Furthermore, we have

$$\begin{aligned} \sigma^2(S_5^1) &= Pr(\delta > 0) \cdot \sigma^2(S_2) + Pr(\delta \leq 0) \cdot \sigma^2(C_2) \\ &\approx 0.567 \cdot 0.292 + 0.433 \cdot 2.335 \approx 1.061 \end{aligned} \tag{4.14}$$

Now, we incorporate the precedence relation $(4, 5)$ to obtain $S_5^2$. We have $\theta(4) \neq \theta(5)$. Using Section 4.1.1, $\rho = 0$, $D_1 = S_5^1 = \mathcal{N}(4.784, 1.061)$ and $D_2 = S_4 + q_{45} = \mathcal{N}(4.335, 0.789)$, we have $\theta = \sqrt{\sigma_1^2 + \sigma_2^2 - 0} \approx \sqrt{1.061 + 0.789} \approx 1.36$ and $\frac{\mu_1 - \mu_2}{\theta} \approx \frac{4.784 - 4.335}{1.36} \approx 0.33$. Now, we obtain

$$\begin{aligned}
E(X) &\approx 4.784 \cdot \Phi(0.33) + 4.335 \cdot \Phi(-0.33) + 1.36 \cdot \phi(0.33) \\
&\approx 4.784 \cdot 0.629 + 4.335 \cdot 0.371 + 1.36 \cdot 0.378 \approx 5.131 \\
E(X^2) &\approx (1.061 + 4.784^2) \cdot 0.629 + (0.789 + 4.335^2) \cdot 0.371 + (4.784 + 4.335) \cdot 1.36 \cdot 0.378 \\
&\approx 27.02 \\
\sigma^2(X) &= E(X^2) - E(X)^2 \approx 27.02 - 5.131^2 \approx 0.683
\end{aligned} \tag{4.15}$$

To conclude, we obtain $S_5^2 = \mathcal{N}(5.131, 0.683)$.

# Chapter 5

# Experiments and results

In this chapter, we compare the algorithms presented in the previous chapters by running them on a set of problem instances. In the first section, we present the problem instances and settings. In order to properly compare these algorithms, we present experimental results to find the right parameters in the second section. Using these parameters, we perform the final experiments in the third section.

## 5.1 Problem instances and general setup

The problem instances are randomly generated similar to the problem instances used by Van Roermund (2013). Instances are titled as $n$J-$r$R-$m$M, e.g. 30J-15R-4M is a problem instance with 30 jobs, 15 relations and 4 machines. The processing times $p_j$ are natural numbers between 1 and 20 and the release dates $r_j$ are natural numbers between 0 and $\lfloor n/2 \rfloor$. The precedence relations are randomly selected between pairs of jobs such that no cycle occurs in $G$ and each $q_{ij}$ is a natural number between 0 and $p_i$.

### 5.1.1 Distributions

For the experiments, we use the following four probability distributions for $D$.

- The uniform distribution $[0.8 \times p_j, 1.2 \times p_j]$ or *Uniform(20)* shorthand, where 20 denotes the offset between $p_j$ and the distribution limits in terms of percentage of $p_j$.
- The normal distribution $\mu(P_j) = p_j, \sigma(P_j) = 0.3 \times p_j$ or *Normal(30)* shorthand, where 30 denotes the standard deviation in terms of percentage of $p_j$.

- The Erlang distribution with shape parameter $k = 4$ and rate parameter $\lambda = 4/p_j$ or *4-Erlang* shorthand.
- The exponential distribution with $\lambda = 1/p_j$.

For the parameter experiments, we only use the *Uniform(20)* and exponential distributions, since these are the most diverse. For the final experiments, we use all distributions.

### 5.1.2 General set-up

All experiments are run using ILS with a VND subroutine. We use the notion of ILS($x$) to express that we perform $x$ perturbations in an ILS algorithm. Note that ILS(0) denotes a single VND run. To approximate $E(C_{max})$ after (part of) a run, we generate a large amount of samples and take the average makespan of these samples. In the experiments, we use 10,000 samples after a VND subroutine of the ILS algorithm in order to decide whether the ILS algorithm continues with the current solution or the previously found best solution. The same 10,000 samples are used throughout the entire ILS run after each perturbation, effectively applying *Common Random Numbers* to minimize comparison bias as explained by Kelton and Law (2000). To approximate the quality of the final schedule, 100,000 samples are used. These samples are preserved throughout all experiments within a problem instance to guarantee a fair comparison. For result sampling, we use the neighbourhood order *1-move*, *2-move*, *2-swap* as described in Section 3.5.2 and for approximation objective functions, we use the neighbourhood order *1-move*, *2-swap*, *2-move*.

## 5.2 Parameter experiments

In order to find the right parameters, we run multiple experiments for one or more problem instances while varying a single parameter and choose either the best performing parameter or a parameter value with a good performance/run time ratio. For each parameter experiment, the default values of the other parameter experiments are used unless otherwise stated. The distributions $D = Uniform(20)$ and $D = Exponential$ are used to enhance comparison variety. We repeat each experiment 10 to 30 times to improve accuracy, that is, we run each experiment 10 to 30 times using a different random seed every run. For most experiments, we only use small problem instances with 30 jobs, since we assume that for most experiments, similar conclusions can be drawn from larger problem instances when the amount of jobs, relations and machines are scaled up linearly. However, to

maximally support our conclusions, the problem instances are picked to be as diverse as possible in terms of used distribution $D$ and values of $r$ and $m$.

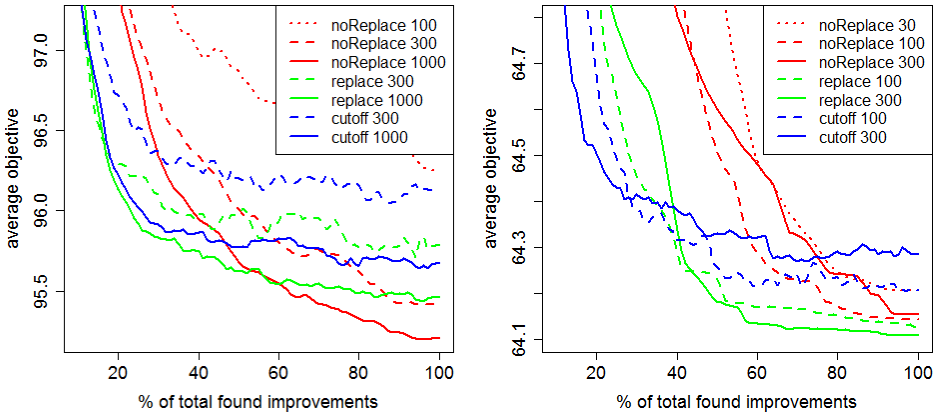
### 5.2.1 Replacement of samples in result sampling

Recall that in Section 3.1, we stated that we do not replace samples in a VND subroutine for result sampling. To support this decision, we compare sampling with no sample replacement (*noReplace*), sampling with sample replacement after every improvement (*replace*) and sampling with sample replacement after every improvement and cutoff sampling (*cutoff*). The amount of samples used is denoted in the legend as the number behind the sampling algorithm, e.g. *noReplace* 100 denotes that samples are not replaced after every improvement and 100 samples are used. For cutoff sampling, we eliminate 20% of the samples with greatest $c_{max}$ and 30% of the samples with smallest $c_{max}$, effectively eliminating half of the samples. Therefore, for cutoff sampling, the presented amount of samples is the amount of samples *after* cutoff sampling is performed, e.g. *cutoff* 100 denotes that originally, 200 samples are used but 100 samples are left over after performing cutoff sampling. Since the *replace* and *cutoff* algorithms do not converge, we stop when we have found 100 improvements. It generally takes 20-80 improvements to find a local optimum for *noReplace*. The problem is run 30 times using a 30J-40R-5M problem instance and ILS(0) is used. After each schedule improvement (according to the used objective function for approximating $E(C_{max})$), the quality of a schedule is measured by running the schedule with 30,000 samples for the exponential distribution and 10,000 samples for the uniform distribution.

In Figure 5.1, the performance of *noReplace*, *replace* and *cutoff* is shown after a certain percentage of total improvements is found. In Figure 5.1a, the exponential distribution is used and in Figure 5.1b, the uniform distribution is used. We have chosen to use percentage of total improvements instead of the absolute amount of total improvements, since the amount of improvements found before the algorithm stops may vary significantly for *noReplace* and consequently, the results will be inaccurate for the largest amount of improvements since only few runs reach that amount of improvements. From Figure 5.1a, it becomes apparent that after the maximum percentage of improvements, *noReplace* clearly performs better than the other sampling algorithms with the same amount of samples used. Furthermore, *cutoff* has the worst performance. From Figure 5.1a and Figure 5.1b, it can be seen that no significant improvement is found after finding $\pm 50$ improvements for *replace* and *cutoff*. Since the uniform distribution with 20% offset is used in Figure 5.1b, fewer samples are used than for Figure 5.1a, since less wildly

distributed samples require fewer samples for accurate $E(C_{max})$ approximation. In Figure 5.1b, the performance difference is much less obvious even though fewer samples are used and 30 runs have been performed. Cutoff sampling still seems to perform slightly worse, but *noReplace* and *replace* seem to be nearly as effective.

For *noReplace*, it should be noted that even though the found local minimum is overfit on the used set of 30 to 1000 samples, the schedule quality still consistently improves on the independent set of 30000 or 10000 samples even after more than 80% of improvements are found, especially when $D = Exponential$ and 1000 samples are used. Since the final improvements will often not be found when only using a single neighbourhood instead of VND, we can conclude that for RPMS with *noReplace* result sampling, VND is significantly more effective than using a single neighbourhood when many samples are used.

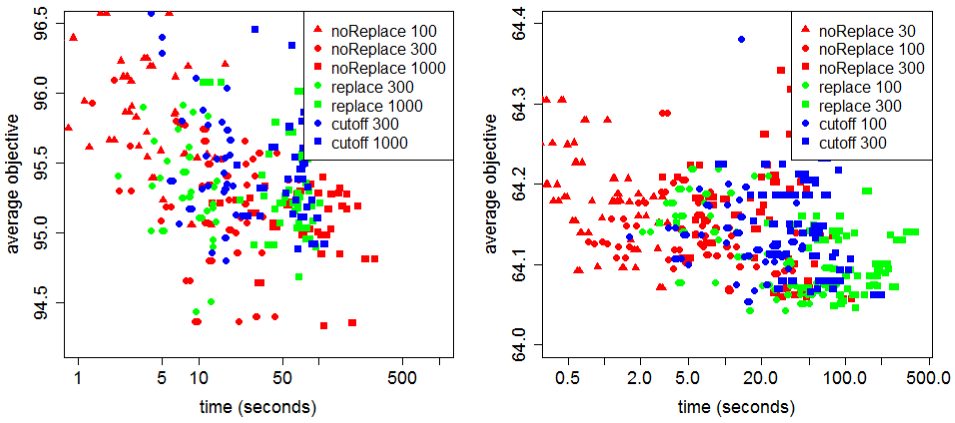



(a) $D = Exponential$ (b) $D = Uniform(20)$

Figure 5.1: The performance of various result sampling algorithms showing the average objective found after $x\%$ of improvements is found before the VND subroutine stops.

Figure 5.2 shows the run time needed to find the solution for the 30 best found solutions for each algorithm used, where the exponential distribution is used in Figure 5.2a and the uniform distribution is used in Figure 5.2b. From Figure 5.2, it can be seen that the run time of the result sampling algorithms is similar when the same amount of samples is used. The run time variance is larger for *noReplace* than for the other algorithms, since the amount of improvements found by *noReplace* before finding a local optimum may vary. From the Pareto frontier in Figure 5.2a, it can be seen

that *noReplace* is slightly more effective than the other algorithms. In Figure 5.2b, there is no significant performance difference, however it should be noted that *noReplace* with 100 samples is generally faster than the other algorithms, whereas the schedule quality is only slightly worse. Note that horizontal sequences of green and blue points in Figure 5.2b appear when the best solution is found multiple times in a single run.




(a) $D = Exponential$ (b) $D = Uniform(20)$

Figure 5.2: The value of the best solution found and the time needed to find this solution for each single run.

From Figure 5.1 and Figure 5.2, we can conclude that *noReplace* is the most effective algorithm for result sampling. Therefore, we choose to use *noReplace* for result sampling in the next experiments.

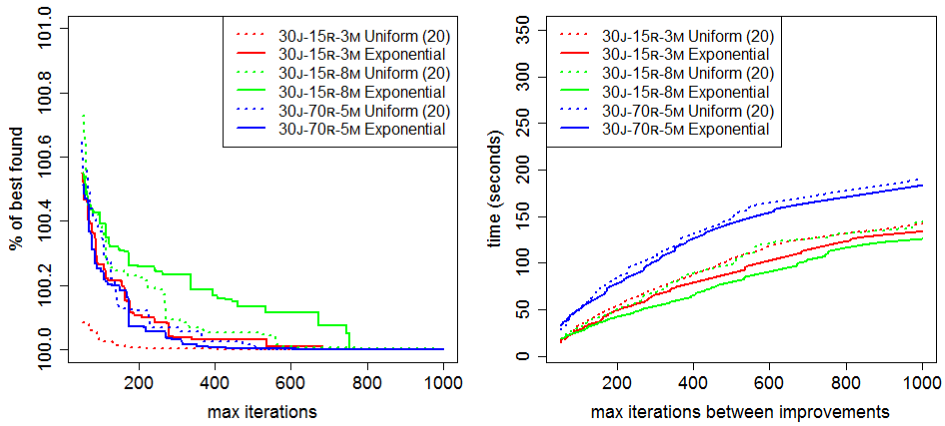
### 5.2.2 VND stop criterion

Recall that the VND subroutine is stopped when no improvement is found after a certain amount of neighbourhood moves is performed. When too few moves are performed, the VND subroutine stops while significant improvements can be made without using much more time. When too many moves are performed, much time will be used only to find minor improvements or no improvements at all. A good maximum amount of neighbourhood moves should find a solution that is nearly as good as when the neighbourhood is completely exhausted, taking significantly less time. Recall from Section 2.3.1 that $l$ denotes the maximum amount of neighbours investigated in the VND subroutine in order to find a single schedule improvement. To find a good value for $l$, we run a single VND instance where at most 50

up to 1000 neighbourhood moves are allowed to find an improvement. Finally, we compare the performance of the solutions found after attempting at most these amount of moves to find a single improvement. Three different problem instances are used and 30 runs are performed for each problem instance. In order to perform 50 to 1000 moves without restarting runs, let $N_1 \dots N_{k_{max}}$ denote the neighbourhoods used by the VND algorithm. Now, the used VND algorithm is run as follows:

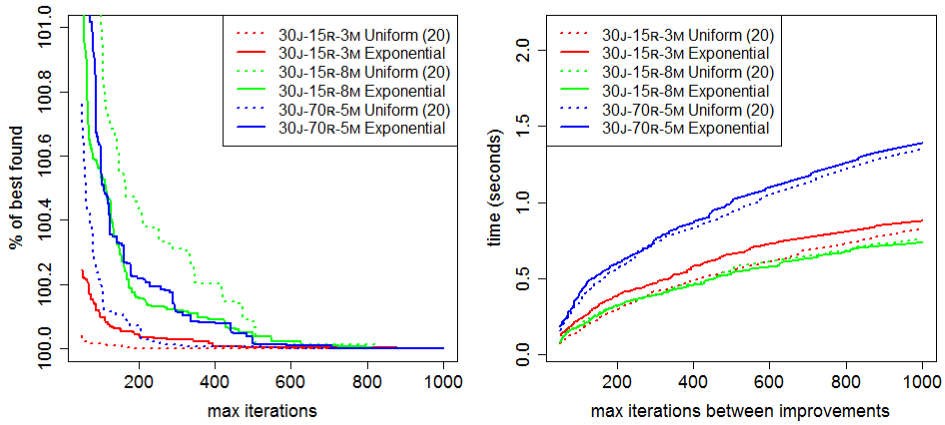
- Generate an initial solution $S$ and let $k = 1$ and $l = 50$
- Repeat the following steps until $l = 1000$:
  - Run a single VND step by investigating at most $l$ neighbours per neighbourhood for $N_1 \dots N_{k_{max}}$, returning a potentially better solution $S'$. Note that the search is stopped for a neighbourhood before $l$ neighbours have been found when that neighbourhood is exhausted.
  - If $S'$ is better than $S$, continue with $S = S'$ and $k = 1$.
  - Otherwise, repetitively increment $l$ and attempt a single neighbourhood move for $N_1 \dots N_{k_{max}}$ until a better solution is found or $l = 1000$. When a better solution $S'$ is found, output the tuple of $l$ and $E(C_{max})$ of $S'$ for the current value of $l$ and continue the process with $k = 1$ and $S = S'$.

Figure 5.3a and Figure 5.3c show the average performance relative to the solution found after an iteration limit 1000 of the *RS 300* and *DM* algorithm respectively after iteration limit 50 to 1000, using problem instances 30J-15R-3M, 30J-15R-8M and 30J-70R-5M. Figure 5.3b and Figure 5.3d show the times needed to find these solutions. We expect that a best reinsertion step will have a significantly greater chance of finding a better solution than a random reinsertion step. Therefore, fewer neighbourhood moves should be necessary for result sampling algorithms than for objective functions to find a good solution. From Figure 5.3a, it can be seen that most improvements have been found after 300 neighbourhood move attempts have already been performed without success, where the $E(C_{max})$ value is only 0.1% off the $E(C_{max})$ value of the local optimum on average. However, from Figure 5.3c, it can be seen that it may often take more than 300 neighbourhood moves to find improvements, though most improvements have been found after 600 improvements, where the $E(C_{max})$ value is less than 0.1% off the $E(C_{max})$ value of the local optimum on average. Therefore, we choose an iteration limit of 300 for result sampling and 600 for objective functions such as *DM* for problem instances with $n = 30$.




(a) Performance of *RS 300*

(b) Required time for *RS 300*



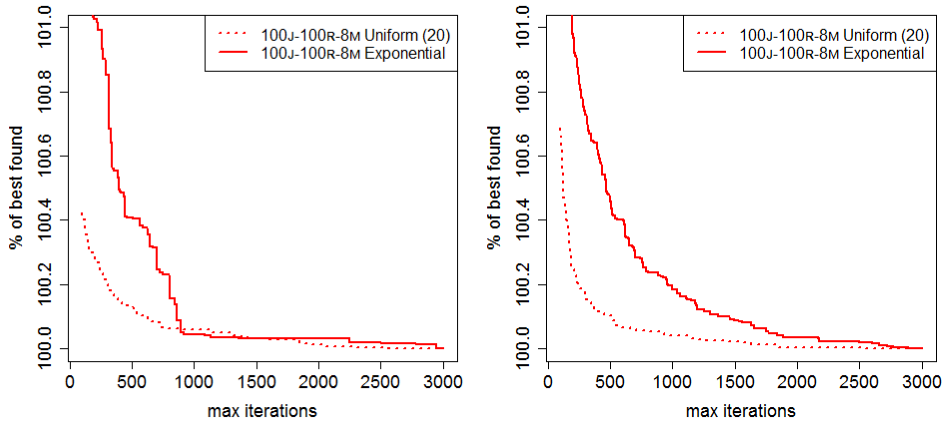

(c) Performance of DM

(d) Required time for DM

Figure 5.3: Experimental results for the maximal amount of neighbourhood moves (iterations) without finding an improvement before stopping the VND subroutine. Schedules are compared with the best found schedule in terms of percentage of objective, e.g. a schedule is ranked 102% when its measured expected makespan is 51 and the expected makespan of the best found schedule for that run is 50.

Figure 5.4 shows the relative performance of *RS 300* (Figure 5.4a) and *DM* (Figure 5.4b) for the 100J-100R-8M problem instance for an iteration limit of 150 up to 3000. From Figure 5.4, it can be seen that the amount of iterations required to find a relatively good solution roughly scales up linearly with the value of $n$. Therefore, we choose an iteration limit of 1000

for result sampling and 2000 for objective functions for problem instances with $n = 100$.




(a) Required time for *RS 300*

(b) Performance of *DM*

Figure 5.4: Experimental results for the maximal amount of neighbourhood moves (iterations) without finding an improvement before stopping the VND subroutine for problem instances with 100 jobs.

### 5.2.3 Perturbation step types

Recall that we introduced two perturbation step types: the *k-swap* operator in Section 2.3.2 and sample replacement in Section 3.2. As explained in Section 2.3.2, the perturbation step size should be chosen such that there is a balance between intensification and diversification.

We have conducted experiments for the *k-swap* operator with $2 \leq k \leq 6$ and perturbations where 10% to 100% of the samples are replaced. To make a fair comparison between the experiments, we first run a single VND subroutine and use the same resulting solution for all experiments. Then, for each experiment, a perturbation is performed and the VND subroutine is run again. Finally, the improvement between the first and the second VND subroutine is measured, which is 0 if the second solution is not better. In total, 30 runs have been performed for each problem instance and result sampling with 300 samples is used. The average improvement after running VND again after the perturbation step is shown in Figure 5.5a and Figure 5.5b for the *k-swap* and the sample replacement perturbation operator respectively. Figure 5.5c and Figure 5.5d show the average run time of the second VND subroutine for the *k-swap* and the sample replacement operator respectively. Even though 30 runs have been performed, it can be seen that

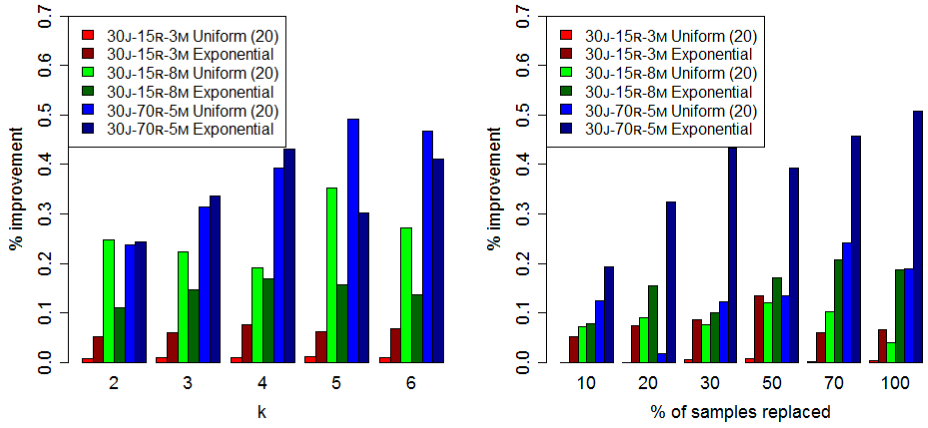
the results have a strong variance and are not very reliable as a consequence. This is caused by the fact that many runs are wasted since no improvement is found.

When comparing Figure 5.5a and Figure 5.5b, it becomes evident that replacing samples is relatively less effective when $D = Uniform(20)$ compared to *k-swap*. For $D = Exponential$, it is not clear which perturbation step type is the best. Sample replacement seems slightly more effective, but this conclusion is unreliable because of the large variance. Furthermore, the larger perturbation steps of both *k-swap* and sample replacement seem most effective. The VND subroutine of sample replacement finishes significantly faster for problems with $D = Uniform(20)$. Since the resulting performance is also worse, it is likely that the perturbation steps of sample replacement are too small, even when all samples are replaced. This holds to a lesser degree when $D = Exponential$, since exponentially distributed samples generally differ greatly. Since the *k-swap* perturbation operator is generally more effective than sample replacement and *k-swap* is most effective for larger values of $k$, we choose the *5-swap* operator for perturbation steps.

### 5.2.4 Amount of perturbations

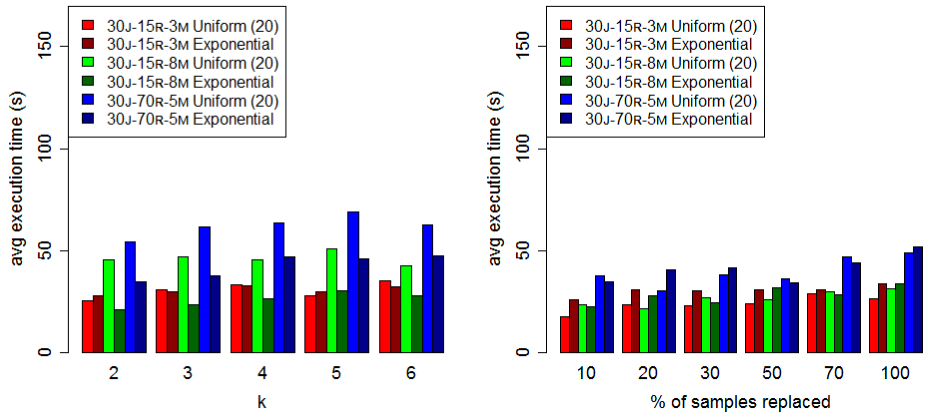
Similar to the stop criterion, we need to make a trade-off between schedule quality and run time in order to choose the right amount of perturbations. Performing many perturbations will cause the algorithm to run for a very long time without much chance of improvement, whereas significant improvement could be made in relatively little time when too few or no perturbations are performed. To find a balance between schedule quality and run time, we perform 10 runs for various problem instances where at most 20 perturbations are performed, using *RS 300*. Each schedule found after a VND round is compared to the best schedule after 20 perturbation and the schedule quality offset is measured.

Figure 5.6 shows the average performance after 0 to 20 perturbations relative to the solution found after 20 perturbations (Figure 5.6a) and the time required to find these solutions (Figure 5.6b). From Figure 5.6a, it can be seen that most improvement is made after the first few VND rounds. Figure 5.6b shows that the required run time increases in a linear fashion. For the final experiments, we present results after both 0 and 8 perturbations for faster algorithms, since the schedules improves significantly less after more than 8 perturbations. For the slower algorithms, *RS 300* and *RS 1000*, we use at most 2 and 0 perturbations respectively in order to save run time.




(a) Performance of k-swap

(b) Performance of replacing samples



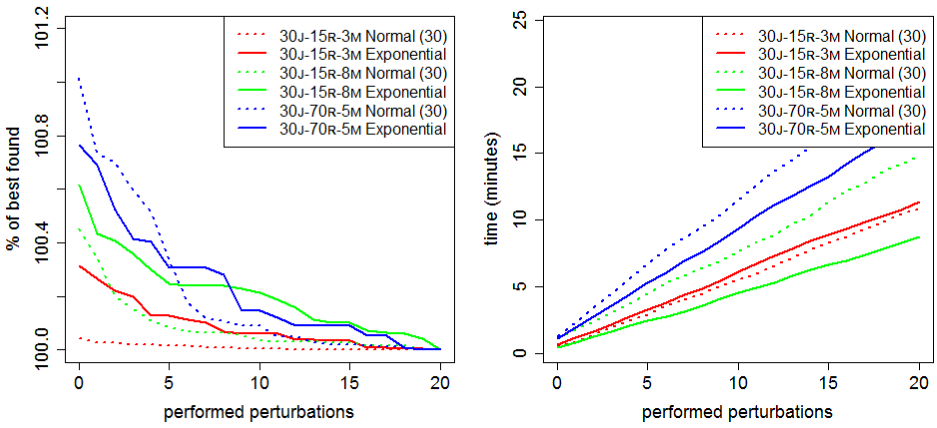

(c) Time required for VND subroutine after k-swap

(d) Time required for VND subroutine after replacing samples

Figure 5.5: Experimental results for various *k-swap* and sample replacement perturbation steps. The best schedule found after a single perturbation step is compared with the schedule found by the first VND subroutine and the average relative improvement and run time of the second VND subroutine is shown.

### 5.2.5 Experiments for makespan without precedence relations

In order to decide which objective function is used for approximating the expected makespan without precedence relations for the final experiments, we compare the *GM* (Section 4.2.2) function with various parameter settings




(a) (b)

Figure 5.6: Experimental results for the perturbation amount parameter. The results are compared with the best found result after 20 perturbations.

with the *AML* (Section 4.2.1) and *IGM* (Section 4.2.3) functions. In order to find an appropriate set of values $x_1 \dots x_s$ for the *GM* function, we introduce two new parameters $d$ and $b$. Here, $d$ denotes the difference between $x_k$ and $x_{k+1}$ relative to $c_{max}$, i.e. $x_{k+1} - x_k = c_{max} \cdot d$. Furthermore, $b$ serves as a lower bound for $x_1$ and an upper bound for $x_s$, i.e. $x_1 \dots x_s$ are chosen such that $Pr(C_{max} \leq x_1) \geq b$ and $Pr(C_{max} \leq x_s) \geq 1 - b$ for maximal $s$. Let $x_c = c_{max}$. Now, all $x_k$ where $1 \leq k < c$ are computed by subtracting $c_{max} \cdot d$ from $x_{k+1}$ until $Pr(C_{max} \leq x_{k-1}) < b$ and all $x_k$ where $c < k \leq s$ are computed by adding $c_{max} \cdot d$ to $x_{k-1}$ until $Pr(C_{max} \leq x_{k+1}) > 1 - b$.

For each experiment, 30 runs of a single VND round are performed. We only test problem instances with no precedence relations involved, since the *AML*, *IGM* and *GM* functions ignore the presence of precedence relations and consequently, adding precedence relations to the problem instances may cause extra unnecessary result variance. Furthermore, we have used $D = Normal(30)$ instead of $D = Uniform(20)$ in order to investigate whether the *IGM* and *GM* functions perform relatively better on normal distributions compared to the *AML* function, since then, no evaluation accuracy is lost by assuming that $P_j$ is normally distributed, contrary to the case when $P_j$ is exponentially distributed. Table 5.1 shows the average performance and required run time for each of the objective functions for the 30J-0R-3M and 30J-0R-8M problem instances. From Table 5.1, it becomes apparent the *GM* function performs slightly better for small $d$ and $b$ at the cost of a significantly greater run time. Furthermore, it becomes apparent

that the run time depends on $m$ for all objective functions except the *AML* function. Even though 30 runs have been performed, the performance difference between the objective functions is unclear because of the relatively large result variance. It can be seen that the *IGM* function is approximately as effective as the *GM 2/2* function, but significantly faster. Furthermore, the *IGM* function seems slightly more effective than the *AML* function, though the *IGM* function also seems slightly slower for large $m$. The expected decrease of accuracy of the *IGM* function for larger $m$ cannot be measured from the results, since the result variance is too large. Finally, no significant improvement of accuracy can be measured for the *IGM* and *GM* functions compared to the *AML* functions when $D = Normal(30)$. Since the *IGM* function has a good balance between effectiveness and required run time, we choose to pair the *IGM* function with the *AD* function in the final experiments.

Table 5.2 show the required run time for the *AML*, *IGM* and *GM 2/2* functions when precedence relations are involved and the AD function is used to approximate the expected delay caused by these precedence relations. From Table 5.2, it can be seen that when precedence relations are involved, the *GM* function is still significantly slower than the *AML* and *IGM* functions. Consequently, we can conclude that approximating $E(C_{max})$ without precedence relations using *GM* requires much more run time than than approximating the slack caused by precedence relations using the *AD* function.

### 5.2.6 Choosing $\alpha$ and $\beta$ for the arctan delay function

In order to choose the best $\alpha$ and $\beta$ values for the *AD* function, we have run the *AD* function paired with the *IGM* function using four different values for $\alpha$ and $\beta$. For each experiments, 30 runs of a single VND round are performed. The results are presented in Table 5.3. Except for the results for $\alpha = 0.1$, it can be seen that the variance of the results is larger than the mutual result differences and the best parameter values cannot unanimously be chosen. Therefore, we choose the middle ground, $\alpha = 0.5$ and $\beta = 0.3$. Note that there is no significant difference in run time, since the run time of an *AD* function evaluation does not depend on $\alpha$ or $\beta$.

## 5.3 Final experiments

In this section, we describe the experiments for comparing the effectiveness of result sampling (*RS*), the combination of the iterative gaussian makespan and arctan delay (*IGM/AD*) function and the dynamic makespan (*DM*) function. We first explain the experimental set-up and then we interpret

| Method | Normal (30) | | | | Exponential | | | |
|---|---|---|---|---|---|---|---|---|
| | 30J-0R-3M | | 30J-0R-8M | | 30J-0R-3M | | 30J-0R-8M | |
| | result | time | result | time | result | time | result | time |
| AML | 0.05 | 0.16 | 0.264 | 0.35 | 0.258 | 0.16 | 0.503 | 0.35 |
| IGM | 0.05 | 0.15 | 0.206 | 0.5 | 0.261 | 0.15 | 0.421 | 0.45 |
| GM 10/5 | 0.261 | 0.39 | 0.363 | 1.92 | 0.327 | 0.47 | 0.453 | 3.16 |
| GM 10/2 | 0.049 | 0.33 | 0.263 | 1.99 | 0.627 | 0.71 | 0.452 | 3.6 |
| GM 10/1 | 0.172 | 0.44 | 0.18 | 2.18 | 0.293 | 0.75 | 0.435 | 4.32 |
| GM 5/5 | 0.125 | 0.38 | 0.177 | 2.65 | 0.446 | 0.8 | 0.416 | 5.49 |
| GM 5/2 | 0.04 | 0.39 | 0.209 | 3.14 | 0.626 | 1.38 | 0.402 | 6.28 |
| GM 5/1 | 0.152 | 0.51 | 0.144 | 3.42 | 0.507 | 1.3 | 0.458 | 7.62 |
| GM 2/5 | 0.048 | 0.62 | 0.205 | 5.46 | 0.268 | 1.28 | 0.387 | 12.7 |
| GM 2/2 | 0.122 | 0.7 | 0.206 | 6.8 | 0.557 | 2.37 | 0.318 | 14.71 |
| GM 2/1 | 0.063 | 0.72 | 0.164 | 7.28 | 0.638 | 3.51 | 0.395 | 16.8 |
| GM 1/5 | 0.042 | 1.06 | 0.181 | 9.2 | 0.531 | 3.34 | 0.398 | 24.54 |
| GM 1/2 | 0.053 | 1.22 | 0.169 | 12.17 | 0.353 | 3.64 | 0.36 | 27.31 |
| GM 1/1 | 0.044 | 1.29 | 0.163 | 12.99 | 0.542 | 6.53 | 0.397 | 30.8 |

Table 5.1: Experimental results for various $E(C_{max})$ measures without precedence relations. Results show the average absolute offset from the global best found solution for each problem instance. Time is in seconds. GM $(X/Y)$ denotes a GM run for $d = X/100$ and $b = Y/100$.

| Measure | time | $\sigma$ |
|---|---|---|
| AML | 0.51 | 0.14 |
| IGM | 0.63 | 0.17 |
| GM 2/2 | 9.03 | 3 |

Table 5.2: Averaged runtime in seconds for three $E(C_{max})$ measures, including run time deviation. The 30-30-5 problem instance and $D = Exponential$ is used to illustrate runtimes when precedence relations are involved.

the results.

### 5.3.1 Experimental set-up

The goal of these experiments is to find out which algorithms have the best performance in various different circumstances. Theoretically, the optimal or near-optimal solution can be found by using an extremely large amount of samples (e.g. one million) and a large amount of large perturbation steps or local search restarts. In practice however, this will take too much time. Therefore, for a given problem instance of some kind, we want to find

| | | 30J-20R-5M | | | | 30J-70R-5M | | | |
|---|---|---|---|---|---|---|---|---|---|
| $D$ | $\alpha\backslash\beta$ | 0.1 | 0.3 | 0.5 | 1 | 0.1 | 0.3 | 0.5 | 1 |
| Uni | 0.1 | 0.147 | 0.16 | 0.16 | 0.158 | 2.467 | 2.849 | 2.729 | 2.875 |
| | 0.3 | 0.166 | 0.142 | 0.149 | 0.155 | 2.689 | 2.921 | 2.712 | 2.742 |
| | 0.5 | 0.171 | 0.121 | 0.117 | 0.117 | 2.956 | 2.467 | 2.676 | 3.013 |
| | 1 | 0.074 | 0.101 | 0.105 | 0.087 | 2.199 | 2.548 | 2.409 | 2.698 |
| Exp | 0.1 | 1.101 | 1.592 | 1.771 | 1.812 | 7.723 | 7.724 | 7.269 | 6.23 |
| | 0.3 | 0.918 | 1 | 1.175 | 1.597 | 5.048 | 3.658 | 4.992 | 5.162 |
| | 0.5 | 1.053 | 0.999 | 1.186 | 1.336 | 3.694 | 3.264 | 3.677 | 4.908 |
| | 1 | 1.071 | 0.936 | 1.026 | 1.124 | 3.682 | 3.336 | 4.112 | 4.81 |

Table 5.3: Experimental results for $E(C_{max})$ measures with precedence relations where $D = Uniform(20)$ (upper rows) and $D = Exponential$ (lower rows). Results show the average absolute offset from the global best found solution for each problem instance. Rows denote tested alpha values and probability distributions and columns denote tested problem instances and beta values for the AD function.

out which algorithms have the best performance when run time is limited, i.e. which algorithms represent the Pareto frontier of average run time and average makespan.

The Pareto frontier may be different for different problem instances or different processing time probability distributions. Therefore, we vary the amount of relations and machines for a fixed amount of jobs in the experiments and we use four different probability density functions for $D$. Larger problem instances with 100 jobs are included as well to investigate the scalability of the approaches. The experiments are repeated five times and the average of the results of these runs is presented in the result tables.

In Appendix A, the average makespans and run times for all attempted problem instances are presented. In order to identify the Pareto frontier, we present the best performing algorithms and their corresponding average makespan with a given time constraint (e.g. less than 10 seconds) in Appendix B. The deviations in terms of percentage for makespans for problem instances are presented in Appendix C. The algorithms *IGM/AD*, *DM* and *RS 100* are compared with the optimal solution found by Van Roermund (2013) in Appendix D. For the experiments in Appendix D, 8 ILS perturbations are performed and 10 runs instead of 5 are performed for each experiment. In order to test the performance for larger $q_{ij}$, we have run experiments with problem instances where each $q_{ij}$ is exponentially distributed with an average of $p_i$, contrary to the other problem instances used throughout this thesis where $q_{ij}$ is uniformly distributed between 0 and $p_i$. In Appendix E, the results and best performing algorithms within a time

constraint are shown for these problem instances. For these experiments, 12 ILS perturbations and 10 runs per experiment are performed, where only problem instances with 30 jobs and 5 machines are used.

In addition to local search experiments, we compare the evaluations of the objective functions presented in this thesis with an $E(C_{max})$ measure using 10000 samples on a large set of schedules $\mathcal{S}$. We refer to the set of objective functions presented in this thesis as $F$ and the $E(C_{max})$ approximation using 10000 samples as $f^*$. In order to compare $f \in F$ and $f^*$, we measure the correlation of function evaluations on the schedules and a measure which we refer to as the position offset. For the position offset, we first order the schedules by increasing $E(C_{max})$ using $f$ and $f^*$. The most accurate objective functions will order the schedules most similar to the ordering by $f^*$. For a schedule $S \in \mathcal{S}$, the difference between the positional index of $S$ in the orderings by $f$ and $f^*$ measures the error $E_S$ of $f$. Now, the position offset is defined as follows:

$$\frac{100}{|\mathcal{S}|^2} \sum_{S \in \mathcal{S}} E_S$$

The factor $\frac{100}{|\mathcal{S}|^2}$ scales the total error so that the position offset represents the average error in terms of percentage, relative to the amount of samples. For instance, a position offset of 10 denotes that the average $E_S$ value is 10% of the size of the ordering. The schedules are obtained separately for each problem instance and $D$ by performing a VND algorithm 30 times in a row and gathering the schedules found after each improvement. Each VND run is started from a random schedule in order to enhance diversity. Finally, we dispose a quarter of the schedules with the worst $f^*$ value so that we keep various schedules with nearly the same $E(C_{max})$ value. The results of these experiments are presented in Appendix F.

### 5.3.2 Comparison of the approaches

From the tables in Appendix A and Appendix B, it can be observed that *DM* is much more effective than *IGM/AD* for nearly all problem instances, though it only requires slightly more run time. The effectiveness of *IGM/AD* declines for large $r$ compared to result sampling, whereas *DM* still remains roughly as effective as *RS 100* though it is approximately 5 times as fast. Furthermore, when the variance of $D$ is small, e.g. when $D = Uniform(20)$ or $D = Normal(30)$, we observe that *RS 100 (8)* often outperforms *RS 300 (2)* and *RS 1000 (0)*, while using less run time. In other words, using relatively few samples and many ILS perturbations is most effective when the variance of $D$ is small, wheras result sampling with many samples and

few perturbations is most effective when the variance of $D$ is large.

From the table in Appendix D, it becomes apparent that the performance of the fixation maximization algorithm of Van Roermund (2013) is roughly comparable to the *IGM/AD* approach. Fixation maximization has the best performance for small $\sigma^2(D)$ and small $r$, since Van Roermund (2013) ensures that $c_{max}$ is minimized and this becomes relatively important when the total expected delay caused by precedence relations is small. The local search algorithm used in this research does not guarantee minimization of $c_{max}$, although the approximation objective functions and result sampling algorithms perform much better than fixation minization when expected delay caused by precedence relations is large, i.e. for large $r$ and large $\sigma^2(D)$. This may be caused by the fact that the amount of fixated precedence relations is a significantly less accurate measure for expected delay caused by precedence relations compared to the objective functions used in this research.

From the tables in Appendix E, it can be seen that the effectiveness of *IGM/AD* declines even more strongly for large $r$ compared to result sampling for problem instances where $q_{ij}$ is exponentially distributed with average $p_i$. However, *DM* still competes with *RS 100* for problem instances with large $r$ in terms of accuracy.

### 5.3.3 Run time

From the table in Appendix A, it becomes apparent that in terms of run time, *IGM/AD* and *DM* are comparable to *RS 30*. The run time depends roughly linearly on the amount of samples used. Furthermore, for problem instances with large $r$, the run times of all objective functions become slightly slower (up to a factor 2).

When comparing run times of problem instances with 100 jobs with run times of problem instances with 30 jobs, we find that the run time of *IGM/AD* and *DM* increases approximately 10-15 times, whereas the run time of result sampling increases approximately 20-30 times. This may be caused by the fact that not only the evaluation time of neighbourhood candidates increases, but also the neighbourhood size and the amount of improvements required to find a local optimum. Therefore, we expect that all algorithms presented in this research scale with a factor of approximately $O(n^3)$ in order to keep their performance. The smaller run time increase of the *IGM/AD* and *DM* functions can be explained by the fact that approximation time of $E(C_{max})$ when all $C_{max}(m_k)$ are known depends on $m$ rather than $n$. Only the remainder of computations depend on $r$ and $n$.

### 5.3.4 Result deviations

From the tables in Appendix C, it becomes apparent that the result variance is inversely correlated with algorithm performance. In other words, the less accurate algorithms generally have a greater result variance. It can also be seen that result variance increases with $r : n$ and $m : n$ ratio. This may be caused by the fact that the error accumulation increases when the $r : n$ and $m : n$ ratio is large. Finally, we observe that result variance is significantly greater when $D$ has greater variance. This observation is mostly notable for problem instances and algorithms where makespan variance is low.

### 5.3.5 Comparing objective function evaluations

From the tables in Appendix F, we can observe some performance difference between result sampling and objective functions without simulation. Result sampling is relatively most effective on problem instances with many precedence relations, many machines and small $\sigma(D)$, whereas objective functions without simulation perform well on problem instances with few relations and machines where $D = Exponential$. For the former problem instances, it can be seen that *DM* performs similar to *RS 100*, *IGM/AD* performs similar to *RS 30* and $c_{max}$ performs significantly worse than *RS 30*. For the latter problem instances, it can be seen that *DM* performs similar to *RS 1000*, *IGM/AD* performs similar to *RS 300* and $c_{max}$ performs similar to *RS 100*. We also observe that the correlation and position offset are inversely correlated for all problem instances and distributions. Furthermore, the *IGM/AD* function approaches the performance of $c_{max}$ for large $r$ and it approaches the performance of *DM* for small $r$, indicating that the *AD* function does not approximate delay caused by precedence relations accurately when delay propagation is frequent. For the evaluation times, we find that the *DM* and *IGM/AD* functions are less than twice as slow as the $c_{max}$ computation and 10-20 times faster than *RS 30*.

$\sigma(D)$ strongly reduces the effectiveness for result sampling, since significantly more samples are required to represent the various outcomes. For small $r$ and $D = Exponential$, the single sample where $P_j = p_j$ may be more representative for approximating $E(C_{max})$ than 30 randomly generated samples. Result sampling is relatively most effective for large $r$, since the various outcomes of delay propagation can be found effectively, especially when many samples are used. The assumptions made for the objective functions without simulation introduce a small error for each of the precedence relations. The effect of error accumulation is significantly more present when many precedence relations are involved, since errors propagate when delays propagate. Nevertheless, result sampling is only more effective than *DM*

when allowing at least 30 times more run time in all tested cases, making the $DM$ function significantly more practical for approximation purposes in general.

# Chapter 6

# Conclusion and further research

In this chapter, we summarize the approaches and draw the most important conclusions from the experiments. Finally, we present pointers for future research in the context of minimizing $E(C_{max})$ and maximizing robustness in general.

## 6.1 Summary

In this thesis, we have studied the problem of maximizing the robustness in machine scheduling. We have used the expected makespan of the Robust Parallel Machine Scheduling (RPMS) problem as the performance measure of robustness, where processing times are drawn from a probability distribution. The expected makespan is optimized by using objective functions for approximating $E(C_{max})$ within a local search framework.

For the local search framework, we have used iterative local search (ILS) with a variable neighbourhood descent (VND) subroutine. For VND, we have consecutively used three neighbourhoods: the *1-move*, *2-swap* and *2-move* neighbourhoods. For the ILS perturbation step, we have used the *5-swap* neighbourhood. Finally, we have examined valid neighbourhood operations.

Based on the graph representation presented in Chapter 1, we have implemented and compared two different types of objective functions. The first type of approximation of expected makespan is called Result Sampling ($RS$), which uses the average makespan of a set of stochastic samples. We have found that replacement of samples during the local search is not effective for RPMS, since the local search does not converge and the power of the different neighbourhoods is not combined. We have found that re-

placement of samples as a perturbation step is less effective than the *k-swap* operator, since the operator is not disruptive enough in most cases. Furthermore, we have prioritized job reinsertion by the amount of samples for which the job is located on the critical path. Finally, we have applied an $|s| \cdot O(m+n+r)$ algorithm for finding the best reinsertion for a single job for the *1-move* operator and applied this algorithm for effectively finding good *2-move* operations.

The second type of approximation of expected makespan uses a single, deterministic schedule where properties of stochastic schedules are captured by applying probability theory. First, we have presented an approach which combines the approximation of expected makespan without regarding precedence relations, called the iterative Gaussian makespan ($IGM$) function, with approximation of the slack time caused by precedence relations, called the arctan delay ($AD$) function. Then, we have presented the dynamic makespan ($DM$) function, which approximates the PDF of the starting and completion time for each job using the PDFs of the starting and completion times of its machine predecessors and precedence predecessors, using the expected delay caused by precedence relations.

## 6.2 Conclusion

In the context of minimizing the expected makespan of the RPMS problem, we have compared the result sampling approach based on the work of Van den Akker et al. (2013) with our presented approaches capturing properties of stochastic schedules in a single deterministic schedule. From the local search experiments, we have found that the dynamic makespan ($DM$) function finds better solutions than result sampling for up to 100 samples for all problem instances while requiring significantly less run time. Furthermore, the $DM$ function performs significantly better than the paired iterative Gaussian makespan and arctan delay $IGM/AD$ function, though it only requires slightly more run time, rendering the $IGM/AD$ function useless in most cases. The $DM$ function is more effective than the approach of Van Roermund (2013) except for problem instances with few jobs and precedence relations where the underlying probability distribution of processing times ($D$) has small variance. The amount of performed perturbations is relatively more effective when $D$ has small variance, whereas the amount of samples used by result sampling is more effective when $D$ has large variance. In terms of function evaluation accuracy, the $DM$ function is most accurate when $D$ has large variance and $r$ is small and result sampling is most accurate when $D$ has small variance and $r$ is large. We have also found that the best performing algorithms generally have the smallest performance

variance when multiple runs are performed, especially when using many ILS perturbations.

## 6.3 Further research

Within the context of maximizing robustness, many enhancements can be made to improve the performance of the algorithms presented in this thesis, to find new algorithms building upon these algorithms and to apply these algorithms within a wider context than RPMS with $E(C_{max})$ maximization.

### 6.3.1 Improving efficiency

Recall from Section 3.3 that job selection in local search for result sampling is prioritized by the amount of critical paths it occurs on. We have not presented a way to prioritize job selection for the objective functions presented in Chapter 4. Selection of neighbourhood operations could be prioritized using a heuristic. A simple prioritization could prefer to select jobs on the critical path. The amount of fixations gained (or lost) by moving the job to a particular machine could be used for prioritization, since the amount of fixations is often correlated with the value of the used objective function. Alternatively, we note that jobs with many descendants should be scheduled as early as possible, whereas jobs with many ancestors should be scheduled as late as possible. This notion can also be used in order to find a prioritization of jobs. A more sophisticated heuristic would try to prune neighbourhood operations by computing a lower bound for the objective function after moving a job to a certain position. The computation of this lower bound should be significantly faster than the computation of the entire objective function. If this lower bound exceeds the value of the objective function for the current schedule, the operation could effectively be pruned from the neighbourhood.

In addition to picking more effective operations first, computation time can also be saved by means of caching. Random insertions can be sped up slightly by caching the valid machine predecessors for each job, so they do not have to be recomputed before each random insertion. Additionally, the head and tail times of $G^-$, presented in Section 3.4.1 can sometimes be partially reused when new jobs $j$ are attempted for best-reinsertions.

Unfortunately, the best reinsertion algorithm for result sampling is not directly applicable for the objective functions without result sampling, since their evaluation depends on more than head and tail times only. Nevertheless, partial computations may still be reusable. For the *IGM* function, computation of the head and tail times may help in order to quickly com-

pute all $C_{max}(m_k)$ values for the best reinsertion algorithm for local search. Furthermore, $E(C_{max})$ only needs partial recalculation for each attempted reinsertion on the same machine. For the $DM$ and $AD$ function, the PDFs of starting and completion times or expected delay values for jobs $i$ can be reused after reinsertion of a job $j$ when there exists no path from $j$ to $i$ in $G$ both before removing $j$ from $G$ and after reinserting $j$ into $G$.

### 6.3.2 Algorithm improvement

The quality of the algorithms presented in this thesis may be improved by various means. For result sampling, we have used a set of random samples. Especially when few samples are used, these samples may be biased so that minimizing the average makespan of these samples may result in a much more optimistic average makespan than the actual value of $E(C_{max})$. Optimization bias can be minimized by selecting samples such that they satisfy a set of properties in order to enhance diversity and balance. An example of reduction of optimization bias is cutoff sampling, presented in Section 3.1.

The $DM$ function presented in Section 4.4 can be improved at the expense of run time. One of the most troublesome assumptions is that given a job $j$ and $i \in pp(j)$, we assume that $S_i$ and $S_j^{k-1}$ are independent. Dependency of jobs causes overestimation of $\sigma^2(\delta_{ij})$, since dependencies eliminate a part of the variance of the difference between the PDF of $S_i$ and $S_j$. The degree of dependency between $i$ and $j$ could be approximated for a more accurate approximation of $\sigma^2(\delta_{ij})$. Furthermore, instead of assuming that all $S_j$ are normally distributed, one could approximate the convolution of the PDFs of predecessors using the work of Schaller and Temnov (2008) in the general case and Bibinger (2013) for exponential distributions specifically. This is especially important when $D = Exponential$, since the exponential distribution and normal distribution differ strongly.

### 6.3.3 Other machine scheduling problems and objectives

In this thesis, we used the context of the robust parallel machine scheduling problem with precedence relations between starting times and expected makespan minimization as a measure for robustness. Although this is a specific context, the RPMS problem is quite generic and therefore, the algorithms presented in this thesis may also be effective on other parallel machine scheduling problems such as job shop scheduling. The algorithms should also still be effective when only a part of the jobs have uncertain processing times or when processing times follow varying probability distributions. Although the best-insertion algorithm used for result sampling may not be applicable for other objective functions than $E(C_{max})$, many

algorithms presented in this thesis could be completely or partially reused in order to optimize other objective functions.

### 6.3.4 Other notions of robustness

Another application of robustness maximization is the minimization of non-punctuality, which is defined as follows. Within a problem instance, we want to find a schedule $S$ where $C_{max}$ is bounded by a maximum and as few jobs as possible finish more than $\Delta$ seconds later than planned in a stochastic realisation of $S$. This objective function can be measured by using a large amount of samples and the starting times of jobs within a schedule $S$ can be optimized by running an ILP. This notion of robustness can directly be applied in the context of minimizing delay in public transport schedules.

# Appendices

# Appendix A

# Final experiment results and run times

The following tables show the absolute results (average makespan) and run times in seconds for the final experiments described in Section 5.3. The deviations of the results and run times are shown in Appendix C.

| | | Method | IGM/AD | | DM | | RS 30 | | RS 100 | | RS 300 | | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Perturbations | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 2 | 0 |
| 30J-15R-4M | Uniform (20) | result | 75.9 | 75.78 | 75.78 | 75.78 | 76.02 | 75.79 | 75.75 | 75.72 | 75.72 | 75.72 | 75.78 |
| | | time | 3.78 | 8.9 | 3.42 | 8.22 | 4.71 | 21 | 13.7 | 68.8 | 56.8 | 123 | 144 |
| | Normal (30) | result | 81.67 | 81.67 | 81.8 | 81.64 | 81.98 | 81.83 | 81.82 | 81.78 | 81.7 | 81.68 | 81.74 |
| | | time | 3.74 | 8.7 | 3.58 | 9.03 | 4.91 | 22.3 | 14 | 73.5 | 46.2 | 126 | 147 |
| | Erlang (4) | result | 88.88 | 88.83 | 88.93 | 88.79 | 89.76 | 89.19 | 89.33 | 88.95 | 88.89 | 88.87 | 88.85 |
| | | time | 6.47 | 11.5 | 6.13 | 12 | 6.34 | 22.6 | 15.2 | 78.4 | 41.6 | 121 | 178 |
| | Exponential | result | 107.21 | 107.05 | 107.77 | 107.01 | 108.7 | 107.88 | 108.39 | 107.63 | 107.57 | 107.22 | 106.93 |
| | | time | 3.68 | 8.55 | 3.54 | 8.51 | 3.87 | 16.4 | 12.6 | 56.3 | 37.7 | 92.5 | 171 |
| 30J-30R-4M | Uniform (20) | result | 77.44 | 77.37 | 77.39 | 77.37 | 77.47 | 77.37 | 77.43 | 77.37 | 77.38 | 77.37 | 77.38 |
| | | time | 3.71 | 9.9 | 3.83 | 10.4 | 5.57 | 27.9 | 18.2 | 86.9 | 49.4 | 138 | 233 |
| | Normal (30) | result | 83.26 | 83.12 | 83.09 | 83.06 | 83.27 | 83.19 | 83.12 | 83.1 | 83.1 | 83.09 | 83.05 |
| | | time | 3.83 | 9.89 | 3.87 | 10.9 | 5.79 | 27.1 | 15.8 | 83.5 | 62.8 | 145 | 250 |
| | Erlang (4) | result | 90.53 | 90.14 | 89.97 | 89.89 | 90.7 | 90.31 | 90.1 | 89.97 | 89.98 | 89.95 | 89.97 |
| | | time | 6.61 | 11.9 | 6.31 | 12.8 | 5.49 | 26.1 | 15.9 | 79.2 | 45.8 | 134 | 201 |
| | Exponential | result | 108.91 | 108.11 | 108.11 | 107.79 | 109.29 | 108.87 | 108.82 | 107.82 | 107.93 | 107.8 | 107.88 |
| | | time | 3.68 | 9.35 | 3.75 | 10.1 | 4.59 | 22.4 | 11.4 | 64.3 | 40.2 | 118 | 186 |
| 30J-75R-4M | Uniform (20) | result | 78.74 | 76.99 | 77.51 | 77.04 | 78.01 | 77.1 | 77.86 | 76.67 | 77.72 | 77.01 | 77.5 |
| | | time | 3.85 | 11.4 | 4.46 | 13.6 | 5.94 | 34.7 | 17.1 | 98.4 | 53 | 160 | 290 |
| | Normal (30) | result | 85.5 | 84.79 | 84.35 | 83.76 | 85.42 | 83.76 | 84.24 | 83.73 | 84.4 | 84.16 | 84.27 |
| | | time | 3.71 | 11 | 4.83 | 15.2 | 7.29 | 35.9 | 27.3 | 122 | 46.3 | 158 | 235 |
| | Erlang (4) | result | 94.71 | 94 | 94.59 | 93.23 | 94.55 | 93.25 | 94.16 | 92.64 | 93.17 | 92.93 | 93.12 |
| | | time | 6.55 | 13.9 | 6.93 | 16.3 | 7.62 | 33.1 | 27.8 | 110 | 80.4 | 165 | 296 |
| | Exponential | result | 119.75 | 118.44 | 117.56 | 116.31 | 120.6 | 118.32 | 117.8 | 116.33 | 117.6 | 116.11 | 115.55 |
| | | time | 3.92 | 10.9 | 4.57 | 13.1 | 7.06 | 29.3 | 20.1 | 95.9 | 79.9 | 194 | 276 |

Table A.1: Processed results for the final experiments for 30-*-4 problem instances

| | | Method | IGM/AD | | DM | | RS 30 | | RS 100 | | RS 300 | | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Perturbations | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 2 | 0 |
| 30J-15R-8M | Uniform (20) | result | 41.44 | 41.36 | 41.45 | 41.36 | 41.49 | 41.4 | 41.56 | 41.33 | 41.47 | 41.35 | 41.41 |
| | | time | 3.86 | 9.42 | 3.67 | 8.88 | 4.63 | 19.5 | 14.7 | 78.8 | 42.5 | 118 | 213 |
| | Normal (30) | result | 47.29 | 47.07 | 47.24 | 47.11 | 47.59 | 47.35 | 47.28 | 47.12 | 47.21 | 47.12 | 47.08 |
| | | time | 3.73 | 9.01 | 3.75 | 9.07 | 5.08 | 21.6 | 14.9 | 73 | 32.9 | 106 | 270 |
| | Erlang (4) | result | 55.12 | 55.01 | 55.02 | 54.9 | 55.62 | 55.31 | 55.55 | 55.05 | 55.13 | 55.06 | 55.09 |
| | | time | 6.29 | 11.6 | 6.29 | 11.7 | 4.42 | 21.1 | 12.5 | 63.4 | 50 | 117 | 176 |
| | Exponential | result | 75.21 | 74.81 | 74.78 | 74.74 | 76.65 | 76.13 | 75.55 | 75.23 | 75.21 | 74.96 | 74.73 |
| | | time | 3.76 | 8.99 | 3.57 | 8.3 | 3.91 | 16.9 | 11.5 | 48.6 | 29.6 | 93.7 | 235 |
| 30J-30R-8M | Uniform (20) | result | 29.93 | 29.86 | 29.89 | 29.79 | 29.88 | 29.8 | 29.84 | 29.79 | 29.8 | 29.79 | 29.8 |
| | | time | 3.59 | 9.29 | 3.62 | 9.53 | 3.79 | 17.9 | 10.7 | 70.8 | 37.8 | 109 | 305 |
| | Normal (30) | result | 33.57 | 33.38 | 33.33 | 33.23 | 33.47 | 33.34 | 33.34 | 33.28 | 33.36 | 33.26 | 33.29 |
| | | time | 3.65 | 9.24 | 4.23 | 11.3 | 5.27 | 25 | 15.5 | 69.1 | 71.7 | 170 | 231 |
| | Erlang (4) | result | 38.8 | 38.75 | 38.64 | 38.51 | 38.79 | 38.72 | 38.64 | 38.61 | 38.57 | 38.53 | 38.57 |
| | | time | 6.34 | 11.6 | 6.48 | 13.1 | 6.3 | 21.2 | 13.8 | 69.2 | 48.4 | 144 | 192 |
| | Exponential | result | 52.55 | 52.37 | 52.09 | 51.98 | 53.88 | 52.74 | 52.39 | 52.07 | 51.99 | 51.95 | 51.93 |
| | | time | 3.5 | 8.69 | 3.72 | 9.41 | 3.75 | 19.3 | 11.8 | 63.5 | 37.7 | 94.7 | 175 |
| 30J-75R-8M | Uniform (20) | result | 50.11 | 49.66 | 49.54 | 49.52 | 49.58 | 49.54 | 49.53 | 49.52 | 49.53 | 49.52 | 49.53 |
| | | time | 4.21 | 12.4 | 4.57 | 13 | 2.78 | 11.6 | 6.06 | 27.7 | 15.6 | 40.4 | 87 |
| | Normal (30) | result | 54.53 | 54.42 | 53 | 52.91 | 53.62 | 53.21 | 53.01 | 52.94 | 53.07 | 52.97 | 52.94 |
| | | time | 4.19 | 12.1 | 5.11 | 15.2 | 3.93 | 17.3 | 30.3 | 142 | 113 | 279 | 613 |
| | Erlang (4) | result | 63.3 | 62.26 | 60.72 | 60.56 | 61.4 | 61.21 | 60.99 | 60.8 | 60.8 | 60.74 | 60.63 |
| | | time | 6.5 | 14.1 | 7.08 | 16 | 6.82 | 32.8 | 25.3 | 127 | 82.3 | 237 | 478 |
| | Exponential | result | 84.79 | 84.26 | 81.96 | 81.72 | 84.57 | 83.09 | 82.52 | 82.09 | 82.09 | 81.92 | 81.65 |
| | | time | 4.1 | 11.1 | 4.64 | 12.7 | 7.3 | 31.5 | 22.5 | 105 | 77.5 | 168 | 418 |

Table A.2: Processed results for the final experiments for 30-*-8 problem instances

| | | Method | IGM/AD | | DM | | RS 30 | | RS 100 | | RS 300 | | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Perturbations | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 2 | 0 |
| 100J-50R-6M | Uniform (20) | result | 186.5 | 186.42 | 186.35 | 186.33 | 186.51 | 186.43 | 186.43 | 186.35 | 186.33 | 186.32 | 186.32 |
| | | time | 37.3 | 134 | 25.6 | 112 | 130 | 487 | 546 | 2166 | 1436 | 4248 | 5401 |
| | Normal (30) | result | 198.37 | 198.29 | 197.83 | 197.8 | 198.29 | 198.03 | 197.94 | 197.89 | 197.83 | 197.81 | 197.75 |
| | | time | 33.7 | 120 | 31.9 | 143 | 80.1 | 404 | 388 | 1722 | 1487 | 3220 | 4388 |
| | Erlang (4) | result | 212.78 | 212.69 | 211.57 | 211.49 | 212.85 | 212.26 | 211.88 | 211.7 | 211.45 | 211.44 | 211.41 |
| | | time | 42.5 | 128 | 43.9 | 162 | 88 | 370 | 299 | 1328 | 1744 | 3200 | 4818 |
| | Exponential | result | 251.69 | 250.77 | 247.89 | 247.53 | 251.04 | 249.75 | 248.75 | 248.05 | 247.52 | 247.47 | 246.73 |
| | | time | 35.8 | 111 | 40.2 | 143 | 53.6 | 236 | 219 | 959 | 785 | 1654 | 4507 |
| 100J-100R-6M | Uniform (20) | result | 173.43 | 173.32 | 173.24 | 173.23 | 173.31 | 173.28 | 173.3 | 173.22 | 173.22 | 173.21 | 173.22 |
| | | time | 48.3 | 182 | 31.5 | 144 | 134 | 548 | 606 | 2581 | 1764 | 4766 | 7901 |
| | Normal (30) | result | 184.7 | 184.41 | 184.07 | 184.02 | 184.76 | 184.38 | 184.18 | 184.09 | 184.01 | 184 | 183.97 |
| | | time | 44.2 | 151 | 38.4 | 171 | 105 | 498 | 452 | 2052 | 1607 | 3644 | 5353 |
| | Erlang (4) | result | 198.75 | 198.1 | 197.22 | 196.99 | 198.62 | 198.07 | 197.56 | 197.42 | 197.23 | 197.15 | 197.03 |
| | | time | 57 | 177 | 53.8 | 205 | 68.9 | 440 | 278 | 1651 | 1327 | 3445 | 6240 |
| | Exponential | result | 236.82 | 235.27 | 232.37 | 232.3 | 238.77 | 236.28 | 234.99 | 233.75 | 232.66 | 232.45 | 231.27 |
| | | time | 57.1 | 187 | 49.4 | 167 | 66.8 | 290 | 227 | 1241 | 1292 | 2652 | 5724 |
| 100J-250R-6M | Uniform (20) | result | 175.77 | 175.64 | 175.72 | 175.6 | 175.73 | 175.7 | 175.69 | 175.61 | 175.64 | 175.59 | 175.64 |
| | | time | 49 | 220 | 45.3 | 238 | 215 | 987 | 716 | 3392 | 2459 | 6541 | 7805 |
| | Normal (30) | result | 187.52 | 187.31 | 187.09 | 186.79 | 187.94 | 187.41 | 187.29 | 186.96 | 186.94 | 186.83 | 186.87 |
| | | time | 52.6 | 210 | 79.8 | 338 | 170 | 729 | 845 | 3453 | 2759 | 6931 | 9849 |
| | Erlang (4) | result | 202.6 | 201.92 | 201.87 | 200.93 | 204.02 | 202.53 | 201.87 | 201.63 | 201.53 | 201.34 | 201.03 |
| | | time | 78 | 234 | 118 | 402 | 141 | 663 | 697 | 3176 | 2023 | 5570 | 10219 |
| | Exponential | result | 245.63 | 244.17 | 244.2 | 242.51 | 250.34 | 247.59 | 246.04 | 244.84 | 242.46 | 242.03 | 240.84 |
| | | time | 65.2 | 255 | 74.7 | 286 | 69.9 | 385 | 351 | 1750 | 2160 | 4588 | 7917 |

Table A.3: Processed results for the final experiments for 100-*-6 problem instances

| | | Method | IGM/AD | | DM | | RS 30 | | RS 100 | | RS 300 | | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Perturbations | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 2 | 0 |
| 100J-50R-12M | Uniform (20) | result | 95.52 | 95.33 | 95.44 | 95.33 | 95.56 | 95.5 | 95.46 | 95.32 | 95.34 | 95.3 | 95.29 |
| | | time | 37.1 | 141 | 29.4 | 116 | 118 | 449 | 365 | 1871 | 1761 | 4485 | 8184 |
| | Normal (30) | result | 105.97 | 105.87 | 105.72 | 105.63 | 106.6 | 106.26 | 105.97 | 105.79 | 105.74 | 105.71 | 105.65 |
| | | time | 36.5 | 131 | 38 | 141 | 78.8 | 347 | 226 | 1470 | 1155 | 3442 | 7178 |
| | Erlang (4) | result | 119.98 | 119.97 | 119.49 | 119.46 | 120.74 | 120.31 | 120.03 | 119.73 | 119.45 | 119.41 | 119.37 |
| | | time | 48.1 | 129 | 44.4 | 147 | 95.4 | 346 | 237 | 1225 | 1309 | 2809 | 5885 |
| | Exponential | result | 157.1 | 156.67 | 155.54 | 155.38 | 159.31 | 158.02 | 157.69 | 156.45 | 155.81 | 155.75 | 155.28 |
| | | time | 41.1 | 142 | 35.5 | 129 | 58.9 | 213 | 177 | 800 | 985 | 1899 | 3678 |
| 100J-100R-12M | Uniform (20) | result | 95.96 | 92.01 | 91.01 | 90.87 | 91.53 | 91.16 | 91.17 | 90.96 | 91.22 | 91.08 | 90.97 |
| | | time | 58.1 | 219 | 52.2 | 189 | 102 | 606 | 611 | 2619 | 2752 | 6399 | 9230 |
| | Normal (30) | result | 104.2 | 102.06 | 101.42 | 101.21 | 102.32 | 101.82 | 101.96 | 101.58 | 101.52 | 101.36 | 101.33 |
| | | time | 53.4 | 191 | 55.5 | 191 | 115 | 470 | 387 | 1842 | 2475 | 4453 | 8219 |
| | Erlang (4) | result | 119.14 | 116.81 | 115.53 | 115.35 | 117.66 | 116.92 | 116.52 | 115.95 | 115.94 | 115.76 | 115.51 |
| | | time | 57.5 | 194 | 61.3 | 203 | 65.6 | 341 | 324 | 1643 | 1653 | 3344 | 6671 |
| | Exponential | result | 157.71 | 154.97 | 152.53 | 151.84 | 156.19 | 155.16 | 154.37 | 153.43 | 152.76 | 152.32 | 151.54 |
| | | time | 48.1 | 153 | 45.6 | 148 | 55 | 228 | 155 | 957 | 1107 | 2510 | 4749 |
| 100J-250R-12M | Uniform (20) | result | 109.94 | 104.12 | 100.88 | 100.18 | 101.25 | 100.49 | 100.84 | 100.38 | 100.76 | 100.24 | 100.64 |
| | | time | 60.5 | 300 | 89.6 | 379 | 131 | 937 | 1238 | 4955 | 3994 | 9304 | 13979 |
| | Normal (30) | result | 120.55 | 115.3 | 113.06 | 111.91 | 114.39 | 113.48 | 113.51 | 112.58 | 112.52 | 112.33 | 112.26 |
| | | time | 56.8 | 272 | 95.2 | 362 | 177 | 873 | 893 | 3693 | 4055 | 8190 | 10759 |
| | Erlang (4) | result | 136.29 | 132.55 | 130.45 | 129.2 | 133.09 | 131.79 | 130.44 | 129.44 | 129.63 | 129.03 | 128.85 |
| | | time | 66.7 | 248 | 89.5 | 302 | 154 | 570 | 735 | 2246 | 2151 | 5585 | 12357 |
| | Exponential | result | 181.19 | 178.21 | 174.72 | 172.79 | 181.33 | 178.51 | 176.8 | 175.37 | 173.85 | 173.55 | 172.49 |
| | | time | 65 | 232 | 75.9 | 242 | 90.8 | 421 | 435 | 1641 | 2081 | 3744 | 7744 |

Table A.4: Processed results for the final experiments for 100-*-12 problem instances

# Appendix B

# Limited run time experiments

The following tables show the best performing algorithms when run time is limited for the final experiments described in Section 5.3. The number between the brackets in the methods (e.g. the 1 in DM (1)) denote the amount of perturbations performed.

| | | Time limit | <10s | <30s | <100s | overall |
|---|---|---|---|---|---|---|
| 30J-15R-4M | Uniform (20) | method | DM (7) | DM (8) | RS 300 (0) | RS 300 (0) |
| | | result | 75.74 | 75.72 | 75.72 | 75.72 |
| | Normal (30) | method | DM (5) | DM (8) | DM (8) | DM (8) |
| | | result | 81.64 | 81.62 | 81.62 | 81.62 |
| | Erlang (4) | method | IGM/AD (3) | DM (7) | DM (7) | DM (7) |
| | | result | 88.85 | 88.73 | 88.73 | 88.73 |
| | Exponential | method | DM (4) | DM (8) | DM (8) | DM (8) |
| | | result | 107.01 | 106.9 | 106.9 | 106.9 |
| 30J-30R-4M | Uniform (20) | method | DM (3) | DM (7) | DM (7) | DM (7) |
| | | result | 77.37 | 77.36 | 77.36 | 77.36 |
| | Normal (30) | method | DM (4) | DM (8) | DM (8) | RS 1000 (0) |
| | | result | 83.06 | 83.06 | 83.06 | 83.05 |
| | Erlang (4) | method | DM (2) | DM (8) | DM (8) | DM (8) |
| | | result | 89.91 | 89.89 | 89.89 | 89.89 |
| | Exponential | method | DM (3) | DM (8) | DM (8) | DM (8) |
| | | result | 107.74 | 107.71 | 107.71 | 107.71 |
| 30J-75R-4M | Uniform (20) | method | DM (2) | DM (8) | RS 100 (5) | RS 100 (8) |
| | | result | 77.24 | 76.85 | 76.67 | 76.65 |
| | Normal (30) | method | DM (2) | DM (8) | DM (8) | DM (8) |
| | | result | 83.95 | 83.43 | 83.43 | 83.43 |
| | Erlang (4) | method | DM (1) | DM (8) | RS 100 (4) | RS 100 (8) |
| | | result | 94.05 | 93.1 | 92.64 | 92.45 |
| | Exponential | method | DM (3) | DM (7) | DM (7) | RS 1000 (0) |
| | | result | 116.79 | 116.07 | 116.07 | 115.55 |
| 30J-15R-8M | Uniform (20) | method | DM (5) | DM (8) | RS 100 (7) | RS 100 (7) |
| | | result | 41.36 | 41.33 | 41.3 | 41.3 |
| | Normal (30) | method | IGM/AD (5) | IGM/AD (5) | IGM/AD (5) | IGM/AD (5) |
| | | result | 47.07 | 47.07 | 47.07 | 47.07 |
| | Erlang (4) | method | DM (3) | DM (7) | DM (7) | DM (7) |
| | | result | 54.9 | 54.89 | 54.89 | 54.89 |
| | Exponential | method | DM (4) | DM (4) | DM (4) | RS 1000 (0) |
| | | result | 74.74 | 74.74 | 74.74 | 74.73 |
| 30J-30R-8M | Uniform (20) | method | DM (5) | RS 30 (7) | RS 100 (6) | RS 100 (6) |
| | | result | 29.79 | 29.79 | 29.78 | 29.78 |
| | Normal (30) | method | DM (4) | DM (5) | DM (5) | DM (5) |
| | | result | 33.24 | 33.23 | 33.23 | 33.23 |
| | Erlang (4) | method | DM (2) | DM (8) | DM (8) | DM (8) |
| | | result | 38.54 | 38.49 | 38.49 | 38.49 |
| | Exponential | method | DM (4) | DM (8) | DM (8) | RS 1000 (0) |
| | | result | 51.98 | 51.94 | 51.94 | 51.93 |
| 30J-75R-8M | Uniform (20) | method | DM (3) | RS 100 (5) | RS 100 (5) | RS 100 (5) |
| | | result | 49.52 | 49.52 | 49.52 | 49.52 |
| | Normal (30) | method | DM (2) | DM (6) | DM (6) | DM (6) |
| | | result | 52.95 | 52.9 | 52.9 | 52.9 |
| | Erlang (4) | method | DM (1) | DM (8) | DM (8) | DM (8) |
| | | result | 60.59 | 60.55 | 60.55 | 60.55 |
| | Exponential | method | DM (3) | DM (5) | DM (5) | RS 1000 (0) |
| | | result | 81.72 | 81.72 | 81.72 | 81.65 |

Table B.1: Best algorithms with limited run time for problem instances with 30 jobs

| | | Time limit | <300s | <1000s | <3000s | overall |
|---|---|---|---|---|---|---|
| 100J-50R-6M | Uniform (20) | method | DM (6) | DM (6) | RS 300 (1) | RS 1000 (0) |
| | | result | 186.33 | 186.33 | 186.33 | 186.32 |
| | Normal (30) | method | DM (6) | DM (6) | DM (6) | RS 1000 (0) |
| | | result | 197.8 | 197.8 | 197.8 | 197.75 |
| | Erlang (4) | method | DM (7) | DM (7) | RS 300 (0) | RS 1000 (0) |
| | | result | 211.48 | 211.48 | 211.45 | 211.41 |
| | Exponential | method | DM (8) | DM (8) | RS 300 (2) | RS 1000 (0) |
| | | result | 247.49 | 247.49 | 247.47 | 246.73 |
| 100J-100R-6M | Uniform (20) | method | DM (4) | DM (4) | RS 300 (0) | RS 300 (2) |
| | | result | 173.23 | 173.23 | 173.22 | 173.21 |
| | Normal (30) | method | DM (8) | DM (8) | RS 300 (1) | RS 1000 (0) |
| | | result | 184.01 | 184.01 | 184 | 183.97 |
| | Erlang (4) | method | DM (5) | DM (5) | DM (5) | DM (5) |
| | | result | 196.99 | 196.99 | 196.99 | 196.99 |
| | Exponential | method | DM (8) | DM (8) | DM (8) | RS 1000 (0) |
| | | result | 232.02 | 232.02 | 232.02 | 231.27 |
| 100J-250R-6M | Uniform (20) | method | DM (4) | DM (7) | DM (7) | DM (7) |
| | | result | 175.6 | 175.59 | 175.59 | 175.59 |
| | Normal (30) | method | DM (4) | DM (8) | DM (8) | DM (8) |
| | | result | 186.86 | 186.78 | 186.78 | 186.78 |
| | Erlang (4) | method | DM (3) | DM (8) | DM (8) | DM (8) |
| | | result | 201.04 | 200.89 | 200.89 | 200.89 |
| | Exponential | method | DM (4) | DM (7) | DM (7) | RS 1000 (0) |
| | | result | 242.51 | 241.93 | 241.93 | 240.84 |
| 100J-50R-12M | Uniform (20) | method | IGM/AD (7) | IGM/AD (7) | RS 100 (8) | RS 1000 (0) |
| | | result | 95.32 | 95.32 | 95.29 | 95.29 |
| | Normal (30) | method | DM (8) | DM (8) | DM (8) | DM (8) |
| | | result | 105.62 | 105.62 | 105.62 | 105.62 |
| | Erlang (4) | method | DM (7) | DM (7) | DM (7) | RS 1000 (0) |
| | | result | 119.39 | 119.39 | 119.39 | 119.37 |
| | Exponential | method | DM (6) | DM (6) | DM (6) | RS 1000 (0) |
| | | result | 155.37 | 155.37 | 155.37 | 155.28 |
| 100J-100R-12M | Uniform (20) | method | DM (7) | DM (7) | DM (7) | DM (7) |
| | | result | 90.87 | 90.87 | 90.87 | 90.87 |
| | Normal (30) | method | DM (8) | DM (8) | DM (8) | DM (8) |
| | | result | 101.14 | 101.14 | 101.14 | 101.14 |
| | Erlang (4) | method | DM (8) | DM (8) | DM (8) | DM (8) |
| | | result | 115.32 | 115.32 | 115.32 | 115.32 |
| | Exponential | method | DM (8) | DM (8) | DM (8) | RS 1000 (0) |
| | | result | 151.71 | 151.71 | 151.71 | 151.54 |
| 100J-250R-12M | Uniform (20) | method | DM (3) | DM (8) | DM (8) | DM (8) |
| | | result | 100.23 | 100.1 | 100.1 | 100.1 |
| | Normal (30) | method | DM (3) | DM (7) | DM (7) | DM (7) |
| | | result | 112.03 | 111.89 | 111.89 | 111.89 |
| | Erlang (4) | method | DM (4) | DM (8) | DM (8) | RS 1000 (0) |
| | | result | 129.36 | 128.97 | 128.97 | 128.85 |
| | Exponential | method | DM (5) | DM (8) | DM (8) | DM (8) |
| | | result | 172.79 | 172.33 | 172.33 | 172.33 |

Table B.2: Best algorithms with limited run time for problem instances with 100 jobs

# Appendix C

# Final experiment result and run time deviations

The following table shows the relative makespan deviation for the final experiments described in Section 5.3 in terms of percentage. For example, when the average makespan is 80 and the relative deviation is 1.2, the absolute makespan deviation is $80 \cdot \frac{1.2}{100} = 0.96$.

| | Method | IGM/AD | | DM | | RS 30 | | RS 100 | | RS 300 | | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Perturbations | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 2 | 0 |
| 30J-15R-4M | Uniform (20) | 0.168 | 0.157 | 0.172 | 0.163 | 0.431 | 0.096 | 0.056 | 0.006 | 0.006 | 0.005 | 0.174 |
| | Exponential | 0.298 | 0.239 | 0.395 | 0.196 | 1.14 | 0.464 | 0.704 | 0.284 | 0.528 | 0.238 | 0.281 |
| 30J-30R-4M | Uniform (20) | 0.087 | 0.011 | 0.059 | 0.011 | 0.125 | 0.02 | 0.081 | 0.018 | 0.009 | 0.017 | 0.004 |
| | Exponential | 0.492 | 0.145 | 0.211 | 0.214 | 0.626 | 0.258 | 0.51 | 0.194 | 0.174 | 0.205 | 0.121 |
| 30J-75R-4M | Uniform (20) | 0.859 | 0.483 | 0.876 | 0.411 | 1 | 0.445 | 0.356 | 0.35 | 0.698 | 0.4 | 0.232 |
| | Exponential | 1.09 | 0.527 | 0.576 | 0.502 | 1.29 | 0.446 | 0.744 | 0.385 | 0.627 | 0.613 | 0.393 |
| 30J-15R-8M | Uniform (20) | 0.227 | 0.266 | 0.308 | 0.222 | 0.348 | 0.294 | 0.524 | 0.187 | 0.549 | 0.147 | 0.332 |
| | Exponential | 0.219 | 0.172 | 0.17 | 0.085 | 0.655 | 0.602 | 0.352 | 0.407 | 0.157 | 0.29 | 0.119 |
| 30J-30R-8M | Uniform (20) | 0.275 | 0.107 | 0.218 | 0.042 | 0.134 | 0.031 | 0.181 | 0.06 | 0.056 | 0.053 | 0.058 |
| | Exponential | 0.267 | 0.123 | 0.111 | 0.229 | 1.97 | 1.31 | 0.714 | 0.2 | 0.205 | 0.174 | 0.265 |
| 30J-75R-8M | Uniform (20) | 0.339 | 0.29 | 0.037 | 0.004 | 0.074 | 0.055 | 0.01 | 0.002 | 0.013 | 0.001 | 0.015 |
| | Exponential | 0.379 | 0.427 | 0.297 | 0.105 | 0.464 | 0.293 | 0.762 | 0.296 | 0.304 | 0.143 | 0.192 |
| 100J-50R-6M | Uniform (20) | 0.015 | 0.032 | 0.009 | 0.005 | 0.045 | 0.049 | 0.066 | 0.008 | 0.009 | 0.01 | 0.008 |
| | Exponential | 0.246 | 0.196 | 0.217 | 0.198 | 0.618 | 0.081 | 0.255 | 0.173 | 0.053 | 0.047 | 0.03 |
| 100J-100R-6M | Uniform (20) | 0.043 | 0.025 | 0.011 | 0.012 | 0.027 | 0.017 | 0.032 | 0.01 | 0.011 | 0.009 | 0.006 |
| | Exponential | 0.239 | 0.124 | 0.136 | 0.096 | 0.482 | 0.416 | 0.392 | 0.192 | 0.18 | 0.129 | 0.168 |
| 100J-250R-6M | Uniform (20) | 0.112 | 0.021 | 0.03 | 0.021 | 0.037 | 0.031 | 0.036 | 0.014 | 0.054 | 0.023 | 0.058 |
| | Exponential | 0.401 | 0.181 | 0.335 | 0.296 | 0.752 | 0.56 | 0.398 | 0.306 | 0.284 | 0.378 | 0.347 |
| 100J-50R-12M | Uniform (20) | 0.271 | 0.008 | 0.112 | 0.048 | 0.149 | 0.127 | 0.249 | 0.043 | 0.065 | 0.034 | 0.035 |
| | Exponential | 0.105 | 0.128 | 0.192 | 0.128 | 0.913 | 0.67 | 0.233 | 0.152 | 0.249 | 0.265 | 0.165 |
| 100J-100R-12M | Uniform (20) | 1.51 | 0.279 | 0.205 | 0.097 | 0.241 | 0.145 | 0.154 | 0.045 | 0.207 | 0.206 | 0.136 |
| | Exponential | 1.04 | 0.323 | 0.313 | 0.202 | 0.818 | 0.752 | 0.362 | 0.491 | 0.307 | 0.251 | 0.232 |
| 100J-250R-12M | Uniform (20) | 1.63 | 1.45 | 0.35 | 0.083 | 0.595 | 0.048 | 0.469 | 0.226 | 0.212 | 0.126 | 0.572 |
| | Exponential | 1.01 | 0.59 | 0.422 | 0.052 | 1.15 | 0.651 | 0.094 | 0.354 | 0.262 | 0.2 | 0.101 |

Table C.1: Relative standard deviations for the final experiments in terms of percentage.

# Appendix D

# Comparison against fixation maximization

The following table compares the average $E(C_{max})$ of the schedules found by the *AGM/AD*, *DM* and *RS 100* algorithms with the optimal solution found by Van Roermund (2013) (in the *Fixation* column) for problem instances from Van Roermund (2013). The best results for each problem instance are printed in bold type.

| | Method | Fixation | IGM/AD | DM | RS 100 |
|---|---|---|---|---|---|
| 20J-10R-5M | Uniform (20) | **48.32** | 48.41 | 48.4 | 48.36 |
| | Normal (30) | 53.84 | 53.89 | **53.71** | 53.88 |
| | Erlang (4) | 60.92 | 61.02 | **60.85** | 61.04 |
| | Exponential | 78.81 | 78.7 | **78.02** | 78.28 |
| 20J-30R-5M | Uniform (20) | **54.32** | 54.35 | 54.35 | 54.35 |
| | Normal (30) | 60.59 | 60.74 | **60.56** | 60.59 |
| | Erlang (4) | **68.95** | 69.24 | 69 | 68.97 |
| | Exponential | 90.24 | 89.93 | **89.04** | 89.7 |
| 30J-40R-2M | Uniform (20) | 134.4 | 134.21 | 134.11 | **134.11** |
| | Normal (30) | 140.25 | 138.89 | **138.23** | 138.25 |
| | Erlang (4) | 147.91 | 145.1 | **142.9** | 143.01 |
| | Exponential | 168.57 | 170.73 | 156.47 | **155.65** |
| 30J-40R-8M | Uniform (20) | 64.47 | 64 | **64** | 64 |
| | Normal (30) | 65.53 | 64.47 | **64.08** | 64.11 |
| | Erlang (4) | 69.18 | 67.97 | **66.08** | 66.16 |
| | Exponential | 85.7 | 84.51 | **80.98** | 81.39 |
| 40J-30R-5M | Uniform (20) | 80.7 | 80.49 | 80.5 | **80.47** |
| | Normal (30) | 87.96 | 87.37 | **87.25** | 87.29 |
| | Erlang (4) | 97.38 | 96.09 | **95.71** | 95.8 |
| | Exponential | 122.01 | 118.41 | **117.49** | 117.75 |
| 40J-50R-5M | Uniform (20) | 95.81 | 95.36 | **95.32** | 95.34 |
| | Normal (30) | 104.09 | 103.36 | **103.18** | 103.26 |
| | Erlang (4) | 114.74 | 113.44 | **112.92** | 113.17 |
| | Exponential | 142.47 | 139.27 | **138.06** | 138.54 |

# Appendix E

# Experimental results for problem instances with large $q_{ij}$

The following tables show the experimental results for the experiments where $q_{ij}$ is exponentially distributed with average $p_i$ for the final experiments described in Section 5.3. Table E.1 shows the results and run times, similar to the tables in Appendix A and Table E.2 shows the best performing algorithms for limited run time, similar to the tables in Appendix B.

| | | Method | IGM/AD | | DM | | RS 30 | | RS 100 | | RS 300 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Perturbations | 0 | 8 | 0 | 8 | 0 | 8 | 0 | 8 | 0 | 3 |
| 30J-15R-5M | Uniform (20) | result | 62.02 | 61.69 | 61.83 | 61.67 | 61.92 | 61.7 | 61.81 | 61.63 | 61.88 | 61.67 |
| | | time | 2.9 | 10.5 | 2.02 | 10.8 | 5.07 | 36 | 14.8 | 118 | 41.6 | 176 |
| | Exponential | result | 95.71 | 95.49 | 95.1 | 95.01 | 97.09 | 95.98 | 95.88 | 95.49 | 95.53 | 95.28 |
| | | time | 2.84 | 11.3 | 1.88 | 10 | 3.81 | 25.9 | 9.57 | 79.3 | 28.3 | 117 |
| 30J-30R-5M | Uniform (20) | result | 62.86 | 61.86 | 62.14 | 61.77 | 62.22 | 61.83 | 62.1 | 61.75 | 62.01 | 61.79 |
| | | time | 2.85 | 12 | 2.22 | 12.5 | 5.78 | 40.6 | 20.5 | 135 | 38.8 | 167 |
| | Exponential | result | 99.56 | 97.85 | 98.59 | 97.73 | 100.34 | 99.03 | 98.37 | 97.68 | 98.28 | 97.68 |
| | | time | 2.91 | 12 | 1.99 | 11.9 | 5.24 | 29.5 | 13.9 | 93.1 | 32.3 | 112 |
| 30J-75R-5M | Uniform (20) | result | 96.84 | 96.15 | 96.06 | 96.06 | 96.07 | 96.06 | 96.06 | 96.06 | 96.06 | 96.06 |
| | | time | 3 | 13.7 | 2.3 | 14.2 | 2.21 | 14 | 3.73 | 30.2 | 8.62 | 35.6 |
| | Exponential | result | 131.94 | 128.48 | 122.87 | 121.68 | 124.56 | 123.17 | 122.62 | 121.75 | 121.99 | 121.84 |
| | | time | 3.12 | 14.6 | 2.71 | 16.9 | 5.24 | 36.9 | 18.5 | 138 | 52.4 | 201 |

Table E.1: Processed results for the experiments for problem instances with exponentially distributed $q_{ij}$

| | | Time limit | <10s | <30s | <100s | overall |
|---|---|---|---|---|---|---|
| 30J-15R-5M | Uniform (20) | method | DM (7) | DM (12) | DM (12) | RS 100 (12) |
| | | result | 61.68 | 61.63 | 61.63 | 61.62 |
| | Exponential | method | DM (6) | DM (6) | DM (6) | DM (6) |
| | | result | 95 | 95 | 95 | 95 |
| 30J-30R-5M | Uniform (20) | method | DM (5) | DM (12) | DM (12) | RS 100 (12) |
| | | result | 61.83 | 61.71 | 61.71 | 61.69 |
| | Exponential | method | DM (6) | DM (11) | DM (11) | RS 100 (10) |
| | | result | 97.85 | 97.66 | 97.66 | 97.64 |
| 30J-75R-5M | Uniform (20) | method | DM (0) | DM (0) | DM (0) | DM (0) |
| | | result | 96.06 | 96.06 | 96.06 | 96.06 |
| | Exponential | method | DM (4) | DM (12) | DM (12) | DM (12) |
| | | result | 122.09 | 121.64 | 121.64 | 121.64 |

Table E.2: Best algorithms with limited run time for problem instances with exponentially distributed $q_{ij}$

# Appendix F

# Experimental results for function evaluations

The following tables show the experimental results for comparison of objective function evaluations on various schedules. A detailed explanation of how these comparisons are performed is given in Section 5.3.1. The run time denotes the total number of seconds required to compute the objective function for all schedules.

| | | Method | $c_{max}$ | IGM/AD | DM | RS 30 | RS 100 | RS 300 | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|
| 30J-15R-4M | Uniform (20) | correlation | 0.9482 | 0.9945 | 0.9967 | 0.9773 | 0.9934 | 0.9974 | 0.9991 |
| | | position offset | 9.52 | 1.91 | 1.6 | 9.75 | 4.59 | 2.81 | 1.84 |
| | | time | 0.157 | 0.223 | 0.101 | 1.65 | 5.53 | 17.4 | 62.8 |
| | Normal (30) | correlation | 0.8359 | 0.9773 | 0.9847 | 0.776 | 0.9083 | 0.9729 | 0.9913 |
| | | position offset | 13.49 | 4.5 | 3.84 | 15.11 | 12.88 | 9.38 | 5.55 |
| | | time | 0.121 | 0.128 | 0.153 | 2.04 | 6.82 | 22.1 | 82 |
| | Erlang (4) | correlation | 0.7856 | 0.9384 | 0.9816 | 0.6712 | 0.8397 | 0.9265 | 0.9808 |
| | | position offset | 15.71 | 8.24 | 3.57 | 21.32 | 14.94 | 9.87 | 7.36 |
| | | time | 0.133 | 0.144 | 0.217 | 2.17 | 7.18 | 23.2 | 84.8 |
| | Exponential | correlation | 0.7109 | 0.8827 | 0.9705 | 0.4938 | 0.6043 | 0.8576 | 0.927 |
| | | position offset | 18.57 | 10.3 | 6.55 | 24.45 | 22.26 | 11.31 | 9.54 |
| | | time | 0.136 | 0.149 | 0.144 | 2.42 | 8.06 | 25.7 | 95.2 |
| 30J-30R-4M | Uniform (20) | correlation | 0.9587 | 0.9956 | 0.9964 | 0.9774 | 0.994 | 0.9983 | 0.9995 |
| | | position offset | 8.56 | 1.66 | 1.5 | 6.83 | 3.42 | 2.09 | 1.34 |
| | | time | 0.089 | 0.097 | 0.11 | 1.59 | 5.29 | 17.1 | 61.3 |
| | Normal (30) | correlation | 0.8764 | 0.9847 | 0.9816 | 0.8628 | 0.9473 | 0.9736 | 0.9931 |
| | | position offset | 12.14 | 3.75 | 2.91 | 13.2 | 7.94 | 6.78 | 3.6 |
| | | time | 0.104 | 0.122 | 0.161 | 1.92 | 6.4 | 20.6 | 74.8 |
| | Erlang (4) | correlation | 0.8762 | 0.9711 | 0.9816 | 0.7364 | 0.8855 | 0.9613 | 0.9835 |
| | | position offset | 13.07 | 5.42 | 4.3 | 18.09 | 13.08 | 8.1 | 6.02 |
| | | time | 0.101 | 0.123 | 0.165 | 1.83 | 6.12 | 19.8 | 72.3 |
| | Exponential | correlation | 0.7803 | 0.9444 | 0.9694 | 0.6532 | 0.816 | 0.9194 | 0.9795 |
| | | position offset | 15.28 | 7.56 | 5.38 | 18.2 | 13.94 | 9.24 | 6.34 |
| | | time | 0.113 | 0.116 | 0.135 | 1.89 | 6.26 | 20.1 | 72.9 |

Table F.1: Results for the function evaluation experiments (part 1)

| | | Method | $c_{max}$ | IGM/AD | DM | RS 30 | RS 100 | RS 300 | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|
| 30J-75R-4M | Uniform (20) | correlation | 0.9815 | 0.9897 | 0.9978 | 0.9955 | 0.9968 | 0.9993 | 0.9998 |
| | | position offset | 4.66 | 3.16 | 1.39 | 2.9 | 1.16 | 0.85 | 0.48 |
| | | time | 0.186 | 0.174 | 0.226 | 2.85 | 9.53 | 29.8 | 101 |
| | Normal (30) | correlation | 0.9416 | 0.9769 | 0.9926 | 0.9694 | 0.9899 | 0.9965 | 0.9988 |
| | | position offset | 8.04 | 4.48 | 2.6 | 6.35 | 3.28 | 2.42 | 1.21 |
| | | time | 0.136 | 0.158 | 0.262 | 2.53 | 8.47 | 26.7 | 90.3 |
| | Erlang (4) | correlation | 0.8803 | 0.9388 | 0.9813 | 0.9031 | 0.9694 | 0.993 | 0.9969 |
| | | position offset | 10.64 | 7.61 | 3.25 | 9.4 | 5.84 | 3.81 | 2.04 |
| | | time | 0.128 | 0.146 | 0.237 | 2.34 | 7.86 | 24.7 | 83.4 |
| | Exponential | correlation | 0.7456 | 0.8782 | 0.9752 | 0.7708 | 0.9238 | 0.9665 | 0.9906 |
| | | position offset | 16.3 | 11 | 5.04 | 15.84 | 7.87 | 7 | 3.58 |
| | | time | 0.138 | 0.159 | 0.206 | 2.6 | 8.66 | 27.4 | 92.7 |
| 30J-15R-8M | Uniform (20) | correlation | 0.9609 | 0.9989 | 0.9998 | 0.9913 | 0.9977 | 0.9993 | 0.9997 |
| | | position offset | 6.51 | 1.24 | 0.42 | 3.15 | 1.94 | 0.99 | 0.71 |
| | | time | 0.139 | 0.125 | 0.1 | 1.77 | 5.88 | 18.7 | 76.7 |
| | Normal (30) | correlation | 0.9187 | 0.9952 | 0.9991 | 0.9674 | 0.9871 | 0.9945 | 0.9977 |
| | | position offset | 8.5 | 2.54 | 1.17 | 8.85 | 5.16 | 3.42 | 1.73 |
| | | time | 0.119 | 0.135 | 0.122 | 1.88 | 6.29 | 20.1 | 75.7 |
| | Erlang (4) | correlation | 0.8948 | 0.987 | 0.9958 | 0.8909 | 0.9413 | 0.9802 | 0.9928 |
| | | position offset | 10.27 | 4.16 | 2.27 | 16.2 | 9.63 | 6.6 | 3.66 |
| | | time | 0.103 | 0.135 | 0.118 | 1.77 | 5.9 | 18.6 | 76.2 |
| | Exponential | correlation | 0.8388 | 0.9443 | 0.9839 | 0.6458 | 0.812 | 0.9448 | 0.9749 |
| | | position offset | 12.09 | 9.14 | 4.79 | 17.09 | 15.49 | 11.33 | 6.51 |
| | | time | 0.099 | 0.131 | 0.095 | 1.74 | 5.76 | 18.2 | 68.6 |

Table F.2: Results for the function evaluation experiments (part 2)

| | | Method | $c_{max}$ | IGM/AD | DM | RS 30 | RS 100 | RS 300 | RS 1000 |
|---|---|---|---|---|---|---|---|---|---|
| 30J-30R-8M | Uniform (20) | correlation | 0.964 | 0.9965 | 0.9995 | 0.9937 | 0.9971 | 0.9996 | 0.9998 |
| | | position offset | 12.03 | 3.16 | 0.95 | 2.52 | 2.19 | 1.11 | 0.59 |
| | | time | 0.116 | 0.143 | 0.128 | 2.11 | 6.97 | 22.6 | 84.5 |
| | Normal (30) | correlation | 0.8939 | 0.9703 | 0.9969 | 0.9548 | 0.9886 | 0.9968 | 0.9985 |
| | | position offset | 15.12 | 8.75 | 2.26 | 8.57 | 5.31 | 2.62 | 1.89 |
| | | time | 0.142 | 0.153 | 0.169 | 2.38 | 7.97 | 25.5 | 101 |
| | Erlang (4) | correlation | 0.8206 | 0.9397 | 0.9819 | 0.8844 | 0.9713 | 0.9883 | 0.9964 |
| | | position offset | 15.34 | 10.63 | 3.95 | 13.38 | 7.89 | 5.85 | 3.64 |
| | | time | 0.135 | 0.144 | 0.158 | 2.25 | 7.5 | 23.9 | 95 |
| | Exponential | correlation | 0.5649 | 0.832 | 0.9577 | 0.6817 | 0.8846 | 0.9462 | 0.9863 |
| | | position offset | 25.9 | 16.07 | 9.87 | 21.76 | 15.17 | 14.22 | 7.24 |
| | | time | 0.136 | 0.152 | 0.125 | 2.18 | 7.25 | 23 | 91.3 |
| 30J-75R-8M | Uniform (20) | correlation | 0.9567 | 0.9816 | 0.9935 | 0.9915 | 0.9976 | 0.9997 | 0.9996 |
| | | position offset | 18.12 | 10.39 | 3.52 | 4.62 | 2.12 | 2.04 | 1.58 |
| | | time | 0.213 | 0.249 | 0.226 | 3.84 | 12.8 | 41.1 | 146 |
| | Normal (30) | correlation | 0.9072 | 0.9454 | 0.9912 | 0.979 | 0.9921 | 0.9979 | 0.9993 |
| | | position offset | 18.18 | 11.59 | 2.36 | 6.51 | 4.17 | 2.38 | 0.9 |
| | | time | 0.209 | 0.242 | 0.277 | 3.76 | 12.6 | 41.4 | 142 |
| | Erlang (4) | correlation | 0.828 | 0.883 | 0.9619 | 0.9525 | 0.9793 | 0.9917 | 0.9982 |
| | | position offset | 16.99 | 11.74 | 3.03 | 9.32 | 5.92 | 3.73 | 2.3 |
| | | time | 0.194 | 0.257 | 0.264 | 3.64 | 12.1 | 39.2 | 142 |
| | Exponential | correlation | 0.7033 | 0.8394 | 0.9805 | 0.8393 | 0.9418 | 0.9754 | 0.994 |
| | | position offset | 23.07 | 14.86 | 4.81 | 15.62 | 8.1 | 6.18 | 3.57 |
| | | time | 0.174 | 0.198 | 0.177 | 3.07 | 10.2 | 32.2 | 112 |

Table F.3: Results for the function evaluation experiments (part 3)

# Bibliography


Bibinger, Markus (2013). "Notes on the sum and maximum of independent exponentially distributed random variables with different scale parameters". In: *arXiv preprint arXiv:1307.3945*.

Diepen, Guido et al. (2013). "Robust planning of airport platform buses". In: *Computers & Operations Research* 40.3, pages 747–757.

Gambardella, L.M. and M. Mastrolilli (1996). "Effective neighborhood functions for the flexible job shop problem". In: *Journal of Scheduling* 3.3.

Graham, R.L. et al. (1979). "Optimization and approximation in deterministic sequencing and scheduling: a survey". In: *Annals of Discrete Mathematics* 5.2, pages 287–326.

Greene, William H (2003). *Econometric analysis*. Pearson Education India.

Hoppenbrouwer, D.J. (2011). "Robust parallel machine scheduling with relations between jobs". Master's thesis. Universiteit Utrecht.

Kelton, W David and Averill M Law (2000). *Simulation modeling and analysis*. McGraw Hill Boston.

Lenstra, J.K. et al. (1977). "Complexity of machine scheduling problems". In: *Annals of discrete mathematics* 1, pages 343–362.

Lourenço, H.R. et al. (2003). *Iterated local search*. Springer.

Mladenović, N. and P. Hansen (1997). "Variable neighborhood search". In: *Computers & Operations Research* 24.11, pages 1097–1100.

Nadarajah, Saralees and Samuel Kotz (2008). "Exact distribution of the max/min of two Gaussian random variables". In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16.2, pages 210–212.

Olive, David (2008). "Truncated distributions". In: *Applied Robust Statistics*, pages 104–130.

Roy, B. and B. Sussmann (1964). "Les problemes dordonnancement avec contraintes disjonctives". In: *Note ds* 9.

Schaller, Peter and Grigory Temnov (2008). "Efficient and precise computation of convolutions: applying fft to heavy tailed distributions". In: *Computational Methods in Applied Mathematics Comput. Methods Appl. Math.* 8.2, pages 187–200.

Vaessens, R.J.M. (1995). *Generalized job shop scheduling: complexity and local search.* Eindhoven University of Technology Eindhoven, The Netherlands.

Van Roermund, D.J. (2013). "Robustness in parallel machine scheduling". Master's thesis. Universiteit Utrecht.

Van den Akker, J.M. et al. (2013). "Finding robust solutions for the stochastic Job Shop Scheduling problem by including simulation in local search". In: *Experimental Algorithms.* Springer, pages 402–413.