Utrecht University

MSc Computing Sciences

Master Thesis

Combining Normal Approximation Heuristics With Simulated Annealing For The Stochastic Job-Shop Scheduling Problem

Author:
F.H.J. (Floris) de Kruijff

Supervisor:
Dr. J.A. (Han) Hoogeveen
Dr. ir. J.M. (Marjan) van den Akker

Examination date:
December 18, 2024

Daily supervisor:
C. (Casper) Loman MSc

Department of Information and Computing Sciences

Utrecht University

Abstract

This thesis addresses the Stochastic Job-Shop Scheduling Problem, incorporating uncertainty in the processing time. Efficiently creating robust schedules is computationally challenging due to the complex search space and stochastic variables. In this work, we adapt an approximation method previously used for Parallel Machine Scheduling, where normal approximation techniques are used as heuristic in a Simulated Annealing framework. By approximating the expected makespan, the algorithm significantly reduces the computational burden compared to traditional Monte Carlo simulations. We conducted a series of experiments on benchmark instances to evaluate the performance of the approximation method, assessing the percentiles of the resulting distribution together with the computation time. The results show that the method outperforms sampling-based methods in both solution quality and computational effort when there is much variability in the distributions and show comparable results with lower variability. This work advances heuristic scheduling methods by integrating probabilistic approximations into a metaheuristic framework, offering a scalable solution for large and uncertain scheduling environments.

Title: Combining Normal Approximation Heuristics With Simulated Annealing For The Stochastic Job-Shop Scheduling Problem
Author: F.H.J. (Floris) de Kruijff, f.h.j.dekruijff@student.uu.nl, 6359248
Supervisor: Dr. J.A. (Han) Hoogeveen, Dr. ir. J.M. (Marjan) van den Akker
Daily supervisor: C. (Casper) Loman MSc
Second Examiner: Dr. ir. J.M. (Marjan) van den Akker
Examination date: December 18, 2024

Department of Information and Computer Science
Utrecht University
Princetonplein 5, 3584 CC Utrecht
https://uu.nl/en/organisation/department-of-information-and-computing-sciences

Contents

1. Introduction

Scheduling is a decision-making process that involves the allocation of limited resources over time to perform a set of tasks. This abstract concept finds its place in many aspects of our lives. In computation, scheduling is needed to allow for multitasking on a processor or by the operating system to prioritize tasks. It finds use in logistics and planning, where resources such as personnel, equipment, and transportation must be efficiently assigned to meet deadlines and reduce costs. In manufacturing, scheduling helps streamline the flow of materials and production processes to maximize efficiency and minimize delays. Similarly, in healthcare, scheduling ensures the optimal use of medical personnel, facilities, and equipment, in order to improve patient care and reduce waiting times. There are countless examples where scheduling can increase efficiency and reduce cost. Constructing these schedules can become computationally difficult when the set of requirements increases, and therefore our focus lies on efficiently and effectively computing these schedules for complex problems.

We restrict ourselves to the *Job-Shop Scheduling Problem* (JSSP), an abstract model that describes a set of scheduling problems in which tasks must be executed on machines in a specified order. Consider, for example, a manufacturing floor where a product requires processing across multiple machines. (e.g., the following job sequence: {*drill hole, apply plate, fasten screw*}. These three operations must be performed by different machines and executed in this specific order, with each operation having a distinct processing duration. This can be generalized by stating that there are n jobs, each consisting of m operations, that need to be executed on m machines. The ordering of operations on the machines is to be determined. A theoretical upper bound for the total number of possible sequences is given by $(n!)^m$. To illustrate this scale of this problem, observe that the number of possible sequences for just 6 jobs and 6 machines is approximately $1.39 \cdot 10^{17}$. This number is so large that verifying 1 million schedules per second would still require over 4000 years to verify all schedules individually for optimality. In practice, efficiently solving scheduling problems can lead to significant cost and time savings, where delays and resource constraints can have a large impact on overall productivity. These problems are characterized by much higher values for n and m, and stress the need for algorithms that solve these problems efficiently and effectively.

We expand upon this definition of JSSP with the *Stochastic Job-Shop Scheduling Problem* (SJSSP), including uncertainty by introducing one or more stochastic variables in parts of the model. This is important since uncertainty models real-world scenarios more accurately; for instance, trucks may be delayed, machines may break down or personnel becoming unavailable. Different causes of delay can exist in different parts of the model. By incorporating uncertainty in the processing time we will be able

to construct robust schedules that are less sensitive to these disturbances. However, including this uncertainty increases the computational burden. Both the uncertainty and the combinatorial difficulty of the problem require us to explore sophisticated methods to find high-quality schedules within a reasonable time.

To solve SJSSP efficiently, we will use local search algorithms. The objective of this set of algorithms is to approximate the optimal solution by iteratively improving the current solution using local information. Consequently, these algorithms have better results exploring the large search space and potentially converge to a global optimum. Local search methods are particularly suitable for complex scheduling problems like JSSP because they balance solution quality with computation times, especially in cases where exact methods would be impractical due to the large solution space. For example, local search has been successfully applied to problems in logistics, such as vehicle routing and manufacturing, where robust schedules are needed to handle unexpected delays.

In this thesis, we will extend the work of Passage (2016), which used an approximation method for the normal distribution to compute the maximum completion time in the context of Parallel Machine Scheduling. We adapt this method for job-shop scheduling and compare this approximated distribution with traditional Monte-Carlo simulations (van den Akker et al., 2013). Both these methods compute the quality of a given schedule, needed by the local search algorithm. The performance of the algorithm is compared between these methods and different simulation numbers. We expect that the approximation methods perform significantly faster than the simulations while maintaining at least the same, if not better, quality approximations. We also compare different levels of stochasticity in well-known job-shop instances in literature and assess the robustness of the produced schedules by evaluating their quality variance.

Outline

The structure of this thesis is organized as follows: Chapter 2 provides a formal definition of JSSP, including its mathematical notation and a precise computational method. This chapter subsequently introduces the stochastic variation of this problem along with the relevant stochastic notation. This is followed by a review of the literature in Chapter 3, which covers deterministic methods for JSSP, followed by the exploration of methods applicable to the stochastic variant covering both exact and heuristic search methods. In Chapter 4 the algorithmic methods are described, particularly focusing on Simulated Annealing and Iterated Local Search. The results of these methods are discussed in Chapter 5. Lastly, Chapter 6 concludes the research and presents the final observations, along with recommendations for future research.

2. Problem Description

The job-shop scheduling refers to the scheduling of jobs in a shop environment. In this chapter, we will formally define the *Job-Shop Scheduling Problem* (JSSP) in Section 2.1, introduce its Disjunctive Graph Notation and Mixed Integer Programming (MIP) formulations. Finally, in Section 2.2, we will extend these definitions to include stochastic variables and present the objective function *expected makespan*.

2.1. Job-Shop Scheduling

The Job-Shop Scheduling Problem is a fundamental problem in operations research and management theory, with numerous practical applications. This theoretical model can be formalized as follows. Given a set of n jobs and m machines, our objective is to determine a schedule that minimizes the makespan, which is defined as the completion time of the last job. Each job consists of m operations that must be executed in a specified order, with each operation scheduled on a different machine. There is no predefined ordering between different jobs. An operation cannot be preempted and a machine can handle only a single operation at a time. Every operation has a processing time associated with a machine and a machine at which it must be executed. Given the start time of an operation, we can calculate its completion time. Creating these schedules is closely related to project scheduling, which employs the notion of a *critical path*. The critical path is the sequence of tasks in a project that directly determines its minimum completion time. Any delay in tasks on the critical path will delay the overall project timeline, as they have no slack time. The completion time of the critical path is the same as the *makespan* in scheduling.

Minimizing this makespan is one example of a metric used to optimize the search process; however, other metrics also exist, including *maximum lateness*, *total weighted completion time*, *total weighted tardiness*, *weighted number of tardy jobs* (Pinedo, 2008). Lateness is defined as the difference between the completion time of an operation and its due date. Tardiness is equal to the maximum of 0 and the lateness. In this research, we will focus on the makespan as the quality measure for schedules.

The Job Shop Scheduling Problem (JSSP) is a variation of several scheduling problems, each with unique constraints and requirements. Starting with the simpler *Parallel Machine Scheduling Problem* (PMSP), this problem involves scheduling jobs across multiple machines that operate in parallel. In PMSP, jobs can be assigned to any available machine. In contrast, JSSP introduces stricter requirements, focusing on a set of jobs where each job has an ordered sequence of operations that must be performed on specific machines. Each machine has a capacity of one and must execute operations

in a fixed order for each job. More complex than PMSP but related, the *Open-Shop Scheduling Problem* (OSSP) and *Flow-Shop Scheduling Problem* (FSSP) involve additional constraints on the order and allocation of jobs. The OSSP allows operations to be scheduled on a prespecified machine without a fixed sequence, while the FFSP enforces a specific order of operations across all jobs. At a more general level, these problems are all specialized cases of the broader *Resource-Constrained Project Scheduling Problem* (RCPSP). RCPSP allows resources, such as machines, to be flexibly allocated to tasks across various jobs, but this flexibility also expands the search space and complexity. While shop scheduling and PMSP are concerned specifically with machines, RCPSP deals with a wider range of resources and constraints, offering even greater versatility in scheduling, but at the cost of increased computational complexity.

2.1.1. Mathematical Representation

To describe JSSP, we will introduce the notation as seen in Table 2.1. Stochastic variables will be elaborated on in Section 2.2. Using the objective function and the constraints, we can introduce the scheduling notation introduced by Graham et al. (1977) in Equation (2.1). The three fields α, β, γ in Equation (2.1), separated by the pipe, can describe

Symbol	**Description**
J_j	Job j, where $j \in \{0, ..., n-1\}$ for n jobs
M_k	Machine k, where $k \in \{0, ..., m-1\}$ for m machines
O_j	Set of operations for job j
O_{ij}	Operation i of job j
N	Number of total operations $(n \cdot m)$
$m_{\prec}(O_{ij})$	The machine predecessor of operation O_{ij}
$m_{\succ}(O_{ij})$	The machine successor of operation O_{ij}
s_{ij}	Deterministic start time of operation O_{ij}
p_{ij}	Deterministic processing time of of operation O_{ij}
c_{ij}	Deterministic end time of operation O_{ij}, $c_{ij} = s_{ij} + p_{ij}$
m_{ij}	Denotes the machine that will perform operation O_{ij}
$c_{max}(\tau)$	Deterministic makespan of a given schedule τ

Table 2.1.: Mathematical notation

many variations of the scheduling problem. Job-shop problems are defined by a J for α, no additional constraints for β and γ defines the objective function to optimize over - in this case minimizing the makespan or c_{max}. Here, c_{max} is the maximum completion time for all operations, as we can see defined in Equation (2.2).

$$J||c_{max} \tag{2.1}$$

Formal Definition

Using the notation of the previous section, we will now declare a more formal statement. Let there be a set of jobs J with n elements, where J_j denotes the j^{th} job in that set. There is also a set of machines M with m elements, where M_k denotes the k^{th} machine in that set, with each machine having a capacity of exactly 1 (that is, one operation simultaneously). Each J_j is a set of exactly m operations, given by O_j. Operation i of job j is given by O_{ij}. We can include jobs with less than m operations by including operations with processing time 0. There exists an order such that $O_{1j} \prec O_{ij} \prec O_{(i+1)j} \prec O_{mj}\ \forall i \in O_j,\ \forall j \in J$. This states that operation O_{ij} precedes $O_{(i+1)j}$. This ordering within a job is called *job sequence*. The operation sequence on a machine, called *machine sequence*, is unknown. The goal of JSSP is to find an ordering in this sequence such that it is optimal given the metric over which it is optimized over. Each operation has a processing time p_{ij}, that is zero or more. A feasible solution is called a *schedule* and is defined as an ordering of operations per machine such that no cycles exist in the machine cliques, the execution order of the operations per machine. We will see a graphical representation of these precedence constraints in Section 2.1.2. The start times are chosen in such a way that each operation is started as soon as possible. We assume that no blocking can occur due to operations waiting to be processed, effectively simulating infinite machine buffers. Summarizing the machine requirements:

- A machine can handle a single operation at the time;
- An operation cannot be interrupted during execution, i.e. no preemption;
- An operation cannot start if its predecessors have not yet been completed.

With as goal to construct a schedule with minimal makespan, using our newly introduced notation we can now formally define this in Equation (2.2).

$$c_{max} \coloneqq \max_{\substack{i=m-1 \\ 0 \leq j < n}} \{s_{ij} + p_{ij}\} \tag{2.2}$$

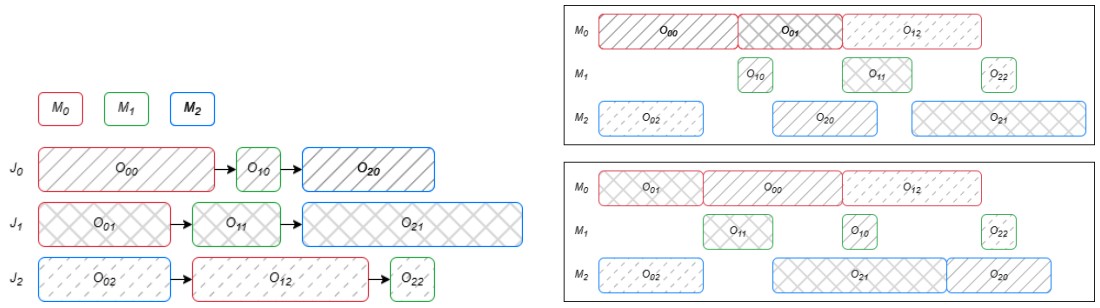
Illustrative Example

Consider a $3 \cdot 3$ problem, as seen in Table 2.2. With the job sequence visualized in Figure 2.1a, the different lines indicate the different jobs the operations belong to. Figure 2.1b displays two potential schedules as Gantt charts, where the operations are plotted with time on the horizontal axis and assigned machine on the vertical axis. Operations are moved to the left as far as possible to make a compact schedule.

	O_{0j}	O_{1j}	O_{2j}
O_{i0}	$(0, 4)$	$(1, 1)$	$(2, 3)$
O_{i1}	$(0, 3)$	$(1, 2)$	$(2, 5)$
O_{i2}	$(2, 3)$	$(0, 4)$	$(1, 1)$

Table 2.2.: Example problem instance with tuples (m_{ij}, p_{ij})

Given the two schedules in Figure 2.1b, we find that the second schedule has a shorter makespan than the first. In the second schedule, O_{20} defines the makespan. Here O_{21} defines the start time as its machine predecessor. This is in turn defined by O_{11} as its preceding job operation. O_{11} has a single predecessor, O_{01}, from which we start at the beginning $T = 0$. This path we have followed from the back towards the start of the schedule is the critical path and is visualized in Figure 2.2. A critical path can be decomposed into *critical blocks*, which defines sequence of operations on the same machine; these critical blocks change when a machine finishes an operation and another operation starts.



(a) Three jobs with three operations

(b) Two potential schedules

Figure 2.1.: Example problem with two example schedules

Schedule types

JSSP is $\mathcal{NP}$-*hard* in the strong sense, proven by Garey et al. (1976) with a reduction from *3-partition*. Looking at the previous example, it is difficult to see how this problem explodes when n and m increase. We have seen previously that for a small number of six jobs and six machines, the number of combinations is tremendous. Although $(n!)^m$ is seen as a weak upper bound, it gives a good estimate of the order of magnitude. In

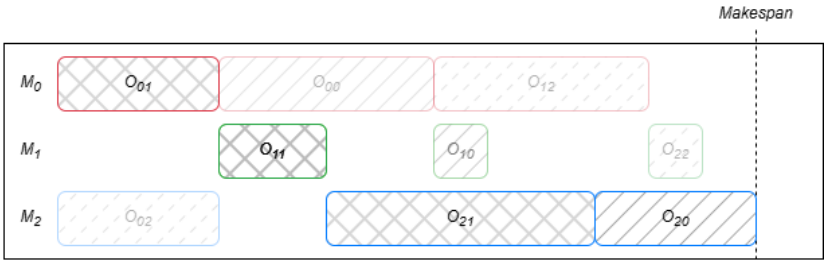
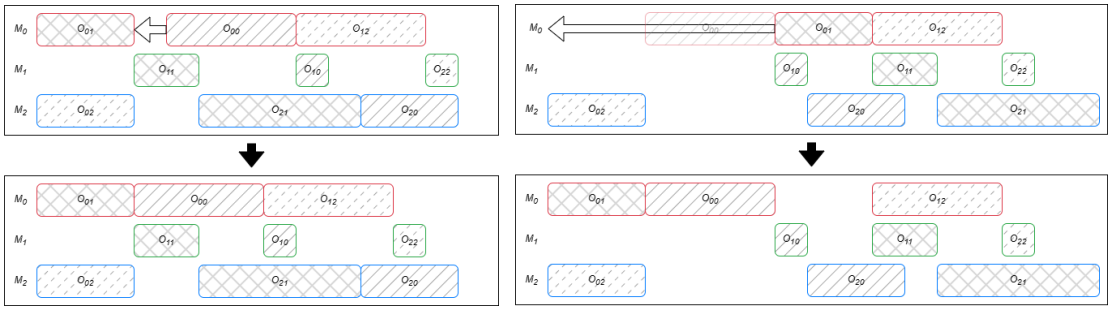


Figure 2.2.: One of the two critical paths of second schedule in Figure 2.1b

our example, there will be a total of 216 possible combinations given machine and job sequences assuming operations are started as soon as possible. It becomes evident that we are only interested in schedules where there is no unnecessary delay between machine operations - without this characteristic the search space would include infinitely many options since we can always introduce some slack somewhere. This subset of solutions we will look at are called *active schedules* (Conway et al., 1967).

Firstly, *semi-active schedules* ensure that no operation can be moved earlier by a *limited-left-shift*, where each operation is shifted left as far as possible without altering the order of operations on a machine or a job (Figure 2.3a). Narrowing further, active schedules allow no operation to be shifted earlier by any *left-shift* move, meaning no unnecessary idle time between operations (Figure 2.3b). Both types reduce the search space, but active schedules are more constrained. We will find that in our implementation of the search process, the constructed schedules will always meet the active schedule criteria.



(a) Limited left-shift operation (b) Left-shift operation

Figure 2.3.: Shift operators on schedules

This allows us to consider a finite set of active schedules, since there exists only one semi-active schedule for each ordering of operations, and there are finitely many orderings. However, this set is still extremely large and an exhaustive enumeration and comparative evaluation are not feasible.

2.1.2. Disjunctive Graph Representation

It is common to model a problem instance as a disjunctive graph, as proposed by Roy and Sussmann (1964). The graph will express both job precedence constraints and machine precedence constraints in the form of conjunctive arcs and disjunctive edges respectively. These definitions are symmetric to the job sequence and the machine sequence. A conjunctive arc (i, j) in scheduling corresponds to the logical AND operator; this arc means that the operation cannot commence before the preceding operation is complete. The undirected disjunctive edge corresponds to the logical OR operation and can be interpreted as the fact that either of the two operations can start and sequentially the other, but not both simultaneously. Which operation will precede the other has yet to be decided by the search procedure.

We define the graph as $G = (V, A, E)$ where V consists of a start vertex V_s, a sink vertex V_e, and for each operation O_{ij} a vertex V_{ij}. The vertex set is described in Equation (2.3).

$$V = (\bigcup_{i=0}^{m-1} \bigcup_{j=0}^{n-1} V_{ij}) \cup V_s \cup V_e \tag{2.3}$$

The weight of the vertex will be set as its corresponding processing time p_{ij}, and set to 0 for the dummy vertices. To construct the paths between the vertices, we include arcs and edges with the sets A and E. The conjunctive arcs are all the arcs from the start vertex to the first operation in a job; the arcs between operations in the same job; and arcs from the last operation in a job to the sink vertex. For the start vertex to all the first operations of all jobs, we have: $(V_s, V_{0j})\ \forall j \in [n]$. The same is done for every last operation of the job to the sink vertex: $(V_{mj}, V_e)\ \forall j \in [n]$. Finally, all the arcs between operations within a job can be added, specifically: $(V_{ij}, V_{i+1j})\ \forall j \in [n], \forall i \in [m]$. This concludes A as described in Equation (2.4).

$$A = \bigcup_{j=0}^{n-1} (V_s, V_{1j}) \cup \bigcup_{j=0}^{n-1} (V_{mj}, V_e) \cup \bigcup_{i=0}^{m-1} \bigcup_{j=0}^{n-1} (V_{ij}, V_{i+1j}) \tag{2.4}$$

Disjunctive edges will initially be undirected. Every pair operation planned to be scheduled on the same machine will receive a disjunctive edge: $\{V_{ij}, V_{i'j'}\}$ if $m_{ij} = m_{i'j'}$.

$$E = \{\{V_{ij}, V_{i'j'}\} \mid i, i' \in [m],\ j, j' \in [n],\ m_{ij} = m_{i'j'}\} \tag{2.5}$$

Longest Path

Finding a feasible schedule means converting all edges from E in such a way that all cliques that represent a machine become acyclic. After this we end up with a *Directed Acyclic Graph* (DAG) where the edges in the graph are directed and no cycles exist. The longest path in the DAG will represent the critical path of the schedule, and the final operation of that critical path will define the makespan and therefore the quality of the schedule. This critical path is defined in the graph using critical arcs. We can find such a longest path using known algorithms such as Bellman-Ford, which finds the

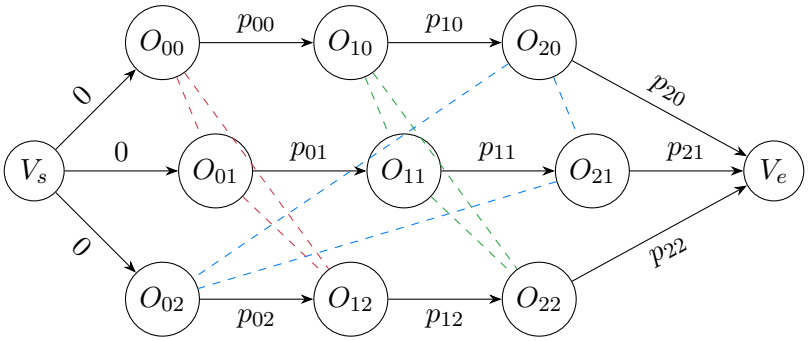


Figure 2.4.: Disjunctive graph representation from example in Table 2.2.

longest path in $\mathcal{O}(V \cdot E')$ if a topological sorting of the graph is not possible (where $E' = E + A$, in this case). A topological sorting sorts the vertices of the graph in a linear ordering such that all the precedence constraints of the DAG are maintained. With such a sorting, a linear-time complexity algorithm can be used, such as a topological-based dynamic programming method that can compute the longest path in $\mathcal{O}(V + E')$. In this research, we will use a variation of the topological-based method together with a linked-list formulation. We make use of the fact that, for job-shop scheduling, any operation can have at most two predecessors in the graph, simplifying the graph traversal significantly. This formulation also computes the longest path in $\mathcal{O}(V + E')$.

2.1.3. Mixed Integer Programming Notation

Using the disjunctive graph representation introduced in Section 2.1.2 we can continue to define a *Mixed Integer Programming* (MIP) model. The conjunctive and disjunctive edges allow us to construct the constraints for the model. We refer to the following definition for the MIP formulation of JSSP by Broek, van den (2009). An operation k here is defined by its start time S_k and their disjunctive edges $\forall\{i, k\} \in D$. The decision variable x is then defined as follows.

$$x_{ij} = \begin{cases} 1 & \text{if } i \text{ is scheduled before } j \text{ on same machine} \\ 0 & \text{otherwise} \end{cases}$$

The MIP will minimize the start time of the dummy vertex V_e, as seen in (MIP.1). The constraints correspond to the conjunctive and disjunctive constraints introduced in the graph representation. M equals the sum of all processing times, Equation (2.6). Equation (MIP.2) depicts the interoperation constraints within a job as conjunctive constraints where an operation cannot start until its predecessor is finished. Equations (MIP.3) and (MIP.4) show the disjunctive constraints. Finally, (MIP.5) and (MIP.6) denote the domains of the variables.

$$M = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} p_{ij} \tag{2.6}$$

$$
\begin{array}{lllr}
\text{minimize} & S_e & & \text{(MIP.1)} \\
\text{subject to} & S_k \geq S_i + p_i & \forall (i,k) \in E & \text{(MIP.2)} \\
& S_k \geq S_i + p_i - M(1 - x_{ik}) & \forall (i,k) \in A & \text{(MIP.3)} \\
& x_{ij} + x_{ji} = 1 & \forall \{i,j\} \in A & \text{(MIP.4)} \\
& x_{ij} \in \{0,1\} & \forall \{i,j\} \in A & \text{(MIP.5)} \\
& S_i \in \mathbb{R}^+ & & \text{(MIP.6)}
\end{array}
$$

Although this model describes an optimal solution exactly, in practice it is infeasible to calculate it due to the number of constraints. From Equation (2.7) and Equation (2.8) we find that the number of edges and arcs increases significantly for large instance sizes and therefore the number of constraints in the MIP.

$$|A| = n + (n \cdot m) \tag{2.7}$$

$$|E| = m \cdot \binom{n}{2} = m \cdot \frac{n \cdot (n-1)}{2} \tag{2.8}$$

Given the use of decision variables, a branch-and-bound algorithm can be implemented, but even so this formulation leads to long computation times for small instances. In Chapter 4 we will instead look at local search methods that can quickly navigate this vast search space and find objective values near the optimum, whereas MIP formulations often strictly find global optimal objective values given the constraints. Variations of MIP exist, such as moving the strict constraints into the objective function using the Lagrangian relaxation. Even so, these exact methods will have an increased difficulty navigation the increased search space of the stochastic variant of job-shop.

2.2. Stochastic Job Shop Scheduling

Let us turn our attention to the stochastic version of job-shop, *Stochastic Job-Shop Scheduling Problem* (SJSSP). This generalization of the JSSP incorporates uncertainty by introducing stochastic variables for one or more elements of the deterministic problem. To see why this is useful, let us look at the concrete implementations of job-shop. More often than not, the schedule can experience delays due to machine breakdowns, late deliveries, human error, etc. If we can include these uncertainties in the schedule, the schedule will be more robust and more applicable to real-world scenarios.

This uncertainty can be incorporated into different aspects of the model. Disruptions can occur at a job level, e.g. when preconditions for a job are not met when the first operation commences, affecting all the operations in the job. Disturbances can occur across jobs, e.g. for all operations being handled on the same machine if it becomes unavailable or breaks down. With all these uncertainties in potentially various different aspects of the model, it becomes challenging to determine the makespan of the schedule, since this depends on multiple stochastic variables. In this Section, we will extend the

SJSSP model from the previous Section to adequately justify the methods employed in Chapter 4.

2.2.1. Stochastic Processing Times

For this research, we will limit the domain of uncertainty to the processing duration of the operations. Although different probability distributions can be employed to better represent the uncertainty concerning the execution of an operation, we will limit ourselves to the normal distribution. Other options as probability distributions include the Uniform, Exponential, 4-Erlang and Log-normal distributions. We refer to the thesis of Passage (2016) for a conversion between these distributions and experiments on SJSSP for different distributions.

We can introduce a stochastic factor in existing deterministic instances to include some arbitrary variation in the problem instances. For deterministic instances, all operations are defined by some processing time p_{ij}. To introduce variability, we set the variance to be some factor α of p_{ij} and p_{ij} as the mean, which results in Equation (2.9).

$$P_{ij} \sim \mathcal{N}(\mu = p_{ij}, \sigma^2 = \alpha \cdot p_{ij}) \tag{2.9}$$

In the rest of this research we will assume that the probability distributions are known and mutually independent. Having stochastic processing times also implies that all other properties related to the processing times are also stochastic.

2.2.2. Expected Makespan

We extend the notation from Table 2.1 with the stochastic variables in Table 2.3. Since the makespan, the start time of the sink vertex, now is a stochastic variable we need to reformulate the makespan as the expected value of this variable, as depicted in Equation (2.10) we now maximise over the start time of an operation including its stochastic processing time.

$$C_{max} \coloneqq \mathbb{E}[\max_{\substack{i=m-1 \\ 0 \leq j < n}} \{S_{ij} + P_{ij}\}] \tag{2.10}$$

This lets us define the problem using Graham's three-field notation in Equation (2.11), as we have seen for the deterministic version. We include the stochastic processing duration $P_{ij}(\sigma)$ for some probability distribution σ and minimize over the *expeced makespan.*

$$J|P_{ij}(\sigma)|C_{max} \tag{2.11}$$

One of the methods to evaluate the expected makespan is called *result sampling* (van den Akker et al., 2013). With result sampling, we sample from each probability distribution and calculate the start and completion times of all operations in order to find the makespan. This makespan can be erratic based on the samples drawn from the different distributions. To mitigate this, we repeat this procedure multiple times to average out

Symbol	Description
P_{ij}	Stochastic processing time of O_{ij}
S_{ij}	Stochastic start time of operation O_{ij}
C_{ij}	Stochastic end time of operation O_{ij}, $C_{ij} = S_{ij} + P_{ij}$
C_{max}	Expected makespan

Table 2.3.: Additional stochastic notation

this variability, where each repetition is a simulation. This Monte-Carlo-based procedure is accurate for high simulation numbers but also computationally intensive, since for every iteration the graph has to be traversed. A more detailed examination of result sampling is presented in Section 4.2.2.

3. Literature Review

In this section we will explore existing methods that authors have used to address this $\mathcal{NP}$-*hard* problem. In Section 3.1 we will look at exact approaches and metaheuristics used to solve the deterministic problem, after which we will look at the less-researched stochastic problem in Section 3.2. Subsequently, we will explore work on the *dynamic makespan* method as an alternative to Monte Carlo simulations in Section 3.3. We conclude our review of the literature in Section 3.4.

3.1. Job-Shop Scheduling

Algorithms that can find good schedules for the Job-Shop Scheduling Problem (JSSP) have been investigated for quite some time. Since Fisher and Thompson (1963) introduced their famous JSSP problem instance with $n = 10$ and $m = 10$, there has been a 25-year competition to solve this optimally, using mainly branch-and-bound methods (Carlier and Pinson, 1989). More recently, methods such as heuristic rules and shifting bottleneck procedures have taken favor over exact computation methods, as their approximation nature allows for efficient searching. The shifting bottleneck procedure is an iterative scheduling algorithm that sequentially identifies the machine causing the most delay (the bottleneck) and optimally schedules it to improve the overall production schedule. Today, other metaheuristic approximation algorithms, such as Simulated Annealing (SA), Tabu Search (TS), or Genetic Algorithms (GA), are the preferred method of choice (Boukedroun et al., 2023).

Van Laarhoven et al. (1992) implemented Simulated Annealing for JSSP and proved some fundamental properties about the neighborhood structure and convergence of the algorithm. They showed, for example, that reversing a critical arc in a DAG can never result in a cycle. The convergence of the annealing process was also described here using Markov chains. A Markov chain describes a sequence of possible events in which the probability of each event depends on the previous state. The performance of SA was compared to the shifting bottleneck heuristics and showed significant improvements. Simulated Annealing was also implemented by Yamada and Nakano (1996), enhancing it with a shifted bottleneck procedure, experimentally showing that the proposed method found near-optimal schedules for difficult benchmarks.

Zhang et al. (2008), more recently, implemented a Hybrid Simulated Annealing with Tabu Search resulting in high-quality solutions in very short computation times, establishing 17 new upper bounds among the then unsolved problems in a short time. Tabu Search is a metaheuristic optimization technique that guides local search methods to escape local optima by keeping track of recently visited solutions, called the "tabu

list," to avoid cycling back to them. It uses a memory structure to record and prevent moves that would undo recent changes, allowing the search to explore new regions of the solution space. With methods such as Tabu Search, Taillard (1994) stated that their method is typically more efficient than shifting bottleneck procedures and outperforms a recent Simulated Annealing approach. Moreover, Nowicki and Smutnicki (2005) showed promising results with Tabu Search based on the big valley phenomenon, including new theoretical properties of neighborhoods, showing unprecedented performance and quality.

Recently, Neural Networks are explored as an optimization technique for JSSP. Neural networks, such as Graph Neural Networks (GNNs) that learn structural representations of job-shop graphs, have been investigated to predict effective moves or to approximate optimal schedules. A survey by Smit et al. (2024) examines how GNNs, particularly in combination with deep reinforcement learning (DRL), are applied to various job-shop scheduling problems (JSSPs). GNNs offer a flexible approach by representing scheduling tasks as graph structures, allowing algorithms to capture complex relationships among tasks and resources.

For a comprehensive review of recent advances and methodologies in deterministic JSSP, including developments in hybrid algorithms and emerging artificial intelligence applications, we refer to Boukedroun et al. (2023).

3.2. Stochastic Job-Shop Scheduling

For the Stochastic Job-Shop Scheduling Problem (SJSSP), different methods have been used to deal with both the large search space and the added uncertainty. Ideally, optimal schedules are to be found within reasonable time, and since we established that for the deterministic variant this is already difficult, it is no surprise that many implementations in the literature use local search in one form or another, as exact methods are practically infeasible. These metaheuristics are often combined with other techniques to further speed up the search or decrease the search space such as *Optimal Cost Budget Allocation* (OCBA), which is a statistical technique concerning the evaluation of the candidate solution, which intelligently allocates the computational budget among candidate solutions by assigning more simulation runs to those with a higher likelihood of being optimal (Yang et al., 2014). Another technique is *Indifference Zone* (IZ), where the difference in quality of two solutions is significant if it is within a given parameter range; otherwise, the decider is indifferent about the new solution (Lee et al., 2010). In this section, we will look at how different metaheuristic approaches are applied in the context of job-shop.

Evolutionary Optimization

Evolutionary Algorithms are widely applied to tackle SJSSP due to their effectiveness in exploring large and complex search spaces. These algorithms are inspired by natural evolution, simulating mechanics such as selection crossover and mutation to evolve a pop-

ulation of candidates. For example, in the work by Horng et al. (2012), an *Evolutionary Strategy Ordinal Optimization* (ESOO) method is used to minimize the expected sum of storage expenses and tardiness penalties in SJSSP with stochastic processing times. In their approach, each candidate solution represents a possible schedule of jobs on machines. The algorithm begins with an initial population of randomly generated schedules. During the selection phase, schedules are evaluated based on their performance under uncertainty and the best performers are selected as parents. The crossover operator then combines pairs of parent schedules to produce offspring by exchanging sequences of operations, thereby inheriting features from both parents that may contribute to a better schedule. Mutation introduces random changes to offspring schedules, i.e. by altering the machine orderings, which helps maintain diversity in the population and prevents premature convergence to local optima. Ordinal optimization is a technique used in optimization to efficiently identify good enough solutions rather than the absolute best. By focusing on selecting a high-ranking subset of options based on order rather than precise numerical values, it reduces computation time and resources needed to reach the desired results.

Yang et al. (2014) integrates (OCBA) into the ESOO algorithm to further enhance performance. OCBA improves the efficiency of simulation-based optimization by allocating more simulations to the most promising candidates. Their approach aimed to minimize the expected sum of earliness and tardiness and showed improved convergence rates and solution quality compared to strictly ESOO.

More recently, Horng and Lin (2022) proposed an *Artificial Immune System* integrated with *Ordinal Optimization* (AISOO), combined with OCBA. The Artificial Immune System is supported by a rough estimate to determine a selected subset, after which OCBA is utilized to look for near-optimal schedules. This was tested on medium-sized (6x6) and large-sized (10x10) instances with a truncated normal, exponential, and uniform probability distribution.

Yoshitomi (2002) designed a GA algorithm to approach SJSSP, where job processing times are uncertain and follow probabilistic distributions. The GA is implemented to handle variability by evolving schedules that can adapt to these uncertainties, with the aim of minimizing job completion time and improving schedule robustness. Experimental results demonstrate that the GA approach outperforms traditional methods, offering more reliable schedules under stochastic conditions.

Lei (2011) extended the use of Genetic Algorithms and incorporated machine breakdowns using exponential processing times and non-presumable jobs, this work highlighted the importance of considering additional stochastic factors into the search procedure to generate more robust schedules.

In more recent work, Boukedroun et al. (2023) proposed a hybrid approach combining Tabu Search with Genetic Algorithms. They study perturbations that affect the processing times of jobs and show that a high value of the number of critical operations is linked to a high variation of the makespan of perturbed schedules, making critical operations not only good targets for optimizing but also a clue of the goodness of a schedule considering stochastic and robustness measures. Additionally, they conducted

an extensive literature review on both the deterministic and stochastic job-shop domain.

These Evolutionary Optimization techniques demonstrate the effectiveness of combining metaheuristic algorithms with methods such as ordinal optimization, OCBA, and genetic programming to tackle the complexities of SJSSP. By addressing uncertainties in processing times and incorporating mechanisms to focus on computational efforts, these methods offer promising solutions to the challenges inherent in stochastic scheduling problems.

Ant colony / particle swarm optimization

Other metaheuristics have also been explored for the SJSSP, such as *Ant Colony Optimization* (ACO) and *Particle Swarm Optimization* (PSO). These algorithms are inspired by natural processes and have been adapted to address the uncertainties in scheduling problems. ACO simulates the foraging behavior of ants by having artificial ants construct solutions incrementally, guided by pheromone trails that represent the collective learning of the colony; in SJSSP, ants probabilistically select the next operation to schedule based on pheromone intensity and heuristic information. PSO, on the other hand, mimics the social behavior of swarms like bird flocks or fish schools, where each particle represents a potential schedule and adjusts its position in the solution space based on its own experience and the best experiences of neighboring particles; in SJSSP, particles share information to find robust schedules that perform well under uncertainty.

Horng and Lin (2015) introduced an *Ant Colony System integrated with Ordinal Optimization* (ACSOO). This method utilizes the multidirectional search capabilities of ACO and the goal-softening approach of ordinal optimization to efficiently search for high-quality schedules. By formulating the SJSSP as a constrained stochastic simulation problem, ACSOO aims to find acceptable solutions within reasonable computation times. In a subsequent study, Horng and Lin (2020) enhanced this approach by incorporating OCBA into the exploitation phase of the two-stage ACSOO algorithm, further improving its performance and efficiency.

Zhang et al. (2012) proposed a two-stage PSO method for the SJSSP. The first stage involves a performance estimate to guide the search, while the second stage employs Monte Carlo simulations combined with OCBA to refine the solutions. This approach takes advantage of the global search capabilities of PSO and the efficiency of OCBA to handle the stochastic nature of the problem. This is extended by Araki and Yoshitomi (2016), combining it with an uncertain environment, using Monte Carlo simulations to address the stochastic programming aspects. Their method enhanced the PSO algorithm by integrating stochastic evaluations directly into the particle update mechanisms. By incorporating Monte Carlo simulations, they effectively estimated the expected performance of schedules under uncertainty, allowing particles to adjust their positions based on more accurate assessments of solution quality. This method outperformed the prior implementation in terms of solution quality and computational efficiency, demonstrating the potential of PSO in stochastic scheduling contexts.

Simulated Annealing

Simulated Annealing (SA) was introduced in around 1983 as a new approach to calculate approximate solutions of combinatorial optimization problems. Simulated Annealing is a metaheuristic technique inspired by the annealing process in metallurgy and finds its approach from the Metropolis method, which calculates equilibrium states in systems composed of interacting molecules. (Steinhöfel et al., 1999).

Tavakkoli-Moghaddam et al. (2005) proposed a nonlinear mathematical programming model for SJSSP, where a SA approach was used to improve the quality and performance of an initial solution generated by a neural network. This hybrid approach allowed for a more robust exploration of the solution space and improved the chances of finding high-quality schedules.

van Blokland (2012) performed an extensive study on job-shop and blocking job-shop scheduling using Simulated Annealing as local search procedure to determine an efficient objective function, named *result sampling* which is based on Monte Carlo simulations. Van Blokland implemented a variation of Simulated Annealing where a different cool-down metric is used. They argued that including the quality difference of the two states in the probability of acceptance, the behavior of accepting solutions will be unstable. They opted to using solely the temperature as probability metric in accepting or rejecting neighboring states. We will use the result sampling as the baseline in this thesis to compare the method discussed in the next section. van Blokland (2012) also introduced a Waiting left shift neighborhood operator for JSSP, where the aim is to reduce the waiting time of the operation that causes the largest waiting time. This neighborhood operator is not guaranteed to produce an acyclic disjunctive graph and has to be followed by a feasibility check. They also incorporate *cutoff sampling* to better guide the local search by removing schedules, comparing them with the lowest and highest makespan, removing potentially erratic schedules from the search. This research was formalized and published by van den Akker et al. (2013).

Our own Simulated Annealing framework and its implementation for (stochastic) job-shop will be further elaborated on in Section 4.2.

3.3. Dynamic Makespan

Evaluating the objective function in a stochastic scheduling problem requires extensive simulation, as we have seen. This quickly becomes a computational burden if you want to accurately simulate by sampling the distributions, or when the problem instances increase in size making the graph traversal slower. Passage (2016) addressed this computational challenge in his thesis on *Stochastic Parallel Machine Scheduling* by proposing methods to approximate the distribution of C_{max} using normal approximation techniques. The key idea is to iteratively estimate the expected value and variances of the maximum of two normally distributed random variables. This is then done iteratively to find the distribution for all starting times of the operations, assuming that the resulting distribution from the maximization is also normally distributed. By adding

the processing duration, which is also normally distributed, to this derived start time we can calculate the completion time for every vertex in the graph.

This approximation method, coined *dynamic makespan*, significantly reduces computational effort compared to traditional simulation approaches. Instead of performing numerous simulations to estimate the distribution of c_{max}, the normal approximation provides arithmetic expressions to estimate the expected makespan and its variability. Passage demonstrated that this method yields better results than using 1000 independent simulations per iteration and consistently outperforms using 300 simulations per iteration, all while requiring a fraction of the computational time. Furthermore, Passage provided methods for approximating a normal distribution from other types of distribution, such as the uniform distribution, the Erlang distribution, and the exponential distribution. The results of their research are the product of a local search algorithm *Variable Neighborhood Descent* (VND). This method dynamically switches between neighborhoods when local minima are encountered, increasing the chances of finding a global optimum.

The foundation of this technique traces back to the work of Clark (1961), who introduced the use of normal distributions to approximate the distribution of the completion time in PERT networks. PERT (Program Evaluation and Review Technique) is a project management tool used to plan, schedule, and control complex tasks by representing them as a network of activities and events. It incorporates uncertainty in activity durations by using probabilistic time estimates, allowing for the calculation of expected durations and variances to estimate the project's overall completion time. Clark's method involves calculating the mean and variance of activity durations and then using these parameters to approximate the distribution of the project's completion time, assuming the activity durations are independent and normally distributed.

Pascual (2022) extended the work of Passage and performed a robustness study for Stochastic Parallel Machine Scheduling. Here, 18 robustness measures were devised, two of which were based on normal approximation methods, to quantify this robustness of a schedule. These measures were categorized as quality robustness and solutions robustness. The slack-based measures performed well together with the aforementioned normal approximation. Pascual used the best performing measures in an *Iterated Local Search* algorithm with as subroutine *Variable Neighborhood Descent*, proving the effectiveness of the measures in a practical setting. This is especially significant given the fact that parallel machine scheduling and job-shop scheduling have overlap in their problem definition. The results found that the normal approximation quality robustness measures performed among the best where weighted free slack measures performed overall the worst.

3.4. Conclusion

In this chapter, we reviewed various approaches to tackle the Job-Shop Scheduling Problem (JSSP) and its stochastic variant (SJSSP), highlighting the challenges of both exact and heuristic methods due to the combinatorial nature and computational complexity

of these problems. For the deterministic JSSP, we observed a progression from early exact approaches, such as branch-and-bound, to more flexible metaheuristic techniques, including Simulated Annealing, Tabu Search, and Genetic Algorithms. These methods have proven efficient in finding near-optimal solutions, balancing solution quality and computational time.

In the stochastic domain, methods like Evolutionary Algorithms, Ant Colony Optimization, and Particle Swarm Optimization show promise in addressing uncertainty, especially when paired with techniques like Optimal Cost Budget Allocation and Ordinal Optimization to refine solution quality efficiently. We found that traditional Monte Carlo simulations for stochastic variable calculation is difficult and explored heuristics employed for Parallel Machine Scheduling by Passage (2016).

In summary, while metaheuristic algorithms effectively address the complexities of stochastic scheduling, there are still improvements to be made in terms of computational efficiency. We found that *dynamic makespan* was an effective method to calculate the objective function in Stochastic Parallel Machine Scheduling. We will use this method for the Stochastic Job-Shop Scheduling Problem to improve existing methods. This adoption and integration with Simulated Annealing is further described in Section 4.2.2.

4. Methodology

This chapter will detail the metaheuristic method adopted to tackle the complex nature of stochastic job-shop scheduling. First, we will explore the method used by Passage (2016) to approximate the expected makespan using normal distributions in Section 4.1. After which the search procedure, Simulated Annealing, will be described with its various parts in Section 4.2.

4.1. Dynamic Makespan Approximation

Working with stochastic variables requires us to do more work in approximating the quality of the schedule. As we have seen, traditional methods of simulation (result sampling) have proved computationally intensive as they require multiple traversals through the graph. Having to repeat this procedure for every iteration of the local search algorithm makes the entire search process cumbersome. To solve this, Passage (2016) proposed an approximation procedure, called *dynamic makespan*. Here, the aim is to calculate the starting times of all the vertices using a dynamic program. The start time of a vertex is calculated by taking the maximum of the completion times of all its predecessors maintaining the precedence relations. This maximization is over variables that are normally distributed, and thus an approximation is done to determine the distribution of the result. Passage (2016) has implemented this method for Parallel Machine Scheduling but is also applicable to job-shop scheduling as we will see.

The assumption that all the Probability Density Functions (PDFs) of the processing times are normally distributed and not correlated allows for an aggregation of the approximated PDFs of all predecessors of an operation using the central limit theorem. Specifically, for any operation O_{ij}, we can approximate its completion time C_{ij} as the maximum of both its machine predecessor and its job predecessor, which can be determined using the topological sorting, plus the processing time, Equation (4.1). If the operation is either the first operation of the job or the first operation on the machine, its predecessor will be the source dummy vertex, whose distribution will default to $\mathcal{N}(0,0)$ as the base case of the recursion.

$$C_{ij} := \max\{C_{(i-1)j}, C_{m_{\prec}(O_{ij})}\} + P_{ij} \tag{4.1}$$

4.1.1. Maximization of Normal Distributions

Suppose that we have two Normal distributions, $\mathcal{N}_1, \mathcal{N}_2$. We now want to calculate the distribution X of the maximum of these two distributions, $X = \max\{\mathcal{N}_1, \mathcal{N}_2\}$. We

assume here that the random variable X is also normally distributed such that it can be reused in further calculations. This allows us to define a method to recursively define the start and completion times of the operation using successive maximization of the normal distribution. We can define what the maximization of two normal distributions looks like in Equation (4.2).

$$X \sim \max\{\mathcal{N}(\mu_1, \sigma_1^2), \mathcal{N}(\mu_2, \sigma_2^2)\} \tag{4.2}$$

Nadarajah and Kotz (2008) propose a method to find the distribution of a min/max of two normally distributed random variables. We use this method to compute $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$, with which we can infer σ_X^2. Let $\theta = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2}$, also let $\Phi(x)$ and $\phi(x)$ be the PDF and *cumulative distribution function* (CDF), respectively. We assume that the correlation coefficient ρ is known and $\rho = 0$, defining the distributions as uncorrelated. We find the calculation for μ_X in Equation (4.3), the variance σ_X^2 is calculated in Equation (4.5).

$$\mathbb{E}[X] = \mu_1\Phi(\frac{\mu_1 - \mu_2}{\theta}) + \mu_2\Phi(\frac{\mu_2 - \mu_1}{\theta}) + \theta\phi(\frac{\mu_1 - \mu_2}{\theta}) \tag{4.3}$$

$$\begin{aligned}\mathbb{E}[X^2] = {} & (\sigma_1^2 + \mu_1^2)\Phi(\frac{\mu_1 - \mu_2}{\theta}) \\ & + (\sigma_2^2 + \mu_2^2)\Phi(\frac{\mu_2 - \mu_1}{\theta}) \\ & + (\mu_1 + \mu_2)\theta\phi(\frac{\mu_1 - \mu_2}{\theta})\end{aligned} \tag{4.4}$$

$$\sigma_X^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{4.5}$$

Illustrative Example

Consider the schedule in Table 2.2. Suppose that the processing times were to be normally distributed with a stochastic factor of $\alpha = 0.25$, such that $P_{ij} \sim \mathcal{N}(p_{ij}, 0.25 \cdot p_{ij})$, this updated problem definition is in Table 4.1 with tuples $(m_{ij}, \mu_{ij}, \sigma_{ij}^2)$.

	O_{0j}	O_{1j}	O_{2j}
O_{i0}	$(0, 4, 1.00)$	$(1, 1, 0.25)$	$(2, 3, 0.75)$
O_{i1}	$(0, 3, 0.75)$	$(1, 2, 0.50)$	$(2, 5, 1.25)$
O_{i2}	$(2, 3, 0.75)$	$(0, 4, 1.00)$	$(1, 1, 0.25)$

Table 4.1.: Stochastic problem instance with tuples $(m_{ij}, \mu_{ij}, \sigma_{ij}^2)$

Given the following machine ordering: $M_0 = \{O_{00}, O_{01}, O_{12}\}$, $M_1 = \{O_{10}, O_{11}, O_{22}\}$, $M_2 = \{O_{02}, O_{20}, O_{21}\}$ (top schedule of Figure 2.1b), we can calculate the start time and

completion time of O_{20} as follows:

$$\begin{aligned} C_{10} &= \max\{C_{00}, 0\} + \mathcal{N}(1, 0.25) \\ &= C_{00} + \mathcal{N}(1, 0.25) \\ &= \mathcal{N}(5, 1.25) \end{aligned}$$

$$\begin{aligned} C_{00} &= \max\{0, 0\} + \mathcal{N}(4, 1.00) \\ &= \mathcal{N}(4, 1.00) \end{aligned}$$

$$\begin{aligned} C_{02} &= \max\{0, 0\} + \mathcal{N}(3, 0.75) \\ &= \mathcal{N}(3, 0.75) \end{aligned} \tag{4.6}$$

$$\begin{aligned} C_{20} &= \max\{C_{10}, C_{02}\} + \mathcal{N}(3, 0.75) \\ &= \max\{\mathcal{N}(5, 1.25), \mathcal{N}(3, 0.75)\} + \mathcal{N}(3, 0.75) \\ &= \textit{Using Equation } (4.7) \\ &= \mathcal{N}(5.050, 1.104) + \mathcal{N}(3, 0.75) \\ &= \mathcal{N}(8.050, 1.854) \end{aligned}$$

We expand the calculation of the maximum of the two normal distributions in Equation (4.7). We find that the approximated normal distribution for O_{20} is $P_{20} \sim \mathcal{N}(8.050, 1.854)$.

$$\begin{aligned} \theta &= \sqrt{\sigma_1^2 + \sigma_2^2 - 0} \approx \sqrt{1.25 + 0.75} \approx 1.414 \\ \frac{\mu_1 - \mu_2}{\theta} &= \frac{5 - 3}{\sqrt{2}} \approx 1.414 \\ \mathbb{E}[O_{20}] &= 5 \cdot \Phi(1.414) + 3 \cdot \Phi(-1.414) + 1.414 \cdot \phi(1.414) \\ &\approx 5 \cdot 0.9214 + 3 \cdot 0.0786 + 1.414 \cdot 0.1468 \\ &\approx 5.050 \\ \mathbb{E}[O_{20}^2] &= (1.25 + 5^2)\Phi(1.414) + (0.75 + 3^2)\Phi(-1.414) \\ &\quad + (5 + 3)\theta\phi(1.414) \\ &\approx 26.25 \cdot 0.9214 + 9.75 \cdot 0.0786 + 8 \cdot 1.414 \cdot 0.1468 \\ &\approx 26.614 \\ \sigma_{O_{20}}^2 &= \mathbb{E}[O_{20}^2] - \mathbb{E}[O_{20}]^2 \\ &\approx 26.614 - 5.0507^2 \\ &\approx 1.104 \end{aligned} \tag{4.7}$$

When this calculation is performed for V_{21} (final operation), the procedure approximated the following normal distribution: $C_{21} = \mathcal{N}(14.469, 2.667)$. When we compare this to 1 million Monte Carlo simulations, we find a mean of 14.241 and a variance of 3.099 from the samples. We are approximately at 1.5% of the anticipated value and around 15% of

the variance for this particular case. This approximated normal distribution compared to the probability density function of the Monte Carlo simulations is visualized in Figure 4.1.

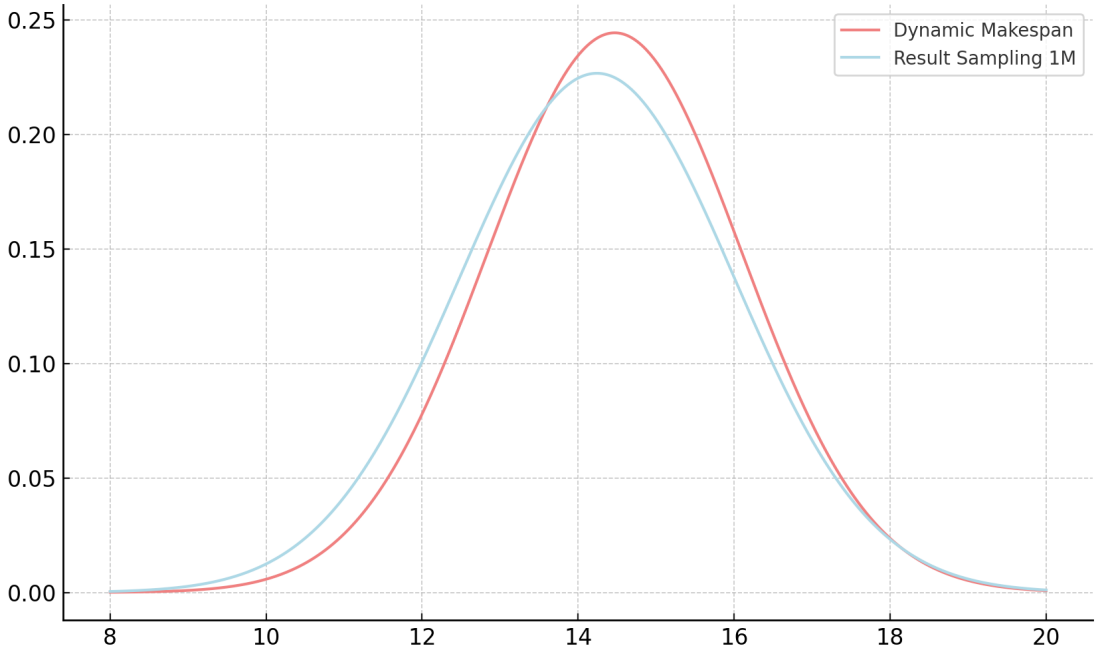


Figure 4.1.: Probability Density Functions for Dynamic Makespan and the Result Sampling simulations

This quick example hints at the advantage that this approximation method has over traditional simulation. The arithmetical approach to normal approximations allow for very quick computation times, approximating C_{max}, as needed for the local search procedure. In the next Section, we will explore how this *dynamic makespan* is incorporated as an objective function into the Iterated Local Search / Simulated Annealing framework.

4.2. Simulated Annealing

Simulated Annealing (SA) is a powerful metaheuristic that can effectively explore large and complex search spaces, making it an ideal candidate for the job-shop scheduling problem. Simulated Annealing is classified as a local search algorithm, specifically a type of improvement algorithm. These algorithms start with a random initial solution and progressively enhance it, as opposed to constructive algorithms.

Local search algorithms usually try to find a state (e.g. a schedule) that is better than the current state in its *neighborhood*. Two states are neighbors if a state can be reached by a neighborhood operator $N(G)$, changing a feature of the state in such a way it impacts the quality of the state (e.g. the makespan) in a good or bad way.

The quality of a given state is calculated using the objective function, defined by $f(G)$. A common local search algorithm is the Hill Climber algorithm, which simply accepts any neighboring state that has an increased objective function (when maximizing). This, however, often results in the algorithm converging into a local optimum in the optimality landscape, and variations of Hill Climbing therefore allow the algorithm to escape this local optimum.

Simulated Annealing does this by the Metropolis criterion, which introduces a controlled acceptance of worse solutions based on the Temperature of the algorithm. In this way, neighboring states with a lower objective function (i.e., representing a worse schedule) may still be accepted with probability P, defining the Metropolis criterion in Equation (4.8). Here, Δf is the difference in the objective function and T defines the temperature. When T decreases, the chance of accepting delta also decreases, approaching 0 for small values of T.

$$P = e^{-\Delta f/T} \tag{4.8}$$

The algorithm subsequently needs to decide whether or not this neighboring state, with either a higher or lower objective function, is to be accepted as the new current state. Together with the Metropolis criterion we define the acceptance function Γ as follows, with r being an uniformly distributed random variable between $[0, 1]$:

$$\Gamma(\Delta f, T) = \begin{cases} 1 & \text{if } \Delta f < 0 \vee r < e^{-\Delta f/T}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.9}$$

The temperature of the system is controlled by a cooling schedule, and in some cases also by a heating schedule. The cooling schedule determines how quickly the temperature T decreases over time, in turn affecting the likelihood of accepting worsening neighboring states. Different cooling methods exist, cooling with a factor α being the most common. This cool down factor is subsequently applied after Q iterations to allow the algorithm to explore more options with the current temperature.

$$T_{k+1} = \alpha T_k \tag{4.10}$$

In some variants, a heating schedule may also be used temporarily to escape challenging local optima by allowing the algorithm to "reheat" and increase T under certain conditions. This enables SA to revisit parts of the search space it may have bypassed due to premature convergence at a lower temperature. We find the general structure for Simulated Annealing in Algorithm 1.

4.2.1. Neighborhood Operators

To navigate the search space of the problem, we need to be able to transform the current state to another state that meets the requirements of the constraints. There can be many different neighborhoods based on the type of neighborhood operator. These neighborhood operators will strictly create neighboring states that respect the *active schedule* definition from Section 2.1.

Algorithm 1 General structure for simulated annealing

input: T_0, T_{min} = initial temperature and minimal temperature respectively
        Q = iterations per T
        α = cooling parameter α
        G = initial state
$G^* \leftarrow G$ ▷ Initialize best state to G
$T \leftarrow T_0$ ▷ Initialize temperature
while $T > T_{min}$ **do**
    for $Q_i \in [0, 1, ..., Q]$ **do**
        $n \leftarrow N(G)$ ▷ select neighborhood operator
        $G' \leftarrow n(G)$ ▷ perform neighborhood operator
        $\Delta f \leftarrow f(G) - f(G')$
        if $\Delta f < 0 \vee r < e^{\Delta f/T}$ **then**
            if $f(G') < f(G*)$ **then**
                $G* \leftarrow G'$
            end if
        else
            undo $n(G)$, *restore* G
        end if
    end for
    $T \leftarrow T * \alpha$
end while
return G^*

We have seen that a schedule can be seen as a graph with disjunctive and conjunctive edges between the vertices (operations). Finding some orientation for the disjunctive arcs directly translates to finding an ordering of operations on the machines. The neighborhood operators for job-shop find their application here on this set of disjunctive arcs. A transition to a neighboring state is generated by selecting two vertices v and w such that v and w are successive operations on the same machine and $(v, w) \in A$ is an arc on the critical path in G. We restrict ourselves to the swapping of arcs of vertices on the critical path in the deterministic case, as motivated by Theorem 4.2.1 (Van Laarhoven et al., 1992). Furthermore, Van Laarhoven et al. showed that the reversal of disjunctive arcs on the longest path will always lead to feasible solutions.

Theorem 4.2.1. *Given a directed acyclic graph $G = (V, E)$ and a critical path $C \subseteq E$, a decrease in c_{max} can only be achieved by swapping $(O_i, O_{i+1}) \in C$, where $m_{i+1} = m_i \wedge J_{i+1} \neq J_i : \forall O_{i+1}, O_i \in C$. In other words, the length of the critical path can only be decreased by swapping vertices that are a subset of this critical path.*

Proof. Proof by contradiction. Suppose the arc $(O_i, O_{i+1}), \notin C$, with length x (i.e. processing time). There exists some other path of length y from O_{i+1} to O_i. Furthermore, there is some length a, which denotes the sum of the weights from V_s to O_{i+1} and length b, the sum of the weights from O_i to V_e. We define the makespan as such, $c_{max} = a+y+b$. If we were to reverse the direction of (O_i, O_{i+1}) to (O_{i+1}, O_i), we can distinguish between two cases:

Case 1: $x > y$, then $c'_{max} = a + x + b > a + y + b = c_{max}$. The makespan increases since $x > y$ because the longest path from V_s to V_e now includes x.

Case 2: $x \leq y$, then $c'_{max} = a + x + b \leq a + y + b = c_{max}$. But since $(O_i, O_{i+1}) \notin C$ there exists some other path from O_{i+1} to O_i with a higher weight that includes y.

Therefore the longest path constitutes the makespan c_{max} and will remain to do so unless a critical arc is reversed or an optimal ordering is already formed. This is only the case in the deterministic setting since no single longest path exists in the graph for stochastic processing durations. □

Swap Operator

This allows us to formulate the neighborhood operators, given the deterministic setting, $N^S(G)$. This operator finds all eligible operators that can be swapped according to the constraints by traversing the critical path and selecting adjacent pairs of operations that are being processed on the same machine and are not part of the same job, we define this neighborhood in Equation (4.11).

$$N^S(G) = \{(V_i, V_j) \in C \mid m_i = m_j \wedge J_i \neq J_j\} \tag{4.11}$$

The size of this neighborhood is dependent on the number of operations to be processed on a machine, m_k. We follow the definition of Van Laarhoven et al. (1992) to define the size of this neighborhood in terms of N and m_k.

$$|N^S(G)| < \sum_{k=1}^{m} (m_k - 1) \leq N - m \tag{4.12}$$

Since we only have to iterate the critical path once to calculate $N^S(G)$, and the critical path is at most N, the time complexity for this neighborhood operator is $\mathcal{O}(N)$.

BlockSwap Operator

Nowicki and Smutnicki (1996) showed that $N^S(G)$ can be limited to only the pairs of operations at the beginning and end of each critical block. Large neighborhoods can slow down the search algorithm since it spends more time looking at neighbors that will not necessarily move the algorithm towards an optimum. Decreasing the size of the neighborhood can help the algorithm converge faster, at the cost that it potentially gets stuck in a local optimum. The *blockswap-operator* can be seen as a subset of the swap-operator: $N^B(G) \subseteq N^S(G)$. Specifically, having critical blocks $[B_1, ..., B_r] \in C$, the first pair and last pair of every block $B_2, ..., B_{r-1}$ is swapped, along with the last pair of B_1 and first pair of B_r. We then select a random candidate neighbor from this reduced set of neighbors. The size of the neighborhood is the sum of these three parts, determined in Equation (4.13) according to the article of Nowicki and Smutnicki (1996).

$$\begin{aligned} |N^B(G)| = &\min\{1, |B_1| - 1\} \\ &+ \sum_{k=2}^{r-1} \min\{2, |B_k| - 1\} \\ &+ \min\{1, |B_r| - 1\} \end{aligned} \tag{4.13}$$

We assume here that there is a single critical path; if there exist multiple critical paths in G then $N^S(G)$ will increase faster in size compared to $N^B(G)$. The computational complexity of $N^B(G)$ remains $\mathcal{O}(N)$, since we can calculate candidate pairs for each block in a scan over the critical path.

Stochastic Neighborhood

With stochastic variables, the notion of a critical path, and therefore the neighborhoods, changes slightly. The critical path now depends on the samples taken from the probability distributions. Consequently, for an identical topological sorting but varying distribution samples, different critical paths can exist. To address this, we distinguish between the two computational methods employed. For the *dynamic makespan* method, we simply use the expected values of the distributions and thus compute a deterministic critical path that is independent of the sample variations.

With *result sampling*, we execute the deterministic critical path operation multiple times, sampling all processing time distributions during each iteration. For each of these iterations, a neighborhood is calculated from the critical path found, aggregating the results in a larger neighborhood. The advantage of this is that if a (part of a) critical path occurs often in the different simulations, its potential swaps from the neighborhood will have more presence in this aggregated neighborhood. If we in turn randomly pick a random element from this aggregate neighborhood, the chances of picking a swap occurring in multiple critical paths increase.

4.2.2. Computational Methods Makespan

As we have seen, objective functions are essential in steering the local search algorithm in the direction of the optimum. Since the objective function for job-shop means we need to traverse the graph, determine the start and ending times for every operation to find the makespan, we soon have a computationally intensive subroutine on our hands. Especially with large problem instances that create large directed graphs. In this Section we will look at the general structure of the objective functions, beginning with the most basic case where we look at the deterministic makespan. Building on top of this we introduce the *result sampling* objective function and the variation that incorporates the earlier defined *dynamic makespan*.

Deterministic

Finding the makespan in the deterministic case is nothing more than finding the longest weighted path from V_s to V_e on the graph where the weights represent the processing time p_{ij}. Using a topological sorting τ of graph G we can calculate the makespan in linear time $\mathcal{O}(N)$, in terms of vertices and edges as shown by Adams et al. (1988). Topological sorting sorts the vertices of the graph in a list while maintaining the precedence constraints set by the arcs in the graph. Finding such a sorting can be done in $\mathcal{O}(N)$ by iterating over the vertices and its edges recursively until all the vertices are visited, using for example Kahn's algorithm (Kahn, 1962).

As we have seen in Chapter 2, we can make use of the fact that, using a machine ordering, an operation has exactly two predecessors: a job predecessor and a machine predecessor. Using this, we use a dynamic program to recursively walk back on these machine and job predecessors until we reach the source. If we maintain which vertices have already been computed using memoization, we can compute the makespan of the graph in $\mathcal{O}(N)$ (Van Laarhoven et al., 1992). In addition, we maintain which predecessor was chosen of a given vertex to reconstruct the critical path, also in $\mathcal{O}(N)$.

Result Sampling

With *result sampling*, a realization is drawn from each distribution P_{ij}. These realizations can be used to calculate the makespan of the schedule in a deterministic fashion. However, if you were to sample all these distributions and calculate the makespan, it might be erratic compared to another sampling of all the distributions. The main idea of result sampling is to perform this L times to average out this randomness. When L approaches infinity, the mean of the makespan will approach the actual expected value for this random variable, but this is in practice not feasible to calculate. Instead, moderate values for L are chosen in search of a balance between computational speed and accuracy.

A direct result from sampling the distributions L times is that there will also be L critical paths in the graph. To this end, we simply store for each critical path the set of potential neighbors. Using the aggregated set of these neighbors, we select a neighbor

randomly, implicitly increasing the chance of a neighbor to be chosen randomly if the pair occurred frequently in different critical paths.

Van Blokland (2012) and van den Akker et al. (2013) also incorporate methods like *cutoff sampling*, to mitigate these instances that have large outliers to the expected makespan to stabilize the local search. Here, the L different results are sorted by makespan and a top and bottom percentage is disregarded in the acceptance function of the local search procedure.

The complexity of result sampling then becomes $\mathcal{O}(L \cdot N)$, since we perform a procedure very similar to the deterministic objective function L times. This method, albeit computationally intensive, gives a good representation of the actual probability distribution of the makespan. We will use result sampling as a baseline measure to compare the *dynamic makespan* method to assess its quality and computational performance.

Dynamic Makespan

As we have seen in Section 4.1, we have a method to approximate the distribution of the makespan. This allows us to assess the quality of the schedule in $\mathcal{O}(N)$ since the steps required closely resemble that of the deterministic objective function, where instead now we apply the normal distribution approximation method discussed.

It is this approximated normal distribution for the sink vertex that is used in the objective function comparison Δf, for this we simply take the expected values of the distributions: $\Delta f_{DM} = \mu_2 - \mu_1$. Where $f(G) \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $f(G') \sim \mathcal{N}(\mu_2, \sigma_2^2)$. This method traverses the graph once and calculates the start-time distributions for all the different operations. The expected values μ of the two preceding distributions are subsequently used to establish which of the two operations to chose as predecessor on the critical path.

4.3. Iterated Local Search

What we will utilize for our experiments is *Iterated Local Search* (ILS). This is a modification on a local search procedure that aims to escape local optima when no improving neighbors are available. This is done by perturbing the state to some degree and subsequently starting another local search procedure on that perturbed state. We employ ILS with Simulated Annealing as the subroutine. Although SA has an exploration phase when the temperature is high, it might still converge to a local optimum. To alleviate this, we "reheat" the temperature to a value smaller than the original temperature and perturb the solution by performing a random walk for some given number of iterations. During a random walk, we apply the neighborhood operator and accept whatever improving or worsening state returns. This procedure is described in Algorithm 2.

Algorithm 2 General structure for iterated local search

input: r = rounds
        G = initial state
        z = Random Walk iterations per round
$G_i \leftarrow G^* \leftarrow G$
for $r_i \in [0, 1, ..., r]$ **do**
    $G' \leftarrow SimulatedAnnealing(G_i, ...)$
    if $f(G') < f(G^*)$ **then**
        $G^* \leftarrow G'$
    end if
    $G_i \leftarrow RandomWalk(G^*, z)$
end for
return G^*

4.4. Conclusion

This chapter outlined the methodology adopted to address the stochastic job-shop scheduling problem using metaheuristic approaches. First, we explored the dynamic makespan approximation method used in the work of Passage (2016) for Parallel Machine Scheduling. This leverages normal distribution properties to estimate the makespan efficiently. This technique offers a computational advantage over traditional result sampling, which requires extensive Monte Carlo simulations for an accurate representation of the expected makespan. We then describe the simulated Annealing approach, detailing the neighborhood operators and objective functions used in the search process. Finally, we introduced the Iterated Local Search framework, combining Simulated Annealing with perturbation techniques to escape local optima.

5. Experiments and Results

5.1. Experimental Setup

To conduct the experiments and perform the analysis consistently, we define the experimental setup in this section. To this end, we introduce stochasticity in existing deterministic instances from the literature. Using these stochastic instances, we will test three different types of objective functions: Deterministic (D), Dynamic Makespan (DM) and Result Sampling (RSn), where n denotes the number of simulations. The two proposed neighborhood operators Swap and BlockSwap are selected with equal probability within each iteration of the search process.

The experiments will be conducted three times to average the stochasticity of the local search and the average values will be reported. After the search procedure has finished, the final schedule will be simulated using one million Result Sampling simulations. We report per instance the mean μ and standard deviation σ, along with the 50^{th}, 70^{th} and 90^{th} percentile to assess the distribution of these simulations. Finally, the processing time of the Iterated Local Search algorithm will be reported on in minutes.

The experiments are run on an Intel® Core™ i7-13700H, performing at most five experiments concurrently.

5.1.1. Problem Instances

In the literature, there are well-known JSSP problem instances, with numerous attempts made to solve these instances optimally and efficiently. Some larger instances do not have a known optimum, being bounded below and above by specific values. It is crucial for the algorithm to perform adequately in various problem sizes. Different well-known instances were used in this research and are presented in Table 5.1. A subset of these instances are used to determine the parameters and are marked with $*$. We will group these instances into three different sizes: small ≤ 100, medium $(100, 225)$, and large ≥ 225.

5.1.2. Initial Schedule

From these problem instances, initial schedules must be constructed before the search algorithms can iteratively improve upon this. There have been studies to show the impact this initial solution has on the final solution of the search procedure. In some cases, starting from a totally random solution, compared to a greedy initial solution, can improve the final solution.

Problem	Jobs	Machines	N	Optimum
Fisher and Thompson (1963)				
ft06	6	6	36	55
ft10	10	10	100	930
ft20*	5	20	100	1165
Adams et al. (1988)				
abz05*	10	10	100	1234
abz06	10	10	100	943
abz07	20	15	300	656
abz08	20	15	300	$645 - 665$
abz09	20	15	300	$661 - 679$
Applegate and Cook (1991)				
orb01	10	10	100	1059
orb02	10	10	100	888
orb03	10	10	100	1005
orb04	10	10	100	1005
orb05	10	10	100	887
orb06	10	10	100	1010
orb07	10	10	100	397
orb08	10	10	100	899
orb09	10	10	100	934
orb10	10	10	100	944
Taillard (1993)				
ta01*	15	15	225	1231
ta02*	15	15	225	1244
ta03	15	15	225	1218
ta11*	20	15	300	$1323 - 1361$
ta12*	20	15	300	$1351 - 1367$
ta13	20	15	300	$1282 - 1342$

Table 5.1.: Different problem instances from literature used in this research, together with their known deterministic optimum, or lower and upper bounds.
$*$ instances used for determining search paramters.

Due to the high temperature in the early phases of Simulated Annealing the algorithm explores different states regardless of their quality, potentially moving away from a "good" initial state before converging back to some optimum. We have experimented with starting the Simulated Annealing algorithm with an instance that was optimized with a greedy Hill Climber, but this did not have significant effect on the final quality. For the

experiments, we generated an initial solution from the problem instances by sequentially directing the undirected machine arcs until all machine arcs are directed. By performing this operation sequentially, no cycles can be generated. This approach results in a feasible solution; however, it may significantly deviate from the known deterministic optimum.

5.1.3. Stochasticity

To introduce randomness in the problem instances, we introduce variance into the processing duration. We have seen in Equation 2.9 how a factor α creates the variances as a factor of p_{ij}.

- $\mathcal{N}_{25} = P_{ij} \sim \mathcal{N}(\mu = p_{ij}, \sigma^2 = 0.25 \cdot p_{ij})$
- $\mathcal{N}_{50} = P_{ij} \sim \mathcal{N}(\mu = p_{ij}, \sigma^2 = 0.50 \cdot p_{ij})$

Parameters

The parameters were demonstrated to be most effective for problem instances of varying sizes. To substantiate the selection of these parameters, preliminary experiments were conducted on a subset of problem instances: *ft20, abz05, ta01, ta02, ta11, ta12*. Each experiment was conducted with five repetitions, and the averages of the results are reported. Figure 5.1 summarizes these findings for the various parameters.

Temperature:
The temperature is directly related to the average delta Δf of the problem instances. Initially, we found a temperature of 40 to perform best for the different problem instances. This value was found by looking at the average delta for the different instances. In practice, the algorithm performs better with a temperature parameter specific to that instance. For this, we sample the temperature by running a SA procedure for 15.000 iterations and recording this average delta. Using this *dynamic temperature* and the intended maximum number of iterations Q_{max} we determine Q using $Q = \frac{Q_{max} \cdot \ln \alpha}{\ln (T_{min}/T_0)}$.

Using test instances, we found that using the *dynamic temperature* the average makespan decreased from 1535.81 to 1530.45, improving approximately 0.37%. We will use this temperature method for every instance when initializing an experiment.

Iterations:
Using the dynamic temperature, we continue to determine the optimal value for Q_{max}, which directly influences the value for Q. In Figure 5.1a, we observe that the average makespan quickly decreases with increasing number of iterations, but appears to converge on average after 125.000 iterations. We set $Q_{max} = 125.000$ in our experiments as a good trade-off in quality compared to computation times.

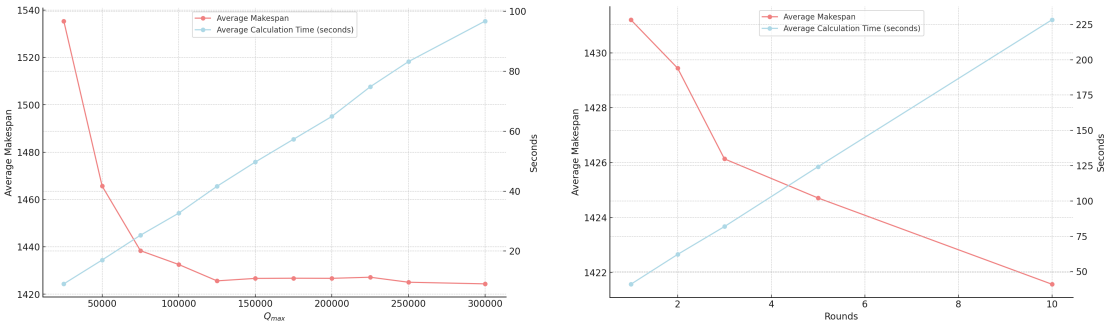
ILS rounds:
We found that, in general, improvement among instances significantly decreases after 3 rounds, while the computation time increases linearly with each additional round. The average makespan with increasing ILS rounds is shown in Figure 5.1b.

Simulated Annealing:

- Initial temperature: $T_0 =$ dynamic based on first 15.000 SA iterations
- Minimum temperature: $T_{min} = 1$
- Cooling factor: $\alpha = 0.95$
- Nested Iterations: $Q = \frac{Q_{max} \cdot \ln \alpha}{\ln (T_{min} \ / \ T_0)}$
- Maximum Iterations: $Q_{max} = 125.000$

Iterated Local Search:

- Rounds: $r = 3$
- Random Walk iterations: $z = 5$
- Simulated Annealing reheating:
  - $T_0 = 5$
  - $Q_{max} = 95.000$



(a) Average makespan for different values of Q_{max}, using a dynamic temperature compared against calculation time in seconds.

(b) Average makespan for different values of r, using a dynamic temperature compared against calculation time in seconds.

Figure 5.1.: Preliminary results for the different parameters

5.2. Results

The experimental results for different instances, objective functions, and values for α are described in the tables in Appendix A. The summary of these results are found in Table 5.2, where the average percentual increase over all instances is reported together with the average time T in minutes. We will compare our findings with the deterministic objective function and use these values as a baseline. We separate these findings according to the size of the instance in Table 5.3.

5.2.1. Deterministic Performance

The results clearly show that the deterministic objective function performs well on all problem instances. The deterministic objective function did not take the variance of the processing durations into account but still performed relatively well compared to the computational methods that did inlucde this variability. This allowed the deterministic method to compute fastest compared to the other methods, where the quality of the solutions is only around 2 percent from the average. The average makespan found is comparable to somewhere between RS05 and RS50. The fact that the deterministic method performs so well is possibly due to the relatively small number of variations in the problem. Even for $\alpha = 0.50$, the deviations from the probability distributions remain fairly small. Given the fact that the average processing duration for all instances used in the experiments is 46.59, we would on average have an average standard deviation of 3.41 and 4.83 for $\alpha = .025$ and $\alpha = 0.50$, respectively, which is only around 7.3% and 10.4% of this average mean.

5.2.2. Dynamic Makespan Performance

The dynamic makespan method shows a slight increase in computation time, needed to perform the maximization calculations on the normal distributions. We find that the solutions found by DM are less than 1% from the optimal on average. There is also a significant decrease in this percentual average between $\alpha = 0.25$ and $\alpha = 0.50$. The increased uncertainty, in the form of larger standard deviations on the normal distributions, results in a higher quality schedule when compared to other the other methods. Dynamic makespan performs somewhere between RS05 and RS50 for $\alpha = 0.25$ and is superior to RS100 with $\alpha = 0.50$ in all but instance types except for small instances. DM outperforms the deterministic method and is significantly faster than the result sampling methods. Table 5.3 gives us insight in how the method performs on different input sizes. DM outperforms RS100 in both medium and large problem instance sizes.

5.2.3. Result Sampling Performance

Result sampling is clearly more computationally intensive. The increase in the computation time increases linearly with the number of simulations and matches the time

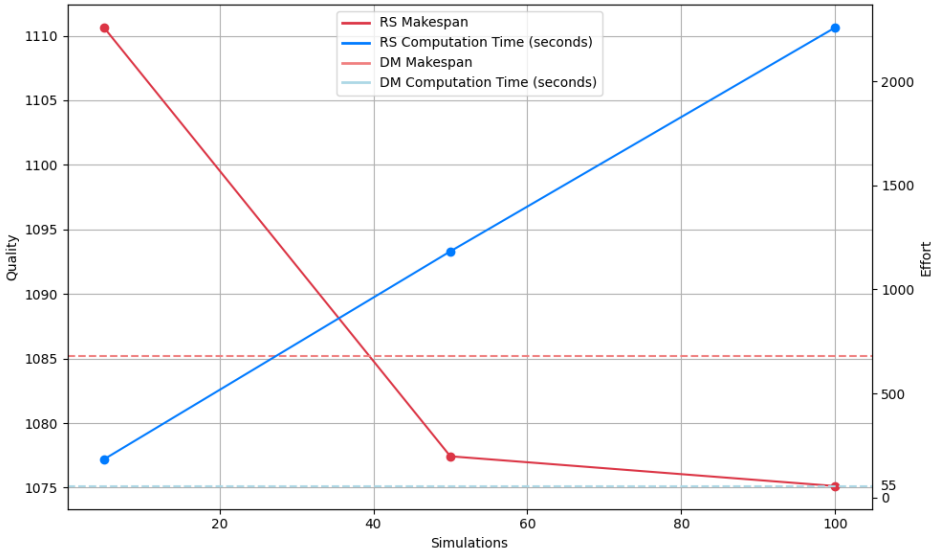
complexity $\mathcal{O}(L \cdot N)$. As expected, the quality of the solutions found increases when more simulations are performed per iteration. RS100 outperforms all other methods for $\alpha = 0.25$, but takes on average 50 times longer compared to the deterministic objective. This trade-off in computation time and simulations is visualized in Figure 5.2. The dynamic makespan average makespan and computation time is displayed as horizontal lines, while the result sampling average makespan and computation time is visualized for different simulation numbers. We can see that for a larger alpha value, DM outperforms RS100, while for the smaller alpha value, DM seems to perform comparable with RS40.

	α	P_{50} %	P_{70} %	P_{90} %	T
D	$\mathcal{N}_{25}$	2.38	2.26	2.10	47.70
	$\mathcal{N}_{50}$	2.61	2.48	2.27	39.20
	avg.	2.49	2.49	2.18	43.45
DM	$\mathcal{N}_{25}$	1.04	1.00	0.96	53.09
	$\mathcal{N}_{50}$	0.55	0.51	0.46	47.90
	avg.	0.80	0.80	0.71	50.49
RS05	$\mathcal{N}_{25}$	3.22	3.23	3.25	158.00
	$\mathcal{N}_{50}$	6.24	6.36	6.55	147.79
	avg.	4.73	4.73	4.90	152.89
RS50	$\mathcal{N}_{25}$	0.64	0.65	0.68	1208.56
	$\mathcal{N}_{50}$	1.78	1.82	2.59	1093.76
	avg.	1.21	1.21	1.64	1151.16
RS100	$\mathcal{N}_{25}$	0.23	0.25	0.29	2331.93
	$\mathcal{N}_{50}$	0.87	0.92	0.99	2093.31
	avg.	0.55	0.55	0.64	2212.62

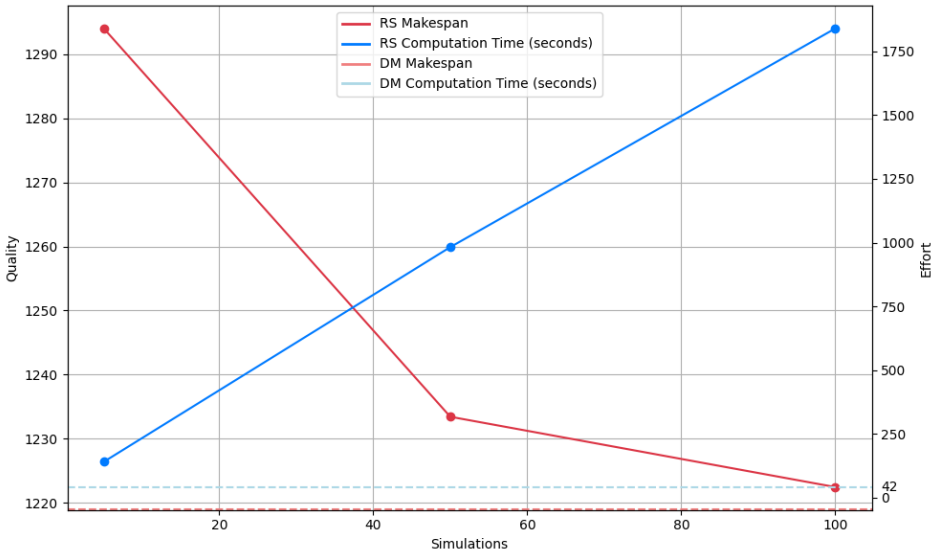
Table 5.2.: Average increase in percentiles and time in seconds from the tables in Appendix A.

		Small				Medium				Large			
	a	P_{50} %	P_{70} %	P_{90} %	T	P_{50} %	P_{70} %	P_{90} %	T	P_{50} %	P_{70} %	P_{90} %	T
D	$\mathcal{N}_{25}$	2.70	2.56	2.37	29.85	2.11	1.98	1.78	51.61	1.91	1.84	1.74	90.07
	$\mathcal{N}_{50}$	2.95	2.79	2.54	23.92	1.46	1.39	1.25	50.98	2.47	2.38	2.22	70.67
	avg.	2.83	2.83	2.45	26.88	1.78	1.78	1.51	51.29	2.19	2.19	1.98	80.37
DM	$\mathcal{N}_{25}$	1.13	1.10	1.07	37.14	1.97	1.89	1.76	65.13	0.55	0.51	0.48	87.12
	$\mathcal{N}_{50}$	0.94	0.87	0.79	30.45	-0.00	0.00	0.00	66.15	0.00	0.00	0.00	81.76
	avg.	1.04	1.04	0.93	33.80	0.98	0.98	0.88	65.64	0.27	0.27	0.24	84.44
RS05	$\mathcal{N}_{25}$	2.27	2.26	2.25	103.72	4.35	4.39	4.46	195.27	4.97	5.00	5.06	273.90
	$\mathcal{N}_{50}$	5.00	5.10	5.25	89.63	7.30	7.48	7.75	199.50	8.81	8.97	9.19	265.78
	avg.	3.63	3.63	3.75	96.67	5.83	5.83	6.10	197.39	6.89	6.89	7.13	269.84
RS50	$\mathcal{N}_{25}$	0.47	0.47	0.49	769.86	0.67	0.67	0.68	1542.17	1.09	1.13	1.21	2133.75
	$\mathcal{N}_{50}$	0.88	0.90	2.12	667.17	1.24	1.36	1.53	1500.96	3.65	3.73	3.83	1943.13
	avg.	0.67	0.67	1.30	718.52	0.96	0.96	1.10	1521.56	2.37	2.37	2.52	2038.44
RS100	$\mathcal{N}_{25}$	0.30	0.32	0.35	1508.81	0.00	0.00	0.00	2987.59	0.15	0.17	0.23	4065.19
	$\mathcal{N}_{50}$	0.13	0.14	0.18	1283.50	0.69	0.84	1.09	3043.60	2.68	2.74	2.82	3622.69
	avg.	0.21	0.21	0.26	1396.16	0.35	0.35	0.54	3015.59	1.41	1.41	1.52	3843.94

Table 5.3.: Average increase from percentiles found and processing time split on instance size. Percentual increases and computation times are averaged from the tables in Appendix 4.5.



(a) Average expected makespan for different simulation numbers with $\alpha = 0.25$.



(b) Average expected makespan for different simulation numbers with $\alpha = 0.50$.

Figure 5.2.

6. Conclusion

6.1. Summary

In this thesis, we explored the Stochastic Job-Shop Scheduling Problem where the processing times are modeled as stochastic variables. We aimed to investigate whether an approximation heuristic could significantly decrease computation times and increase the quality. We compare this to traditional Monte Carlo simulation, which requires multiple graph traversals per local search iteration and becomes computationally difficult with large simulation numbers.

To this end we extended the dynamic makespan method proposed by Passage (2016) for parallel machine scheduling. With dynamic makespan, we approximated the normal distribution of each operation as a maximization of both its machine and its job predecessor. Using this, the distribution of the sink vertex can be calculated and thus the makespan. We performed preliminary tests to determine the search parameters. Using a dynamic temperature for Simulated Annealing based on the average delta of a problem instance. We also determined which number of rounds for Iterated Local Search would best meet the decreased makespan and increased computation time considerations. Two different neighborhood operators were proposed, and their time and size complexity were analyzed. The search procedure was performed on well-known job-shop instances from literature, modified to include the uncertainty in processing durations.

6.2. Conclusion

This goal of this research was to implement a local search heuristic that could adequately improve existing local search methods and deal with a stochastic environment. We have found that the dynamic makespan method is computationally comparable to the deterministic makespan method, while being significantly more effective, outperforming all other objectives when there is much deviation in the distributions. The additional arithmetic operations to approximate the normal distributions have little effect on the computation time, showing only a slight increase. Including the variability of the underlying distributions resulted in a lower average makespan. Computationally DM performed faster than the result sampling methods, which showed a linear correlation in the computation time compared to the number of simulations.

We have concluded from the results that the amount of uncertainty introduced could have been higher to further distance the performance of DM from other methods, this by applying the stochastic factor to the standard deviation, for example (Pascual, 2022).

We have successfully shown the potential of normal approximation heuristics for local

search algorithms in the context of stochastic job-shop scheduling. Including such a heuristic will significantly decrease computation times compared to traditional sampling methods.

6.3. Future Research

Predecessor selection with DM

Within Simulated Annealing, the predecessor for a given operation is chosen depending on what objective function is selected. These predecessors are then used to reconstruct the critical path after the graph has been traversed from the sink to the source. For the deterministic objective function, we simply look at which of the two completion times of the predecessors is higher and store that operation. For Result Sampling we perform the same technique multiple times with different samples from the distributions. For Dynamic Makespan we look at the expected values of the normal distributions, which neglect the information given by the variance. Including this variance in the predecessor decision process can potentially increase the robustness of the final schedule, and it would be interesting to explore methods to include this using methods like the KL-divergence, for example, as a measure of how different a distribution Q is from another distribution P.

Result Sampling optimization

Result Sampling is a computationally intensive method to calculate the quality of a given schedule. When simulating a neighboring state within the Simulated Annealing procedure, we can determine the Δ value needed in advance to accept the state with the Metropolis criteria. Given a neighboring state G', we could already get a sense of $f(G')$ and what value it will converge to with smaller iteration numbers. If this number significantly deviates from the required value needed by the Metropolis criteria, we can cancel the simulation entirely and mark the neighbor as not accepted (de Bruin et al., 2023). This will especially speed up Result Sampling with higher simulation counts, i.e. RS100.

Partial graph traversal

After the application of a neighborhood operator, the start and end times of operations are recalculated by traversing the graph or linked-list data structure, this is because the change in machine order can have effect on the start times of other operations. However, it can never have an effect on the operations that precede these changed operations in the topological sorting. Since the graph is recalculated for every iteration of the search algorithm and potentially many different times for Result Sampling per iteration, it can be very effective to recalculate parts of the graph instead of the entire graph. Especially if the graph has a large number of vertices, this pruning step can significantly increase performance.

Stochastic sources

Currently we only work with processing durations that are normally distributed in the model. Passage (2016) also proposes methods to approximate uniform, exponential, and Erlang distributions, and it would be interesting to examine the effects of these distributions on the performance of the dynamic makespan objective function. It may additionally be interesting to measure the robustness of a schedule given different types of uncertainty such as machine breakdown, delaying all the operations for that machine.

Bibliography

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401.

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156.

Araki, K. and Yoshitomi, Y. (2016). Stochastic job-shop scheduling: A hybrid approach combining pseudo particle swarm optimization and the monte carlo method. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 10(3).

Boukedroun, M., Duvivier, D., Ait-el Cadi, A., Poirriez, V., and Abbas, M. (2023). A hybrid genetic algorithm for stochastic job-shop scheduling problems. *RAIRO - Operations Research*, 57(4):1617–1645.

Broek, van den, J. (2009). *MIP-based approaches for complex planning problems*. Phd thesis, Technische Universiteit Eindhoven.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Clark, C. E. (1961). The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162.

Conway, R., Maxwell, W., and Miller, L. (1967). *Theory of Scheduling*. Addison-Wesley Publishing Company.

de Bruin, P., van den Akker, M., Hoogeveen, H., and van Kooten Niekerk, M. (2023). Scheduling Electric Buses with Stochastic Driving Times. In Frigioni, D. and Schiewe, P., editors, *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023)*, volume 115 of *Open Access Series in Informatics (OASIcs)*, pages 14:1–14:19, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Fisher, H. and Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling*, page 225–251.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.*, 1(2):117–129.

Graham, R., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1977). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326.

Horng, S.-C. and Lin, S.-S. (2015). Integrating ant colony system and ordinal optimization for solving stochastic job shop scheduling problem. In *2015 6th international conference on intelligent systems, modelling and simulation*, pages 70–75. IEEE.

Horng, S.-C. and Lin, S.-S. (2020). Two-stage bio-inspired optimization algorithm for stochastic job shop scheduling problem. *Int. J. Simul. Syst. Sci. Technol.*

Horng, S.-C. and Lin, S.-S. (2022). Apply ordinal optimization to optimize the job-shop scheduling under uncertain processing times. *Arab. J. Sci. Eng.*, 47(8):9659–9671.

Horng, S.-C., Lin, S.-S., and Yang, F.-Y. (2012). Evolutionary algorithm for stochastic job shop scheduling with random processing time. *Expert Syst. Appl.*, 39(3):3603–3610.

Kahn, A. B. (1962). Topological sorting of large networks. *Communications of The ACM*, 5(11):558–562.

Lee, L. H., Chen, C.-h., Chew, E. P., Li, J., Pujowidianto, N. A., and Zhang, S. (2010). A review of optimal computing budget allocation algorithms for simulation optimization problem. *International Journal of Operations Research*, 7(2):19–31.

Lei, D. (2011). Scheduling stochastic job shop subject to random breakdown to minimize makespan. *The International Journal of Advanced Manufacturing Technology*, 55(9–12):1183–1192.

Nadarajah, S. and Kotz, S. (2008). Exact distribution of the max/min of two gaussian random variables. *IEEE Transactions on very large scale integration (VLSI) systems*, 16(2):210–212.

Nowicki, E. and Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management science*, 42(6):797–813.

Nowicki, E. and Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159.

Pascual, L. (2022). *Improving Robustness For Stochastic Parallel Machine Scheduling With Robustness Measures*. Master's thesis, Utrecht University.

Passage, G. (2016). *Combining local search and heuristics for solving robust parallel machine scheduling*. Master's thesis, Utrecht University.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition.

Roy, B. and Sussmann, B. (1964). Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds*, 9.

Smit, I., Zhou, J., Reijnen, R., Wu, Y., Chen, J., Zhang, C., Bukhsh, Z., Nuijten, W., and Zhang, Y. (2024). Graph neural networks for job shop scheduling problems: A survey. Workingpaper, arXiv.org.

Steinhöfel, K., Albrecht, A., and Wong, C.-K. (1999). Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research*, 118(3):524–548.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6(2):108–117.

Tavakkoli-Moghaddam, R., Jolai, F., Vaziri, F., Ahmed, P. K., and Azaron, A. (2005). A hybrid method for solving stochastic job shop scheduling problems. *Appl. Math. Comput.*, 170(1):185–206.

van Blokland, K. (2012). Solution approaches for solving stochastic job shop and blocking job shop problems. Master's thesis, Utrecht University.

van den Akker, M., van Blokland, K., and Hoogeveen, H. (2013). Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search. In *Experimental Algorithms*, Lecture notes in computer science, pages 402–413. Springer Berlin Heidelberg, Berlin, Heidelberg.

Van Laarhoven, P. J., Aarts, E. H., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations research*, 40(1):113–125.

Yamada, T. and Nakano, R. (1996). Job-shop scheduling by simulated annealing combined with deterministic local search. *Meta-heuristics: Theory and applications*, pages 237–248.

Yang, H.-A., Lv, Y., Xia, C., Sun, S., and Wang, H. (2014). Optimal computing budget allocation for ordinal optimization in solving stochastic job shop scheduling problems. *Math. Probl. Eng.*, 2014:1–10.

Yoshitomi, Y. (2002). A genetic algorithm approach to solving stochastic job-shop scheduling problems. *International Transactions in Operational Research*, 9(4):479–495.

Zhang, C. Y., Li, P., Rao, Y., and Guan, Z. (2008). A very fast ts/sa algorithm for the job shop scheduling problem. *Computers & operations research*, 35(1):282–294.

Zhang, R., Song, S., and Wu, C. (2012). A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem. *Knowledge-Based Systems*, 27:393–406.

Appendices

A. Tabulated Experimental Results For All Problem Instances

The following tables show the experimental results of the Iterated Local Search algorithm. These are the result of one million Result Sampling simulations performed on the most optimal schedule found by the search algorithm. The experiment was repeated five times. We report 50^{th}, 70^{th} and 90^{th} percentiles of the results in Table A.1, A.2 and A.3, together with the computation times in seconds in Table A.4.

		D			DM			RS05			RS50			RS100		
	α	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}
abz5	$\mathcal{N}_{25}$	1352.54	1382.10	1426.60	1340.34	1371.46	1418.33	1354.34	1386.76	1435.72	1338.92	1370.86	1419.16	**1333.58**	**1365.82**	**1414.50**
		+1.42%	+1.19%	+0.85%	+0.51%	+0.41%	+0.27%	+1.56%	+1.53%	+1.50%	+0.40%	+0.37%	+0.33%	–	–	–
	$\mathcal{N}_{50}$	1508.10	1564.22	1648.61	1502.39	1559.76	1646.39	1518.48	1577.60	1666.96	1498.54	1556.93	1645.06	**1484.34**	**1542.28**	**1630.24**
		+1.60%	+1.42%	+1.13%	+1.22%	+1.13%	+0.99%	+2.30%	+2.29%	+2.25%	+0.96%	+0.95%	+0.91%	–	–	–
abz6	$\mathcal{N}_{25}$	1062.86	1086.91	1123.01	1031.18	1056.99	1095.99	1040.94	1066.83	1105.67	1024.04	1050.03	1089.19	**1022.88**	**1049.09**	**1088.61**
		+3.91%	+3.60%	+3.16%	+0.81%	+0.75%	+0.68%	+1.77%	+1.69%	+1.57%	+0.11%	+0.09%	+0.05%	–	–	–
	$\mathcal{N}_{50}$	1201.36	1248.14	1318.18	1155.96	1204.06	1276.48	1185.04	1235.12	1310.91	1137.95	1186.51	1260.12	**1136.94**	**1185.74**	**1259.76**
		+5.67%	+5.26%	+4.64%	+1.67%	+1.54%	+1.33%	+4.23%	+4.16%	+4.06%	+0.09%	+0.07%	+0.03%	–	–	–
abz7	$\mathcal{N}_{25}$	757.59	768.95	786.19	**745.93**	**757.18**	**774.38**	780.40	793.07	812.49	756.82	768.83	787.28	749.08	760.83	778.91
		+1.56%	+1.55%	+1.53%	–	–	–	+4.62%	+4.74%	+4.92%	+1.46%	+1.54%	+1.67%	+0.42%	+0.48%	+0.59%
	$\mathcal{N}_{50}$	860.98	882.37	914.88	**842.84**	**864.71**	**898.42**	921.52	947.01	985.98	872.24	894.80	929.39	862.09	884.95	920.13
		+2.15%	+2.04%	+1.83%	–	–	–	+9.34%	+9.52%	+9.75%	+3.49%	+3.48%	+3.45%	+2.28%	+2.34%	+2.42%
abz8	$\mathcal{N}_{25}$	766.37	777.78	795.06	**763.25**	**774.92**	**792.73**	805.78	818.41	837.70	776.85	789.70	809.49	766.87	779.06	797.60
		+0.41%	+0.37%	+0.29%	–	–	–	+5.57%	+5.61%	+5.67%	+1.78%	+1.91%	+2.11%	+0.47%	+0.53%	+0.61%
	$\mathcal{N}_{50}$	873.61	895.12	927.73	**868.90**	**891.28**	**925.36**	935.35	960.72	999.32	902.69	926.68	963.28	892.85	916.97	953.89
		+0.54%	+0.43%	+0.26%	–	–	–	+7.65%	+7.79%	+7.99%	+3.89%	+3.97%	+4.10%	+2.76%	+2.88%	+3.08%
abz9	$\mathcal{N}_{25}$	801.16	813.28	831.63	791.36	803.91	822.98	815.83	828.79	848.56	790.11	803.39	823.66	**782.89**	**795.56**	**814.85**
		+2.33%	+2.23%	+2.06%	+1.08%	+1.05%	+1.00%	+4.21%	+4.18%	+4.14%	+0.92%	+0.98%	+1.08%	–	–	–
	$\mathcal{N}_{50}$	908.53	930.98	965.24	**871.16**	**893.76**	**928.24**	972.28	999.82	1041.84	917.01	941.62	979.14	902.32	926.03	962.20
		+4.29%	+4.17%	+3.99%	–	–	–	+11.61%	+11.87%	+12.24%	+5.26%	+5.36%	+5.48%	+3.58%	+3.61%	+3.66%
ft06	$\mathcal{N}_{25}$	59.32	61.30	64.26	59.01	61.01	64.05	59.75	61.79	64.85	59.13	61.12	64.10	**58.91**	**60.92**	**63.92**
		+0.69%	+0.62%	+0.52%	+0.16%	+0.15%	+0.19%	+1.42%	+1.43%	+1.45%	+0.37%	+0.33%	+0.28%	–	–	–
	$\mathcal{N}_{50}$	65.77	69.63	75.40	65.42	69.35	75.24	66.39	70.48	76.63	65.59	69.56	75.54	**65.31**	**69.18**	**75.01**
		+0.71%	+0.65%	+0.53%	+0.17%	+0.24%	+0.31%	+1.65%	+1.88%	+2.17%	+0.44%	+0.55%	+0.72%	–	–	–
ft10	$\mathcal{N}_{25}$	1077.94	1101.72	1137.44	1061.65	1085.43	1121.50	1050.44	1074.06	1109.70	1032.88	1056.94	1093.14	**1026.74**	**1050.64**	**1086.60**
		+4.99%	+4.86%	+4.68%	+3.40%	+3.31%	+3.21%	+2.31%	+2.23%	+2.13%	+0.60%	+0.60%	+0.60%	–	–	–
	$\mathcal{N}_{50}$	1211.32	1256.81	1325.20	1166.84	1210.87	1277.65	1221.00	1268.75	1340.45	1175.48	1222.30	1293.66	**1157.88**	**1203.85**	**1273.49**
		+4.62%	+4.40%	+4.06%	+0.77%	+0.58%	+0.33%	+5.45%	+5.39%	+5.26%	+1.52%	+1.53%	+1.58%	–	–	–
ft20	$\mathcal{N}_{25}$	1317.88	1344.15	1383.45	1251.14	1278.55	1320.77	1286.85	1313.79	1354.69	1255.51	1281.68	1321.69	**1246.65**	**1272.75**	**1312.84**
		+5.71%	+5.61%	+5.38%	+0.36%	+0.46%	+0.60%	+3.23%	+3.22%	+3.19%	+0.71%	+0.70%	+0.67%	–	–	–
	$\mathcal{N}_{50}$	1464.09	1515.05	1591.32	1393.00	1441.97	1516.00	1465.14	1520.62	1604.88	1401.18	1452.80	1531.36	**1386.33**	**1437.51**	**1515.51**
		+5.61%	+5.39%	+5.00%	+0.48%	+0.31%	+0.03%	+5.68%	+5.78%	+5.90%	+1.07%	+1.06%	+1.05%	–	–	–

Table A.1.: Computational results for different instances, objective functions and alpha values. Reporting the 50^{th}, 70^{th} and 90^{th} percentiles.

		D			DM			RS05			RS50			RS100		
	α	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}
orb01	$\mathcal{N}_{25}$	1196.17	1223.63	1264.61	1216.45	1245.15	1287.97	1198.29	1226.06	1267.56	**1178.18**	**1205.82**	**1247.19**	1181.76	1209.78	1251.44
		+1.53%	+1.48%	+1.40%	+3.25%	+3.26%	+3.27%	+1.71%	+1.68%	+1.63%	–	–	–	+0.30%	+0.33%	+0.34%
	$\mathcal{N}_{50}$	1339.04	1391.65	1470.00	1338.30	1389.29	1465.97	1411.81	1470.17	1558.24	1324.56	1377.73	1457.38	**1313.55**	**1366.36**	**1445.16**
		+1.94%	+1.85%	+1.72%	+1.88%	+1.68%	+1.44%	+7.48%	+7.60%	+7.82%	+0.84%	+0.83%	+0.85%	–	–	–
orb02	$\mathcal{N}_{25}$	983.62	1006.71	1041.66	**964.66**	**988.35**	**1025.05**	991.96	1016.50	1054.00	971.79	995.74	1032.35	974.79	998.77	1035.10
		+1.97%	+1.86%	+1.62%	–	–	–	+2.83%	+2.85%	+2.82%	+0.74%	+0.75%	+0.71%	+1.05%	+1.05%	+0.98%
	$\mathcal{N}_{50}$	1111.74	1156.03	1222.96	**1087.11**	**1129.66**	**1194.79**	1153.95	1204.62	1282.49	1092.38	1135.82	1202.02	1094.07	1136.57	1201.50
		+2.27%	+2.33%	+2.36%	–	–	–	+6.15%	+6.64%	+7.34%	+0.48%	+0.55%	+0.61%	+0.64%	+0.61%	+0.56%
orb03	$\mathcal{N}_{25}$	1128.11	1152.83	1189.76	**1117.63**	**1141.35**	**1177.02**	1156.86	1183.20	1222.73	1130.57	1156.55	1195.60	1148.11	1174.92	1215.19
		+0.94%	+1.01%	+1.08%	–	–	–	+3.51%	+3.67%	+3.88%	+1.16%	+1.33%	+1.58%	+2.73%	+2.94%	+3.24%
	$\mathcal{N}_{50}$	1268.86	1316.74	1388.17	1275.77	1326.62	1403.56	1350.78	1403.77	1483.68	1271.92	1319.85	1392.43	**1257.43**	**1305.06**	**1376.77**
		+0.91%	+0.89%	+0.83%	+1.46%	+1.65%	+1.95%	+7.42%	+7.56%	+7.77%	+1.15%	+1.13%	+1.14%	–	–	–
orb04	$\mathcal{N}_{25}$	1119.11	1144.07	1181.51	1100.64	1127.21	**1167.45**	1122.58	1150.22	1191.67	1103.41	1130.38	1171.04	**1098.65**	**1126.80**	1169.18
		+1.86%	+1.53%	+1.20%	+0.18%	+0.04%	–	+2.18%	+2.08%	+2.07%	+0.43%	+0.32%	+0.31%	–	–	+0.15%
	$\mathcal{N}_{50}$	1250.83	1298.86	1371.00	1231.47	1280.62	1354.50	1307.47	1362.54	1445.73	**1222.33**	**1273.04**	**1349.15**	1222.43	1273.69	1350.76
		+2.33%	+2.03%	+1.62%	+0.75%	+0.60%	+0.40%	+6.96%	+7.03%	+7.16%	–	–	–	+0.01%	+0.05%	+0.12%
orb05	$\mathcal{N}_{25}$	982.28	1004.58	1038.15	974.08	**996.58**	**1030.46**	986.89	1009.69	1044.16	976.76	1000.14	1035.39	**973.62**	996.64	1031.27
		+0.89%	+0.80%	+0.75%	+0.05%	–	–	+1.36%	+1.32%	+1.33%	+0.32%	+0.36%	+0.48%	–	+0.01%	+0.08%
	$\mathcal{N}_{50}$	1104.12	1146.53	1210.47	1096.86	1139.28	**1203.56**	1144.42	1190.67	1260.60	1126.18	1171.91	1440.46	**1094.44**	**1138.98**	1206.45
		+0.88%	+0.66%	+0.57%	+0.22%	+0.03%	–	+4.57%	+4.54%	+4.74%	+2.90%	+2.89%	+19.68%	–	–	+0.24%
orb06	$\mathcal{N}_{25}$	1149.97	1174.56	1211.88	1129.24	1154.02	1191.39	1129.63	1155.11	1193.55	**1102.24**	**1127.22**	**1165.17**	1102.94	1128.12	1166.41
		+4.33%	+4.20%	+4.01%	+2.45%	+2.38%	+2.25%	+2.49%	+2.47%	+2.44%	–	–	–	+0.06%	+0.08%	+0.11%
	$\mathcal{N}_{50}$	1294.29	1341.39	1412.45	1247.38	1292.63	1361.46	1292.15	1341.38	1416.00	1242.83	1290.10	1361.75	**1237.11**	**1283.05**	**1352.65**
		+4.62%	+4.55%	+4.42%	+0.83%	+0.75%	+0.65%	+4.45%	+4.55%	+4.68%	+0.46%	+0.55%	+0.67%	–	–	–
orb07	$\mathcal{N}_{25}$	441.73	451.65	466.78	**435.39**	**445.30**	**460.51**	446.58	456.98	472.76	436.36	446.42	461.80	437.38	447.64	463.32
		+1.46%	+1.43%	+1.36%	–	–	–	+2.57%	+2.62%	+2.66%	+0.22%	+0.25%	+0.28%	+0.46%	+0.53%	+0.61%
	$\mathcal{N}_{50}$	499.46	518.29	547.10	**489.15**	**507.92**	**536.48**	513.64	533.64	564.43	511.88	531.78	562.74	494.39	513.70	543.17
		+2.11%	+2.04%	+1.98%	–	–	–	+5.01%	+5.06%	+5.21%	+4.65%	+4.70%	+4.90%	+1.07%	+1.14%	+1.25%
orb08	$\mathcal{N}_{25}$	1052.79	1076.58	1112.33	1018.12	1041.55	1077.79	1027.14	1051.91	1089.60	1003.11	1027.75	1065.15	**996.24**	**1020.38**	**1057.12**
		+5.68%	+5.51%	+5.22%	+2.20%	+2.07%	+1.95%	+3.10%	+3.09%	+3.07%	+0.69%	+0.72%	+0.76%	–	–	–
	$\mathcal{N}_{50}$	1182.33	1227.00	1294.37	1153.94	1197.82	1264.43	1172.93	1220.31	1292.71	**1133.04**	**1177.41**	**1244.78**	1140.15	1185.04	1253.32
		+4.35%	+4.21%	+3.98%	+1.84%	+1.73%	+1.58%	+3.52%	+3.64%	+3.85%	–	–	–	+0.63%	+0.65%	+0.69%

Table A.2.: Computational results for different instances, objective functions and alpha values. Reporting the 50^{th}, 70^{th} and 90^{th} percentiles.

		D			DM			RS05			RS50			RS100		
	α	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}	P_{50}	P_{70}	P_{90}
orb09	$\mathcal{N}_{25}$	1029.15	1052.54	1087.86	1016.45	1041.42	1078.90	1036.66	1062.28	1100.82	1013.48	1038.98	1077.11	**1007.62**	**1032.39**	**1069.69**
		+2.14%	+1.95%	+1.70%	+0.88%	+0.87%	+0.86%	+2.88%	+2.90%	+2.91%	+0.58%	+0.64%	+0.69%	–	–	–
	$\mathcal{N}_{50}$	1158.34	1202.51	1269.09	**1119.84**	**1165.95**	**1236.15**	1184.13	1234.50	1310.32	1135.69	1183.27	1255.28	1125.26	1173.70	1246.89
		+3.44%	+3.14%	+2.66%	–	–	–	+5.74%	+5.88%	+6.00%	+1.42%	+1.49%	+1.55%	+0.48%	+0.66%	+0.87%
orb10	$\mathcal{N}_{25}$	1053.86	1076.17	1110.22	1052.63	1075.76	1111.14	1050.57	1074.02	1109.84	1040.17	1062.89	1097.31	**1035.81**	**1058.61**	**1093.22**
		+1.74%	+1.66%	+1.56%	+1.62%	+1.62%	+1.64%	+1.42%	+1.46%	+1.52%	+0.42%	+0.40%	+0.37%	–	–	–
	$\mathcal{N}_{50}$	1193.23	1236.18	1301.69	1187.67	1233.03	1302.40	1216.17	1263.05	1334.25	1176.75	1220.65	1287.79	**1165.23**	**1209.19**	**1276.32**
		+2.40%	+2.23%	+1.99%	+1.93%	+1.97%	+2.04%	+4.37%	+4.45%	+4.54%	+0.99%	+0.95%	+0.90%	–	–	–
ta01	$\mathcal{N}_{25}$	1407.03	1431.63	1469.06	1403.25	1428.61	1467.07	1447.74	1475.52	1517.87	1388.26	1415.17	1456.31	**1381.72**	**1407.97**	**1448.44**
		+1.83%	+1.68%	+1.42%	+1.56%	+1.47%	+1.29%	+4.78%	+4.80%	+4.79%	+0.47%	+0.51%	+0.54%	–	–	–
	$\mathcal{N}_{50}$	1610.40	1657.20	1728.19	**1583.34**	**1631.28**	**1704.21**	1692.89	1746.78	1829.08	1610.72	1660.24	1735.66	1589.94	1639.54	1716.36
		+1.71%	+1.59%	+1.41%	–	–	–	+6.92%	+7.08%	+7.33%	+1.73%	+1.78%	+1.85%	+0.42%	+0.51%	+0.71%
ta02	$\mathcal{N}_{25}$	1414.82	1439.94	1478.07	1403.56	1429.40	1468.79	1430.67	1458.66	1501.32	1383.51	1409.58	1449.30	**1377.75**	**1404.44**	**1444.97**
		+2.69%	+2.53%	+2.29%	+1.87%	+1.78%	+1.65%	+3.84%	+3.86%	+3.90%	+0.42%	+0.37%	+0.30%	–	–	–
	$\mathcal{N}_{50}$	1608.01	1655.45	1727.48	**1584.16**	**1630.70**	**1701.55**	1674.35	1727.50	1808.71	1587.18	1638.12	1716.24	1584.76	1635.10	1712.24
		+1.51%	+1.52%	+1.52%	–	–	–	+5.69%	+5.94%	+6.30%	+0.19%	+0.46%	+0.86%	+0.04%	+0.27%	+0.63%
ta03	$\mathcal{N}_{25}$	1395.08	1419.69	1456.65	1404.44	1429.29	1467.22	1431.43	1458.66	1500.38	1385.85	1411.51	1450.47	**1370.52**	**1395.57**	**1433.41**
		+1.79%	+1.73%	+1.62%	+2.47%	+2.42%	+2.36%	+4.44%	+4.52%	+4.67%	+1.12%	+1.14%	+1.19%	–	–	–
	$\mathcal{N}_{50}$	1595.67	1642.39	1712.53	**1577.16**	**1625.27**	**1698.77**	1723.42	1778.49	1862.05	1605.62	1655.29	1730.80	1602.86	1653.83	1731.30
		+1.17%	+1.05%	+0.81%	–	–	–	+9.27%	+9.43%	+9.61%	+1.81%	+1.85%	+1.89%	+1.63%	+1.76%	+1.91%
ta11	$\mathcal{N}_{25}$	1630.34	1656.43	1696.24	1574.24	1599.45	1638.03	1644.42	1672.84	1716.15	1592.30	1618.72	1659.21	**1566.77**	**1593.45**	**1634.06**
		+4.06%	+3.95%	+3.81%	+0.48%	+0.38%	+0.24%	+4.96%	+4.98%	+5.02%	+1.63%	+1.59%	+1.54%	–	–	–
	$\mathcal{N}_{50}$	1854.05	1903.15	1977.91	**1801.73**	**1850.93**	**1926.53**	1926.76	1980.38	2061.70	1831.88	1883.50	1962.28	1818.60	1868.43	1944.77
		+2.90%	+2.82%	+2.67%	–	–	–	+6.94%	+6.99%	+7.02%	+1.67%	+1.76%	+1.86%	+0.94%	+0.95%	+0.95%
ta12	$\mathcal{N}_{25}$	1614.87	1640.15	1678.44	1576.75	1602.11	1640.91	1661.07	1688.14	1729.23	1571.01	1597.44	1637.95	**1567.21**	**1593.30**	**1633.16**
		+3.04%	+2.94%	+2.77%	+0.61%	+0.55%	+0.47%	+5.99%	+5.95%	+5.88%	+0.24%	+0.26%	+0.29%	–	–	–
	$\mathcal{N}_{50}$	1848.91	1896.56	1969.14	**1784.48**	**1831.70**	**1904.21**	1943.88	1999.32	2084.32	1861.15	1912.73	1992.01	1843.19	1893.27	1970.30
		+3.61%	+3.54%	+3.41%	–	–	–	+8.93%	+9.15%	+9.46%	+4.30%	+4.42%	+4.61%	+3.29%	+3.36%	+3.47%
ta13	$\mathcal{N}_{25}$	1557.55	**1583.25**	**1622.10**	1574.09	1600.61	1641.11	1626.02	1655.03	1699.14	1564.69	1591.19	1631.64	**1556.75**	1583.73	1624.96
		+0.05%	–	–	+1.11%	+1.10%	+1.17%	+4.45%	+4.53%	+4.75%	+0.51%	+0.50%	+0.59%	–	+0.03%	+0.18%
	$\mathcal{N}_{50}$	1789.62	1838.25	1911.94	**1765.83**	**1814.66**	**1889.30**	1914.19	1969.26	2053.31	1824.19	1876.16	1955.22	1822.89	1874.17	1951.96
		+1.35%	+1.30%	+1.20%	–	–	–	+8.40%	+8.52%	+8.68%	+3.31%	+3.39%	+3.49%	+3.23%	+3.28%	+3.32%

Table A.3.: Computational results for different instances, objective functions and alpha values. Reporting the 50^{th}, 70^{th} and 90^{th} percentiles.

		D		DM		RS05		RS50		RS100	
Instance	Size	$\mathcal{N}_{25}$	$\mathcal{N}_{50}$	$\mathcal{N}_{25}$	$\mathcal{N}_{50}$	$\mathcal{N}_{25}$	$\mathcal{N}_{50}$	$\mathcal{N}_{25}$	$\mathcal{N}_{50}$	$\mathcal{N}_{25}$	$\mathcal{N}_{50}$
abz5	small	23.80	23.58	36.21	29.64	96.19	86.97	1018.90	652.95	1859.21	1151.55
abz6	small	68.56	24.26	67.00	29.15	109.56	84.92	864.73	647.36	1598.88	1255.14
abz7	large	142.90	62.42	94.80	79.03	290.16	266.93	2431.62	2172.84	4435.96	4174.18
abz8	large	73.23	63.76	88.49	76.33	292.92	271.11	2308.34	2051.21	4394.52	3772.26
abz9	large	129.66	68.00	91.30	78.13	311.91	264.93	2211.15	2109.43	4283.85	3995.56
ft06	small	17.18	16.64	19.48	21.80	62.89	54.37	361.73	349.11	917.60	662.41
ft10	small	31.55	24.47	33.60	30.40	90.58	90.09	925.45	701.98	1921.56	1347.13
ft20	small	23.12	22.74	36.13	30.66	183.42	99.61	981.07	704.90	2027.74	1437.17
orb01	small	25.56	25.85	30.41	31.18	97.20	95.27	748.40	720.21	1435.17	1356.68
orb02	small	25.00	24.44	31.80	31.69	95.56	91.38	721.91	695.55	1351.51	1331.69
orb03	small	26.65	25.69	33.39	30.43	97.11	87.46	741.67	700.98	1422.29	1316.38
orb04	small	25.52	25.03	33.93	31.95	101.29	100.97	754.88	695.29	1409.22	1201.05
orb05	small	31.70	25.21	41.09	31.90	104.10	95.71	728.21	716.66	1404.32	1335.94
orb06	small	30.53	25.34	36.37	31.82	98.39	89.88	733.33	679.56	1375.98	1382.34
orb07	small	31.65	29.07	36.15	34.16	110.72	98.89	798.27	748.16	1489.00	1403.37
orb08	small	28.94	24.50	41.66	32.46	106.76	92.55	751.66	677.05	1459.45	1376.77
orb09	small	29.87	23.49	38.77	32.49	108.80	95.23	734.75	702.99	1521.01	1476.73
orb10	small	29.84	23.61	40.09	30.79	100.23	90.36	711.37	695.84	1419.48	1338.07
ta01	medium	49.97	46.02	65.63	61.23	192.53	203.08	1584.43	1508.83	3013.60	3068.09
ta02	medium	49.01	50.67	66.60	68.16	194.07	204.95	1554.84	1530.63	2993.13	3127.59
ta03	medium	55.85	56.25	63.15	69.05	199.21	190.47	1487.23	1463.42	2956.04	2935.11
ta11	large	65.12	69.24	80.07	81.85	258.80	278.78	2010.35	2026.28	3852.84	3657.45
ta12	large	66.24	84.64	84.55	98.69	247.68	271.53	1947.07	1715.25	3756.75	3317.50
ta13	large	63.28	75.96	83.55	76.55	241.92	241.43	1893.95	1583.76	3667.23	2819.16

Table A.4.: Average computation time in seconds