Comparison of Scheduling Methods for Different Cases of a Complex Flexible Job Shop Problem

A.J. van Stiphout

master's thesis

Industrial Engineering & Management

UNIVERSITY OF TWENTE.

Master's Thesis

Industrial Engineering & Management

Production & Logistics Management Specialization

Author

A.J. van Stiphout

University of Twente

Drienerlolaan 5

7522 NB Enschede

First supervisor

Dr.ir. J.M.J. Schutten

Associate Professor

Second supervisor

Dr. P.C. Schuur

Associate Professor

Date

17 September 2020

Management Summary

We conduct this research for Syze (a fictitious name used to ensure anonymity), the developer of an Enterprise Resource Planning (ERP) system, mainly focused on manufacturing companies. One of the components of the ERP-system is the finite production scheduler (FPS), which creates an operational schedule of the operations to be processed on different machines of Syze's customers, by solving a Flexible Job Shop Problem (FJSP).

An FJSP can be solved by assigning each operation to a machine and determining its start and end time. Each operation belongs to a sequence (or precedence structure) of operations to be completed in that order. We call this sequence a job, which also has a due date. The goal is to complete jobs before their due date as much as possible. We score the performance of a schedule using average tardiness, which refers to how much later than its due date each job is completed (zero when the job is completed before the due date). The FJSP is further extended with constraints concerning release dates for jobs, preventive maintenance on machines, queue, wait and setup times of operations, and tree precedence structures in jobs.

Currently, only one scheduling method is implemented in the FPS, while many other methods exist in literature. Syze wants to improve the FPS in order to better serve their customers. In order for Syze to be able to decide with which scheduling methods they should extend the FPS, and to give their customers advice on what scheduling method to use for their case, this research aims to find out what **methods** perform **best** in solving different **cases** of an FJSP by selecting and applying methods to available cases.

Methods

Table M.1 shows all scheduling methods we find in literature, split into the categories of constructive and improvement methods. Constructive methods generate a single schedule. Improvement methods start from one or multiple initial schedule(s) and then improve it (them). Based on Syze's criteria of explainability and repeatability, we select Dispatching Rules (DRs), Shifting Bottleneck (SB), Tabu Search (TS), Greedy Random Adaptive Search Procedure (GRASP) and Genetic Programming (GP) as methods to test.

Cases

We initially select 6 customer cases to test the scheduling methods on, as these are the only customer schedule data we have available. The cases vary in their usage of constraints, e.g. some use preventive

Table M.1 | Scheduling methods found in literature

	CONSTRUCTIVE		IMPROVEMENT	
Dispatching Rules (DRs)	Earliest Due Date	EDD	Greedy random adaptive search procedure	GRASP
	Earliest Operation Due Date	ODD	Tabu Search	TS
	Shortest Processing Time	SPT	Genetic Programming	GP
	Allowance per Operation	A/OPN	Simulated Annealing	SA
	Slack per Operation	S/OPN	Genetic Algorithm	GA
	Minimum Slack Time	MST	Particle Swarm Optimization	PSO
	Smallest Critical Ratio	SCR	Artificial Bee Colony	ABC
	Longest Processing Time	LPT	Bi-population-based Estimation of Distribution	
	Shortest Remaining Processing Time	SRPT	Harmony Search	
	Longest Remaining Processing Time	LRPT	Steepest Decent with Perturbation	
	Shifting Bottleneck	SB	Guided Local Search with Shifting Bottleneck	
	Branch and Bound	BnB	Fuzzy Evolutionary	
			Clonal Selection	
			Knowledge-based Ant Colony Optimization	
			Parallel Variable Neighborhood Search	
			Artificial Immune Algorithm	
			Hybrid Shuffled Frog-leaping Algorithm	

maintenance while others do not, and the values of various attributes, e.g. one has a horizon of 54 days while another has a horizon of 1145 days.

For each of these 6 cases, the initial data they are based on is combined with randomness (using mostly empirical distributions) to create 5 data instances per case. This increases the reliability of our conclusions.

Best performers

As customers differ in how much computation time is available to generate a schedule (a matter of either minutes or hours), we separately discuss constructive methods, which require several minutes of computation time, and improvement methods, which can only achieve better results the more time is spent.

Of the constructive methods, the scheduling method currently used by Syze, the Earliest Operation Due Date DR (ODD), performs best (regarding average tardiness) in 3 of the 6 cases. In 2 of the other cases, which are the only cases that include preventive maintenance, the Longest Remaining Processing Time DR (LRPT) performs best. When adapting one of these cases to exclude preventive maintenance, ODD becomes the best performer, thereby proving that LRPT is better when using

preventive maintenance and ODD is best otherwise. In the final case, SB is the best performer. Due to SB's inability to cope with larger schedule instances (i.e. having more jobs, operations and/or machines), this last case is the only case SB can be applied upon. The second-best performer in this case is the Shortest Processing Time DR (SPT). It is unclear why this is the second-best performer, instead of ODD and LRPT that have proven effective in the other cases.

Of the improvement methods, TS performs best for 3 of the 6 cases, while GRASP performs best for the other half. TS performs better than GRASP on average over all cases. Factors that are proven to negatively impact the performance of GRASP are not using release dates and not using tree precedence structures. A factor that is proven to positively impact the performance of GRASP is not using preventive maintenance.

Recommendations

Considering these findings, we deem LRPT, TS and GRASP as methods that should be implemented by Syze in their ERP system. As one DR is already implemented in the software system, implementing another will require less resources than implementing an entirely different method, which makes LRPT the quickest win. After that, we recommend implementing TS, as it on average performs better than GRASP, which is then left as the final method to be implemented. For now, we do not suggest implementing SB due to its inapplicability to larger cases.

Acknowledgements

This document is the fruit of six months' labor. During these six months, the world changed due to COVID, and so did my working conditions. I started off working in the same office as Bert, Wim, Oepke, Maaike, Henk, Sicco and Karstjan, whom I regularly joined in their daily ritual of planking, during which they just as regularly outperformed me. I thank them for their welcoming posture towards me, even though I did not see them face-to-face as often as I would have liked.

I further thank Marcel for our shared car rides, Stephan for his help in data-gathering, and Adriaan & Ron for their day-to-day supervising and question-answering.

On the university side of things, I thank Marco and Peter for their supervisory roles. This thesis would not have the same quality without their constructive criticism.

Most of this thesis was written while working from home, and I therefore thank my housemates Ruben and Erik for their support and our shared coffee breaks. I am thankful for the support of many other friends and family members as well. Their names are too many to mention, but they know who they are.

Finally, yet most importantly, I thank God, who is my Rock and my Salvation. He made me who I am today and guided me through this entire project.

Table of Contents

List of Abbreviations

ABBR.	MEANING	EXPLANATION
A/OPN	Allowance per Operation dispatching rule	DR sorting operations by minimum remaining Allowance per Operation of the job
ABC	Artificial Bee Colony	An improvement scheduling method
AL	Approach by Localization	A method to create an initial JSP solution for GA
ANS	Adaptive Neighborhood Search	A local search method
ASX	Assignment Crossover	A GA operator
avg.	average	
BnB	Branch and Bound	A constructive scheduling method
DEAP	Distributed Evolutionary Algorithms Python	Python library used for implementing GP
DR	Dispatching Rule	A type of constructive scheduling method, which sorts operations by some variable
EDD	Earliest Due Date dispatching rule	DR sorting operations by Earliest job Due Date
e.g.	exempli gratia	Latin phrase meaning “for example”
EOX	Enhanced Order Crossover	A GA operator
ERP	Enterprise Resource Planning	Software that helps companies gain insight into data relating to all core business activities and resources
FJSP	Flexible Job Shop Problem	Same as JSP, with the addition that operations can be assigned to one of multiple machines
FPS	Finite Production Scheduler	ERP component that creates an operational schedule of jobs to be processed on machines
GA	Genetic Algorithm	An improvement scheduling method
GOX	Generalized Order Crossover	A GA operator
GRASP	Greedy Random Adaptive Search Procedure	An improvement scheduling method
HTS/SA	Hybrid Tabu Search/Simulated Annealing	An improvement scheduling method
INSA	Insertion Algorithm	A method to create an initial JSP solution for TS
JBX	Job-based crossover	A GA operator
JSP	Job Shop Problem	Mathematical problem involving the scheduling of production orders, called jobs, on machines that can each process one job at a time
KPI	Key Performance Indicator	A variable to measure performance
LB	Lower Bound	Lowest found value of some objective function
LOX	Linear Order Crossover	A GA operator
LPT	Longest Processing Time dispatching rule	DR sorting operations by Longest Processing Time
LRPT	Longest Remaining Processing Time dispatching rule	DR sorting operations by Longest Remaining Processing Time of the job
max.	maximum	
min.	minutes	
MPT	random sequencing after routing to optimal machine with Minimal Processing Time	A method to create an initial JSP solution for TS
MPX	Multipoint Preservative Crossover	A GA operator
MST	Minimum Slack Time dispatching rule	DR sorting operations by minimum remaining Slack of the job
no.	number of	

ABBR.	MEANING	EXPLANATION
norm.	normalized	
ODD	Earliest Operation Due Date dispatching rule	DR sorting operations by earliest Operation Due Date
PPS	Precedence Preserving Shift mutation	A GA mutation operator
POX	Precedence preserving Order-based Crossover	A GA operator
PSO	Particle Swarm Optimization	An improvement scheduling method
S/OPN	Slack per Operation dispatching rule	DR sorting operations by minimum remaining Slack per Operation of the job
SA	Simulated Annealing	An improvement scheduling method
SB	Shifting Bottleneck	A constructive scheduling method
SCR	Smallest Critical Ratio dispatching rule	DR sorting operations by Smallest Critical Ratio (remaining allowance over remaining time required)
SMP	Single Machine Problem	A simple JSP with only one machine
SPT	Shortest Processing Time dispatching rule	DR sorting operations by Shortest Processing Time
SRPT	Shortest Remaining Processing Time dispatching rule	DR sorting operations by Shortest Remaining Processing Time of the job
s.t.	such that	
st. dev.	standard deviation	
TS	Tabu Search	An improvement scheduling method
UB	Upper Bound	Highest found value of some objective function
VNS	Variable Neighborhood Search	A local search method

1 Introduction

This chapter introduces the aim of this research by describing background information (Section 1.1), identifying the problem to solve (Section 1.2) and the research questions used (Section 1.3).

1.1 Background

Syze (a fictitious name used to ensure anonymity) is the developer of an Enterprise Resource Planning (ERP) system, which is software that helps companies gain insight into data relating to all core business activities and resources, such as sales, accounting, production and human resource management. They have more than 15,000 employees, located in about 50 countries, working for about 70,000 customers. Their focus is primarily on the manufacturing industry, with customers operating in industries ranging from automotive to food.

One of their unique selling points is that they do not merely develop a product, but also help their customers with the implementation process of the system and provide advice on how to best set up the various functionalities the system incorporates.

Syze launched a new version of their product some years ago. Due to the technical changes involved in this new product, some functional components must be recreated from scratch. One of these components is the Finite Production Scheduler (FPS), which creates an operational schedule of the jobs to be processed on the machines of Syze's customers by solving a Job Shop Problem (JSP).

Due to the diverse nature of Syze's customers, their requirements of the FPS differ largely, with schedule horizons ranging from seven weeks to three years and the average amount of jobs per day ranging from 1.5 to 100.

1.2 Problem identification

An initial version of the FPS has been created, but customers are not completely satisfied yet, as this FPS does not fully meet their needs. They want to use the scheduler with the best scheduling method for their specific case but are unable to because of two reasons.

Primarily, the FPS offers no choice in methods, and therefore needs to be extended with more scheduling methods. Syze only has limited development resources, and therefore has to prioritize what methods to implement next. This also causes the second, more important reason: Syze is currently unable to advise their customers which method is optimal for their case, as they have no insight into what their case looks like, nor which method fits that case.

Gaining this knowledge would solve both the recommendation and the prioritization problem, as determining the added value of methods enables prioritization. Figure 1.1 shows the problem cluster (Heerkens, 2012) with an overview of the mentioned problems and their causal relationships.

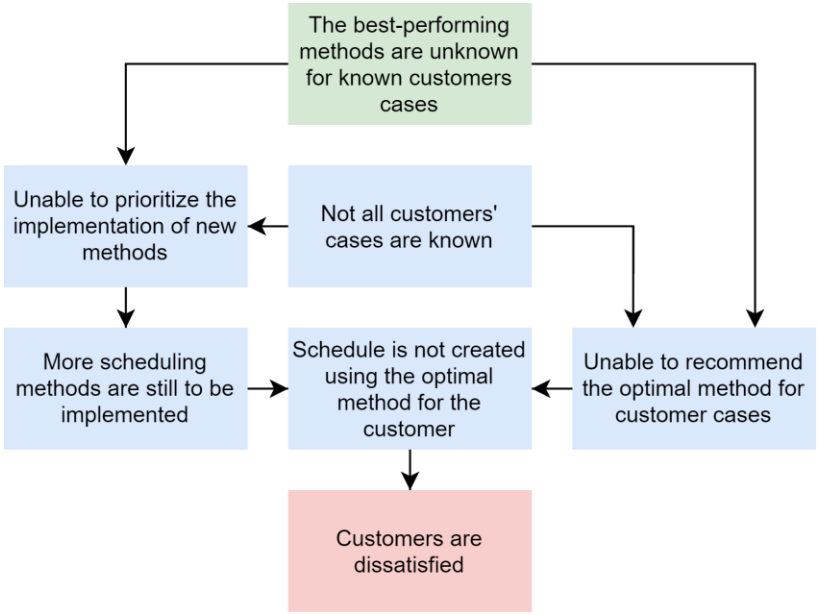

Figure 1.1 | Problem cluster containing an action problem (red) and the underlying core problem (green)

1.3 Research questions

Based on the selected core problem, we formulate the research question this research aims to solve as follows:

*"Which **methods** perform **best** in solving a Job Shop Scheduling problem for different **cases** of customers of **Syze's software**?"*

We define a method as a complete algorithm used for solving the JSP. Chapter 2 gives a definition of the JSP and describes the results of a literature review to determine which methods exist to solve it, answering the research question:

*"What **methods** are available for solving a JSP?"*

In order to provide insight into the context, Chapter 3 describes the current situation, based on internal documents and interviews with personnel. By establishing the used JSP model and scheduling method, and determining some attributes of the jobs that customers schedule, it answers the research question:

*"How is the scheduling functionality currently implemented and used in **Syze's software**?"*

By elaborating on the selection and generation of data instances to test the found methods on, and selecting scheduling methods that might be suitable for the FPS, Chapter 4 answers the research question:

*"What different customer's scheduling **cases** and **methods** are relevant to the FPS?"*

Chapter 5 elaborates on the decisions of what methods to test, and shows the computational results of the found methods for the determined cases, answering the research question:

*"What methods are **best** for which cases?"*

Chapter 6 states the solution for the core problem, along with limitations of this research and recommendations for future research.

2 Literature review

This chapter establishes methods for solving the job shop problem, by giving a formal definition of the job shop problem (Section 2.1) and showing the result of a literature review of both constructive (Section 2.2) and improvement methods (Section 2.3) used to solve the JSP. Section 2.4 explains how generation schemes can be used to turn the results of these methods into a functional schedule.

2.1 Job Shop Problem

Job shop scheduling is a classic problem, first introduced in 1963 (Muth, Thompson, & Winters, 1963). In its simplest form, it involves scheduling production orders, called jobs, on machines that can each process one job at a time. The processing of a job on a machine is called an operation, which requires use of a given machine for a given amount of time without being interrupted. Each job consists of one or more required operations, which need to be completed in a certain sequence, forming a chain of operations. Scheduling involves allocating time on a machine for all operations during which they are processed. The resulting schedule should be feasible (i.e. satisfy all constraints) and optimize some objective. This section first covers two ways to model the JSP, then elaborates on some additional constraints which can be used, and finally discusses the most often used JSP objectives.

2.1.1 JSP model

There are two common ways to represent the JSP. The first is a mathematical model, with an objective function and constraints. Below is the mathematical model used by Adams, Balas and Zawack (1988). Here $N = \{0, 1, \dots, n\}$ denotes the set of operations to be scheduled, with 0 and n as dummy start and finish operations, A the set of precedence pairs of operations, M the set of machines and E_k the set of operations to be scheduled on machine k. The processing time of operation i is denoted by p_i and t_i refers to the completion time of operation i.

$$\min t_n \tag{1}$$

s.t.

$$t_j - t_i \geq p_i, \qquad \forall (i,j) \in A \tag{2}$$

$$t_i \geq 0, \qquad \forall i \in N \tag{3}$$

$$t_j - t_i \geq p_j \vee t_i - t_j \geq p_i, \qquad \forall (i,j) \in E_k, k \in M \tag{4}$$

The objective function (1) refers to the makespan, i.e. the time required to complete all jobs. This is the most frequently used objective in literature. Constraint (2) makes sure precedence relationships are

considered. Constraint (3) sets the starting time of the schedule to 0. Lastly, Constraint (4) makes sure no two operations are processed on the same machine at once.

The second representation is the Disjunctive Graph (Balas, 1969). Here, the problem is modelled using nodes, conjunctive arcs and disjunctive arcs. Each node represents an operation, each initially conjunctive arc a precedence relationship between operations, and each disjunctive arc a (possible) machine sequential relationship. Each pair of operations that requires processing on the same machine is connected by such a disjunctive arc. When an operation is scheduled on a machine, all of its disjunctive arcs are made conjunctive, in the direction from the node representing the scheduled operation to the node of the other operations to be scheduled on the same machine. When all operations are scheduled, the makespan of the schedule can be determined by calculating the longest path from the starting node to the end node. Figure 2.1 gives an example.

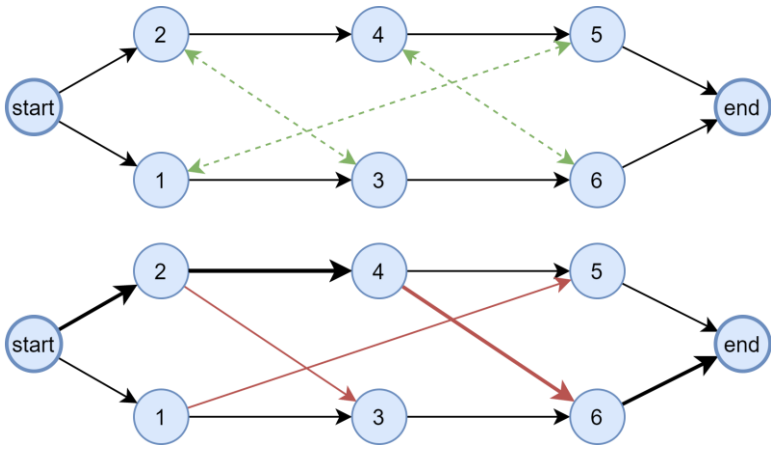

Figure 2.1 | A disjunctive graph (above) and its corresponding optimal solution (below). The numbers in the node refer to each operation's processing time. Disjunctive arcs are green, their conjunctive counterparts are red. The bold arcs form the longest path in the graph and therefore determine the makespan of this schedule.

2.1.2 Additional constraints

A simple addition to the JSP is the use of release dates (Brucker, Jurisch, & Sievers, 1994). The release date of a job refers to the moment in time this job becomes available for processing, meaning processing can start only after a job's release date.

More complex additions are preventive maintenance (Li, Pan, & Tasgetiren, 2014), where there are certain downtimes scheduled on machines during which no processing can take place, and variable changeover times (Saidi Mehrabad & Fattahi, 2007). The latter refers to an additional time, called setup time, that must be completed on a machine before the operation can start processing. The length of this changeover time depends on the operation that was processed previously. This models for example the

requirement to clean a painting machine when the next operation requires a different color. These two additions, though common in practice, are rare in literature.

The Flexible Job Shop Problem (FJSP) is a JSP adaption that includes the so-called multi-purpose machines extension, meaning an operation is assigned not to a machine, but to a work center consisting of multiple machines, one of which must process the operation. Each operation might have different processing times on different machines. The FJSP is a two-fold problem: on the one hand each operation must be assigned to a machine of its work center (called the routing subproblem), and on the other hand each machine's operations have to be sequenced (called the sequencing subproblem), just as in the regular JSP. Methods can either approach the FJSP hierarchically, meaning the routing and sequencing subproblems are solved sequentially, or concurrently, meaning they are solved at the same time.

2.1.3 Objective functions

As mentioned, the most frequently used objective function in literature is minimum makespan, which is the time required for completing all jobs. However, in practice, completing jobs before their due dates is a more important objective than minimizing makespan (Baker K., 1984). Due dates refer to the moment in time a job should be completed in order to deliver the product to the customer in time. The degree to which due dates are met can be measured using either the maximum or average of lateness or tardiness of jobs. Lateness is the difference between a job's completion time and the due date, which is positive if a job is completed late and negative if a job is completed early. Tardiness is the maximum of a job's lateness and zero, meaning it is zero if a job is completed in time, and otherwise is equal to lateness.

For the FJSP, as explained in Section 2.1.2, some methods in literature solve a multi-objective function, meaning it optimizes multiple objectives at once. Usually these objectives are minimal makespan, minimal total workload and minimal critical machine workload. Workload refers to the total amount of processing time required (which can change in the FJSP as the processing time of an operation differs depending on the machine it is processed on). The critical machine refers to the machine with the highest workload.

This multi-objective function is often solved using a Pareto-based approach (e.g. in Chiang & Lin, 2013). This approach results in multiple solutions that are all nondominated, meaning none of the objectives can be improved without worsening one of the other objectives.

2.2 Constructive methods

JSP is an NP-hard problem, meaning it is thus far impossible to solve to optimality in polynomial time (Garey, Johnson, & Sethi, 1976). This creates a need for smart methods that generate a (near-)optimal schedule without proving it is optimal and thereby formally solving the JSP. Many of these methods have been developed since the introduction of the problem. They can be subdivided in constructive methods, which create a single schedule based on some logic, and improvement methods, which aim to improve one or multiple initial schedule(s). This section describes constructive methods, and Section 2.3 describes improvement methods. For each method, first the application on the JSP is described, followed by the application on the FJSP. Each method also takes the release dates constraint into account and aims to minimize makespan, unless it is mentioned that it solves a multi-objective function, in which case it solves for the objectives of minimal makespan, minimal total workload and minimal critical machine workload (recall Section 2.1.3).

2.2.1 Dispatching rules

Dispatching rules were the first methods used to solve the JSP. These methods aim to prioritize operations using a certain rule. The operation with the highest priority (and no unscheduled precedents) is scheduled on the first possible moment, after which the priorities of remaining unscheduled operations (with the operations that now have now predecessors added) are reassessed and the next highest priority operation is scheduled. This process repeats until all operations are scheduled.

The advantage of using such rules is that they are easy to implement, simple to understand and fast in execution. However, the objective function value they result in is usually of lesser quality than other, more advanced methods.

Baker (1984) presents an overview of dispatching rules for solving the JSP and their performance with regards to certain KPIs. An overview of the methods that he found to perform well, can be found in Table 2.1. Allowance is defined as the difference between an operations due date and its current earliest possible starting time, taking currently scheduled operations into account. Slack is the difference between an operation's allowance and remaining time required to complete that operation.

Kalma (2020) proposes a method to apply these dispatching rules to the FJSP, by scheduling each prioritized operation on the machine that would result in the earliest completion time. He also proposes three additional dispatching rules, found in Table 2.2.

Table 2.1 | Dispatching rules (Baker K., 1984)

Dispatching rule	Abbr.	Criterion
Earliest Due Date	EDD	All operations of the job with the earliest due date
Earliest Operation Due Date	ODD	The operation with the earliest due date
Shortest Processing Time	SPT	The operation with the shortest processing time
Allowance per Operation	A/OPN	The next operation of the job with minimum remaining allowance per operation
Slack per Operation	S/OPN	The next operation of the job with minimum remaining slack per operation
Minimum Slack Time	MST	The next operation of the job with minimum remaining slack
Smallest Critical Ratio	SCR	The next operation of the job with the smallest ratio of remaining allowance over remaining time required

Table 2.2 | Additional dispatching rules (Kalma, 2020)

Dispatching rule	Abbr.	Criterion
Longest Processing Time	LPT	The operation with the longest processing time
Shortest Remaining Processing Time	SRPT	The next operation of the job with the shortest total time remaining
Longest Remaining Processing Time	LRPT	The next operation of the job with the longest total time remaining

2.2.2 Shifting bottleneck

A step up in complexity from dispatching rules is the shifting bottleneck algorithm (SB) for solving a JSP (Adams, Balas, & Zawack, 1988). Instead of the single pass the dispatching rules make, SB iterates multiple times over the same operations to achieve better results, as represented in Figure 2.2 gives a high-level representation of the algorithm. The mentioned single machine problems (SMPs) are simplifications of the JSP, where only a single machine and the operations to be scheduled on that machine are considered. Solving the SMPs in step 2 and 5 is done using Carlier's algorithm (Carlier, 1982), a branch and bound method (see Section 2.2.3).

In the SMPs, operations are given a release date equal to the longest path to the node representing that operation in the disjunctive graph (recall Section 2.1.1). Initially, these release dates were fixed during the process of solving the SMP. Later, it was found more effective to update release dates every time an operation was scheduled, as this leads to a more accurate SMP and better end-results (Dauzere-Peres & Lasserre, 1993).

SB can be adapted to solve the FJSP (Kalma, 2020). This is done by not solving a single machine problem, but a parallel machine problem, relating to a single work center. This assumes do not belong to multiple work centers. The problem does arise that there is no fast method for solving a parallel machine problem to optimality. Kalma (2020) therefore implements the SRPT dispatching rule, which is fast but

does not solve to optimality, in contrast to Carlier's algorithm that does solve the SMP to optimality in the JSP case but requires more computation time.

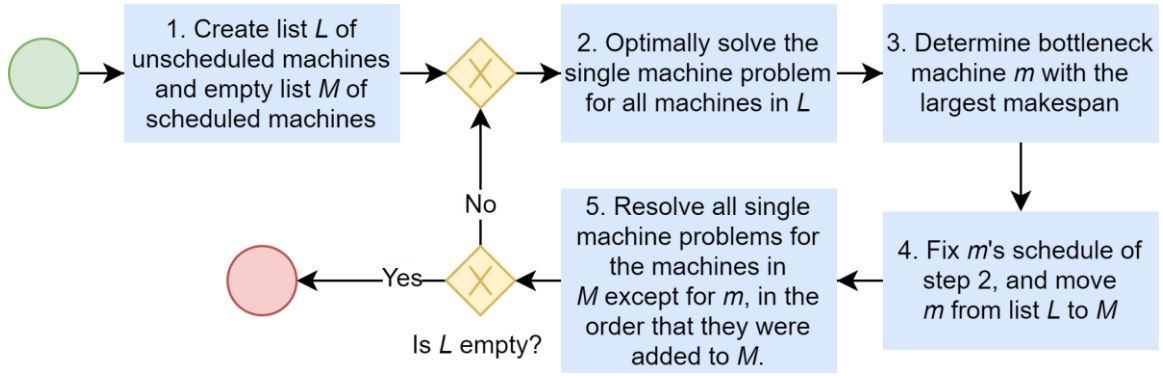

Figure 2.2 | The shifting bottleneck algorithm

2.2.3 Branch and bound

Branch and bound (BnB) is a method to solve the JSP to optimality. This is done by systematically enumerating all available options using a rooted tree, starting from a root node and ending at leaf nodes, with many branch nodes in between.

For each of these nodes, a lower bound (LB) and an upper bound (UB) of the objective function value is determined. Using some logic taking the LB and UB into account, a node is selected to create new branch nodes from, until a stopping criterion is met. As an example of this logic, when minimizing (maximizing), the node with the lowest LB (highest UB), and therefore intuitively the most potential for reaching the optimal value, can be selected for branching. The LB (UB) of each node is usually determined by relaxing some constraints such that the problem becomes easily solvable. The UB (LB) of each node is usually determined by applying a fast and simple heuristic. BnB stops when the LB and UB become equal at some node and no lower LB (higher UB) exists at any other node. This node then refers to the optimal schedule.

A well-performing implementation of BnB for solving the JSP uses a theorem based on critical blocks (Brucker, Jurisch, & Sievers, 1994). A critical block is a subset of operations in this critical path that are processed consecutively on the same machine in sequence. A critical path is a combination of operations that are connected via precedence constraints or a machine's sequence, leading to the makespan of the schedule. Therefore, the makespan of a schedule cannot be reduced without changing some of the operations in the critical path. This critical path is equal to the longest path in the disjunctive graph. It is proven that the makespan only improves when an operation of this block is moved to the first

or last position in this block. Using this fact, a branching tree can be created starting from an initial schedule (created by iteratively scheduling the operations that have minimal effect on the makespan of the partial schedule when added to it), and then creating two branches for every operation in a critical path block, moving it either to the front or to the back.

Due to the nature of BnB, these algorithms are very time consuming. When a JSP increases in size, the number of nodes required to find an optimal solution increase with it, and therefore so does the required computational time. No BnB implementation that solves the FJSP is known to us.

2.3 Improvement methods

So far, we have covered dispatching rules and the shifting bottleneck algorithm, which are fast but lack in performance, and branch and bound methods, which perform well but require a long time to do so. Improvement methods are in the middle ground. They perform better than DRs and SB and are faster than BnBs. However, the reverse is also true. In general, improvement methods perform worse but are faster than branch and bound methods, and are slower but better performing than DRs and SB.

Improvement methods usually incorporate a local search component, where a starting solution is improved by making small changes. These changes create neighboring solutions by for example swapping two random operations. This swap is an example of a neighborhood operator. Many different operators exist, each resulting in a different neighborhood structure.

In the steepest descent local search, the best neighbor out of the set of all neighbors is selected to replace the starting solution as the current solution if it performs better. This continues until no better neighbor of the current solution exists. These methods can also be applied to the multi-objective JSP (Li, Pan, & Chen, 2012).

If the neighborhood operator (and therefore the structure) changes per search iteration, it is called a variable neighborhood search (VNS) or adaptive neighborhood search (ANS) (Rego & Duarte, 2009).

The problem with a basic local search is that finds a local optimum, which is not necessarily a global optimum. Metaheuristics are a subtype of improvement methods, that uses some logic to overcome this local optimum problem.

2.3.1 Greedy random adaptive search procedure

Called GRASP for short, the greedy random adaptive search procedure consists of two phases (Aiex, Binato, & Resende, 2003). First a schedule is constructed by using an LRPT dispatching rule to select a

number of the highest-priority operations that can be scheduled without violating precedence constraints, and then randomly determining which one of those operations will be scheduled (with equal probability for each operation). This is iterated until a full schedule is created, completing the first phase. Then, a steepest descent local search is performed on the constructed schedule using a critical block swap as an operator, meaning the sequential position of two operations in the critical block, as defined in Section 2.2.3, are swapped. This iterates until a locally optimal schedule is found, completing the second phase. By repeating these two phases, multiple locally optimal schedules are generated, of which the best one is used as output for the GRASP method.

2.3.2 Simulated annealing

Simulated annealing (SA) is another, well-known method for solving JSPs (Laarhoven, Aarts, & Lenstra, 1992). It overcomes the problem of finding a local optimum by, while performing a local search, accepting neighbors based on a certain probability. This selection probability is a function of the improvement compared to the current solution and the cooling parameter c. The probability is non-zero even when the improvement is negative (and therefore a deterioration), allowing for the random acceptance of worse neighbors as the new current solution, via which a local optimum can be left in hope for arriving at a global optimum (or at least, a better local optimum). The cooling parameter c changes over time, such that the later iterations are less likely to accept worse neighbors, eventually ending in accepting mostly better solutions. To control this cooling process, three settings are required by SA: a starting value for c, a cooling function by which c changes each iteration and a final value for c, after which SA stops iterating. An additional setting is the Markov chain length L, which is the maximum number of times a new neighbor accepted (based on a c-dependent probability) per iteration before updating the value of c. In total, then, SA requires four parameters to be set.

It is proven that this method produces similar results to SB for small JSP instances (Laarhoven, Aarts, & Lenstra, 1992). The larger the JSP instance becomes, the better the performance of SA becomes when compared to SB. This does, however, come at the cost of larger running times. No application of SA to the FJSP is found.

2.3.3 Tabu search

The tabu search method (TS) overcomes the local optimum problem by keeping a list of solutions evaluated so far. These solutions are forbidden, or tabu, from the local search for as long as they are on the list. This forces the search to continue in unexplored areas. Figure 2.3 shows a simple version of the TS algorithm.

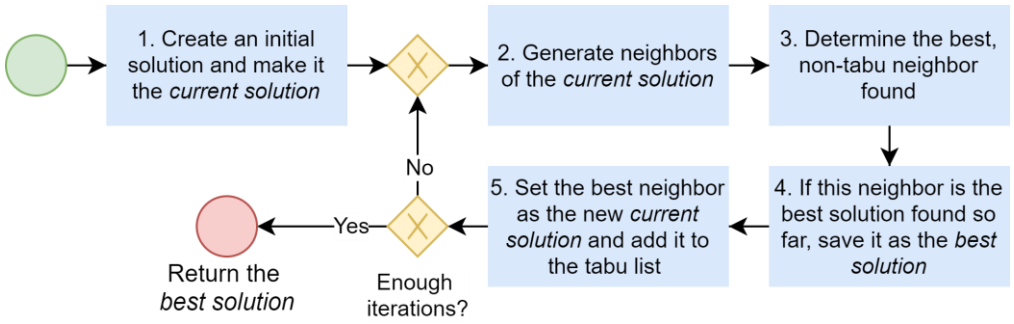

Figure 2.3 | The tabu search algorithm

In practice, instead of storing entire solutions which requires a large amount of memory and computation time (due to having to compare entire solutions with one another), the (inverse of) operators applied are usually stored. The search can then not move back to the previous solution because that change cannot be made. The problem that arises then is that other solutions that have not been selected previously, are made tabu as well, because the same operator is required to get to that solution. Often, an aspiration criterion is used to overcome this problem. This means that when a move that is on the tabu list results in an objective function value better than the best value found so far, it becomes an allowed move, ignoring its tabu status. Other additional strategies include tabu list resetting, where the tabu list resets if no improvement is made after a certain number of iterations; forbidden moves, where if only forbidden moves are available, the oldest move on the list is allowed anyway, in order to keep the search moving; and elite solutions, where, if no improvement is made after a certain number of iterations, the search moves to one of the top solutions (called elite solutions) found so far.

Many different applications of TS on the JSP and FJSP exist, with differences in starting solution generation, neighborhood operators and tabu list. Table 2.3 shows an overview of these applications. For details, the reader is referred to their respective sources. The differences in starting solutions and neighborhood operators is striking.

Worthy of a special mention due to its promising results is a method also presented by Fattahi, Mehrabad, & Jolai (2007), but a different one than in Table 2.3. This method is called Hybrid Tabu Search/Simulated Annealing (HTS/SA). As the name suggests, it involves a combination of tabu search, applied to the routing subproblem, and simulated annealing, applied to the sequencing subproblem.

Table 2.3 | Overview of various tabu search implementations

METHOD	FJSP APPROACH	STARTING SOLUTION GENERATION	NEIGHBORHOOD OPERATOR(S)	TABU MEMORY	ASP. CRIT.	ADDITIONAL STRATEGIES
Dell'Amico & Trubian (1993)	N/A	Bi-directional dispatching rule	Permute four machine sequential operations of which at most three are in a critical path block Move a critical operation to the head or tail of its block	Inverse operator	✓	Resetting
Nowicki & Smutnicki (1996)	N/A	Insertion Algorithm (INSA)	Move a critical operation to the head or tail of its block	Inverse operator	✓	Resetting Forbidden moves
Nowicki & Smutnicki (2005)	N/A	INSA	Move a critical operation to the head or tail of its block	Inverse operator	✓	Elite solutions
Brandimarte (1993)	Hierarchical	Best result of SPT or Most Work Remaining.	Critical operation insert (routing) Critical operations swap (sequencing)	Inverse operator	✓	-
Hurink, Jurisch, & Thole (1994)	Concurrent	LPT with insertion	Critical operation insert (routing) Critical operations swap (sequencing)	Partial solution	✓	-
Dauzere-Peres & Paulli (1997)	Concurrent	SPT with workload balancing	Insert	Inverse operator	x	-
Mastrolilli & Gabardella (2000)	Concurrent	Random	Insert	Inverse operator	✓	Forbidden moves
Fattahi, Mehrabad, & Jolai (2007)	Hierarchical	Random sequencing after routing to optimal machine with minimal processing time (MPT)	Swap (routing) Swap (sequencing)	Inverse operator	✓	Forbidden moves
Li, Pan, & Liang (2010)*	Hierarchical	Best result of a combination of routing & scheduling dispatching rules	Fully random or random extract from the machine with the most critical operations (routing) Swap or insert the head/tail into the critical block, or move a critical operation to the head or tail of its block (sequencing).	Inverse operator	✓	-
Li, Pan, Suganthan, & Chua (2011)	Hierarchical	Best result of a combination of routing & scheduling dispatching rules	Fully random, extract from machine with the most operations, or critical path extract (routing). Swap or insert the head/tail into the critical block, or move a critical operation to the head or tail of its block (sequencing).	Inverse operator	✓	-

* This method has a multi-objective function

2.3.4 Genetic algorithm

Based upon the evolutionary processes visible in nature, genetic algorithms (GA) work on a survival-of-the-fittest principle. A (usually randomly) generated set of solutions is selected, called the first generation. Every solution's quality, or fitness, is determined by calculating its objective function value. A subset of the generation is then selected randomly, with the chance of selection of a certain solution increasing proportionally to its fitness in some way (e.g. binary tournament, where two random solutions are selected and the best one survives, is the most frequently used selection method). A new generation with child solutions is then created based on this subset of parent solutions, by applying some genetic operators that combine parent solutions to generate children, and mutations that make some small random changes. This process then iterates for several times, all the while saving the best solution found so far. Figure 2.4 visualizes this process.

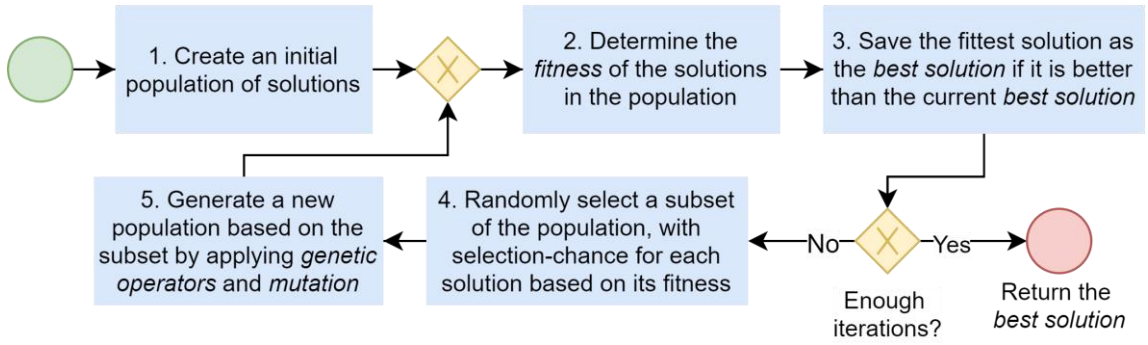

Figure 2.4 | The genetic algorithm

In this approach, solutions are encoded to a string format. Usually, this is through a job list, such as '1 2 2 1 2'. Here, each symbol refers to a job, and each job occurs in the list as many times at it has operations. The first occurrence refers to the first operation, etcetera. Other encodings are used as well. The string is called a chromosome, and each symbol is called a gene. Encoding solutions to a chromosome can be done in different ways. Table 2.4 shows an overview of various GA implementations for both the JSP and FJSP. For the FJSP, the mentioned job list is combined with a machine assignment list (where each operation is assigned a certain machine), in order to encode the solution for both the sequencing and routing subproblems.

Noticeable is that many later implementations involve a hybridization with local search or another metaheuristic, that are used to improve the individual solutions generated by the GA. The elitist strategy is often employed, where a certain number of the best solutions in a generation, get copied to the new generation. Initially, the fitness of solutions was often scaled, meaning the lowest objective value is

subtracted from each solution's value, resulting in a scale starting from 0 (or in the case of relative scaling, these new values are then divided by the maximum of the scaled values, resulting in a scale from 0 to 1). However, this strategy seems to have become obsolete in later years.

Table 2.4 | Overview of various genetic algorithm implementations

SOURCE	FJSP	SOLUTION GENERATION	SELECTION CHANCE FUNCTION	ENCODING	OPERATOR	MUTATION	ADDITIONAL STRATEGIES
Della Croce, Tadei, & Volta (1995)	x	Random	Not mentioned	Operation preference list per machine	Linear Order Crossover (LOX)	Swap	-
Dorndorf & Pesch (1995)	x	Random	Baker's algorithm (Baker J. , 1987)	List of dispatching rules to select next operation	Simple crossover	Swap	Scaling Elitist
Dorndorf & Pesch (1995)	x	Random	Baker's algorithm (Baker J. , 1987)	Sequence of machines to be scheduled using SB	Cycle crossover	-	Scaling Elitist
Bierwirth (1995)	x	Not mentioned	Probabilistic ranking	Job list	Generalized Order Crossover (GOX)	Insert	Replacement: A parent solution gets replaced by its child when the child performs at most 1% worse
Wang & Zheng (2001)	x	Random	Best + random	Job list	Simple crossover	Inversion	SA hybrid
Gonçalves, De Magalhães Mendes, & Resende (2005)	x	Not mentioned	Random	List of priorities for each operation, followed by a list of delay times for each operation	Parameterized uniform crossovers	Immigration mutation (replacing the worst solutions by randomly generated ones if they are better)	Elitist Local search hybrid
Kacem, Hammadi, & Borne (2002A)*	✓	Approach by Localization (AL) (routing) Random (sequencing)	Not mentioned	Matrix of machines × operations with start time and finish time tuples if assigned to that machine	Simple crossover	Smart machine assignment change (reducing workload or job processing time)	Schemata: Complex procedure for preselecting solutions based on a skeleton encoding generated by AL.
Gao, Gen, Sun, & Zhao (2007)*	✓	Random	Mixed sampling	Job list and machine assignment list	Enhanced order crossover (EOX)	Machine assignment change Swap with previous operation	Relative scaling Local search hybrid
Gao, Sun, & Gen (2008)*	✓	Not mentioned	Probabilistic ranking	Job list and machine assignment list	EOX Uniform crossover	Machine assignment change Sequence swap Immigration mutation	Elitist VNS hybrid

* This method has a multi-objective function

Table 2.4 | Overview of various genetic algorithm implementations **(ctd.)**

SOURCE	FJSP	SOLUTION GENERATION	SELECTION CHANCE FUNCTION	ENCODING	OPERATOR	MUTATION	ADDITIONAL STRATEGIES
Pezella, Morganti, & Ciaschetti (2008)	✓	AL (routing) Random/MRPT/MOR (sequencing)	Binary tournament	Task sequencing, with 3-tuples of job, machine and operation numbers	Smart machine assignment Precedence preserving order-based crossover (POX)	Precedence Preserving Shift mutation (PPS)	-
Wang, Gao, Zhang, & Shao (2010)*	✓	Random (sequencing) Binary tournament, 0.8 chance of selecting the lowest processing time machine (routing)	Binary tournament	Job list and machine assignment list	Improved POX Multipoint preservative crossover (MPX)	Double machine assignment change Insert (sequencing)	Immune system: The fitness of solutions is decreased proportionally to their similarity to other solutions Elitist
Zhang, Gao, & Shi (2011)	✓	Global Selection/Local Selection/Random Selection (routing) Not mentioned (sequencing)	Trinary tournament	Job list and machine assignment list	Two-point crossover Uniform crossover POX	Shortest processing time machine selection Insert (sequencing)	-
Chiang & Lin (2012)*	✓	AL (routing) Random/ MRPT /MOR (sequencing)	Binary tournament	Task sequencing, with 3-tuples of job, machine and operation numbers	Assignment crossover (ASX) POX	Move from max to min makespan machine	Local search hybrid
Chiang & Lin (2013)*	✓	Combination of five routing and three sequencing rules	Binary tournament	Task sequencing, with 3-tuples of job, machine and operation numbers	ASX POX	Random or smart machine assignment change Sequence swap or insert	-
Yuan & Xu (2015)*	✓	NSGA-II	Binary tournament	Job list and machine assignment list	ASX POX	Routing swap	Local search hybrid
Li & Gao (2016)	✓	Random	Binary tournament	Job list and machine assignment list	POX Job-based crossover (JBX)	Sequence swap Neighborhood mutation Machine assignment change	Elitist Tabu search hybrid

* This method has a multi-objective function

2.3.5 Particle swarm optimization

By essentially running several local search procedures (each starting from a different solution) in parallel and sharing information across all local searches, particle swarm optimization (PSO) is a proven effective method for solving the JSP. Each local search procedure is called a particle. All particles together form a swarm, referring to the phenomenon of information sharing in a swarm of birds this method is based on. Each particle moves from solution to solution based on a certain velocity. This velocity is determined based on a constant inertia factor, which makes sure the velocity does not change too rapidly, and the best solutions found both by the particle itself and over-all, called the cognition factor and social factor respectively. These three factors are combined to update a particle's velocity using equations such as the following (Xia & Wu, 2005):

$$V_{i,t+1} = W \cdot V_{it} + C_1 \cdot Rand \cdot (p_{it} - X_{it}) + C_2 \cdot rand \cdot (P_{gt} - X_{it})$$

where V_{it} refers to the velocity of particle i in iteration t, W to the inertia factor, C_1 and C_2 to the weight of the cognition and social factors, $Rand$ and $rand$ to random numbers between 0 and 1 (updated every iteration), p_{it} and P_{gt} to vectors of the local and global best solutions, and finally X_{it} to the particle's current position vector, which is updated by adding the velocity to it ($X_{i,t+1} = X_{it} + V_{i,t+1}$). Figure 2.5 shows an overview of the algorithm in its simplest form.

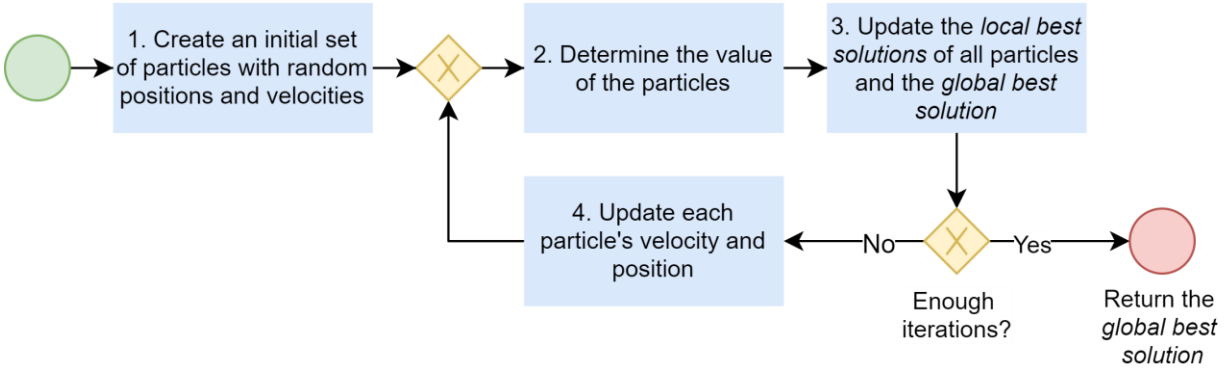

Figure 2.5 | The particle swarm optimization algorithm

Solutions must be encoded to a vector similarly as with GA in order to use p_{it}, P_{gt} and X_{it} as described above. The problem that this raises, is that velocity and positions are traditionally on a continuous scale in PSO. It is difficult to do this for JSP, as the sequence of operations is discrete. In order to cope with this, Sha & Hsu (2006) change the velocity function to a discrete scale, combined with positions represented as a preference list of jobs for each machine. Also notable is that they combine PSO with TS, applying it to each particle after updating.

Xia & Wu (2005) apply PSO to the FJSP, and therefore have to deal with the scaling issue differently. In their implementation, for each operation a preference list of machines is created, with machines that have a lower processing time first. The position of a particle is then described with an integer value for each operation, where a 1 refers to the machine with the highest preference, and higher numbers referring to machines further down the preference list. If velocity updating then results in non-integer values, these are rounded off to the nearest integer, resulting in a discrete position again. SA is used to determine sequencing for the routing obtained by PSO, optimizing a multi-objective function with a hierarchical approach.

Changing this hierarchical approach to a concurrent one, Moslehi & Mahnam (2011) add a list with real-valued priorities for each operation. With this extension, a particle's position contains information for solving both the routing and the sequencing subproblems. Moslehi & Mahnam (2011) hybridize the PSO with a local search procedure, to optimize the particles even further. These two adaptions make their implementation of PSO perform better than that of Xia & Wu (2005).

2.3.6 Artificial bee colony

Where PSO is based on the behavior of birds, the relatively new artificial bee colony algorithms (ABC) are inspired by the behavior of bees. Their functioning is similar to a local search hybrid GA with immigration mutation (where the worst solutions are replaced by randomly generated ones if these are better, as implemented by Wang, Zhou, Xu, Wang, & Liu (2012) for solving the FJSP). The difference lies in the local search, as ABC aims to do this more efficiently. This is done by generating a fixed total amount of neighbors every GA iteration, spread-out over all solutions in the population. The neighbors are distributed proportional to the quality of the solutions, meaning more neighbors will be generated based on a better solution. The reasoning behind this is that an already good solution is more probable to have even better

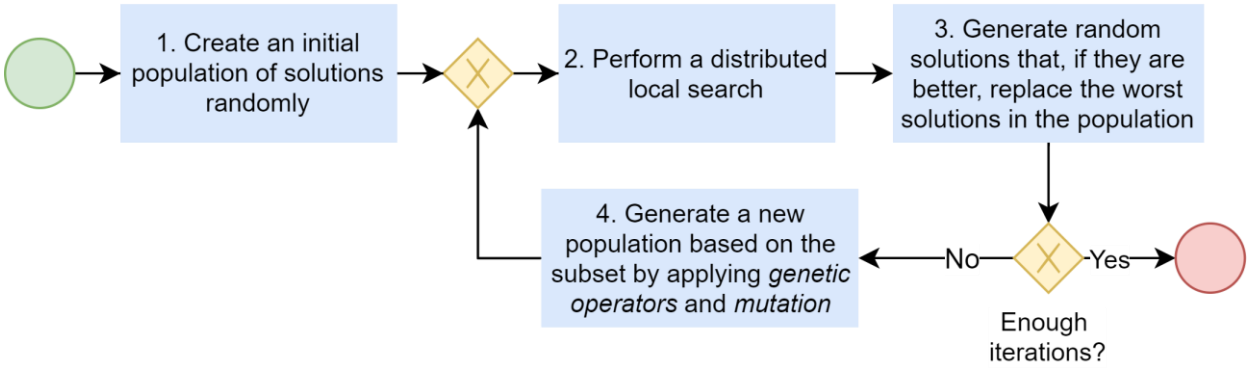

Figure 2.6 | The artificial bee colony algorithm

neighbors. Figure 2.6 shows an overview of the entire process, during which the best solution is always stored and updated if a newly generated solution is better. Step 1 and 4 are referred to as *employed bees*, step 2 as *onlooker bees* and step 3 as *scout bees*.

Li, Pan, & Gao (2011) create an ABC implementation for solving the FJSP with a multi-objective function. They also replace the GA functioning of employed bees with the same local search method as onlooker bees, which results in a simpler algorithm that still achieves the same level of results. The implementation of Wang, Zhou, & Liu (2012) also employs a local search for employed bees, and changes the distribution function of onlooker bees to a binary tournament. This results in similar results as previous implementations. ABC has also been applied to an FJSP with preventative maintenance (Li, Pan, & Tasgetiren, 2014).

2.3.7 Miscellaneous

Various other methods are applied to the JSP and FJSP. A genetic programming approach is presented by Tay & Ho (2008), which effectively solves a multi-objective FJSP with a self-learning dispatching rule, composed of multiple other dispatching rules (called terminals). The self-learning is done using a method similar to genetic algorithms, with the difference that each individual is a dispatching rule instead of a completed schedule. This type of approach is sometimes referred to as a hyperheuristic.

Wang, Wang, Zhou, & Liu (2012) propose a bi-population based estimation of distribution algorithm to solve the FJSP. This method repeatedly samples a probability model in order to simultaneously update the model and find new best solutions.

The harmony search procedure of Gao et al. (2014) solves the multi-objective FJSP with minimal makespan and the mean of earliness and tardiness as objectives. The method involves a list of the best solutions found so far. New solutions are, depending on chance, generated either entirely random or by slight adjustments to solutions currently in the list. If the new solution is better than the worst solution in the list, it replaces it. Repeating this process several times leads to results similar to other methods in a fraction of the time.

Table 2.5 lists miscellaneous methods that exist in literature as well but are not elaborated on in this research, as literature suggests these do not outperform the previously discussed methods.

Table 2.5 | Miscellaneous methods

METHOD	SOURCE	FJSP	MULTI-OBJECTIVE
Steepest decent with perturbation in heuristics and problem data	Storer, Wu, & Vaccari (1992)	x	x
Guided local Search with shifting bottleneck	Balas & Vazacopoulos (1998)	x	x
Fuzzy evolutionary	Kacem, Hammadi, & Borne (2002b)	✓	✓
Clonal selection	Ong, Tay, & Kwoh (2005)	✓	x
Knowledge-based ant colony optimization	Xing, Chen, Wang, Zhao, & Xiong (2010)	✓	x
Parallel variable neighborhood search	Yazdani, Amiri, & Zandieh (2010)	✓	x
Artificial immune algorithm	Bagheri, Zandieh, Mahdavi, & Yazdani (2010)	✓	x
Hybrid shuffled frog-leaping algorithm	Li, Pan, & Xie (2012)	✓	✓

2.4 Generation schemes

The methods elaborated on in Sections 2.2 and 2.3 result in a priority-sorted list of operations to be scheduled. Turning this list into a functional schedule, i.e. where every operation has a specific start and completion time, is an additional step distinct from the scheduling method, but also affects the schedule's performance. Kelley Jr. (1961) proposes two generation schemes that take this additional step, which are still the two mainly used approaches. These are the serial method and the parallel method, the latter of which was improved upon by Brooks (see Bedworth & Bailey, 1999).

2.4.1 Serial method

During the serial method, the operations are divided into three sets: the completed set, the decision set and the remaining set. The completed set contains all scheduled operations; the decision set contains the operations that are not scheduled, but all their predecessors are, so they are schedulable; and the remaining set contains all other operations.

The serial method iterates until all operations are scheduled, by taking the highest-priority operation from the decision set and moving it to the completed set, by scheduling it at its earliest possible start time, taking machine availability and the completion time of its predecessors into account. Operations that succeed the scheduled operation and have no other unscheduled predecessors are then moved to the decision set, and another iteration begins.

2.4.2 Parallel method

As the name suggests, the parallel method can schedule multiple operations per iteration. Each iteration n has a time t_n that represents the current schedule time (which starts at 0, and then increases after each iteration). The operations are divided into four, instead of three, sets, namely the completed, active,

decision and remaining set. The completed set contains all operations that are scheduled and completed at (or before) time t_n; the active set contains all operations that are scheduled but not completed yet at time t_n; the decision set contains all operations that could be scheduled at time t_n, taking both machine availability (i.e., its machine is unoccupied at time t_n) and predecessors' completion into account; and the remaining set contains all other operations. The steps of this method can be found in Figure 2.7.

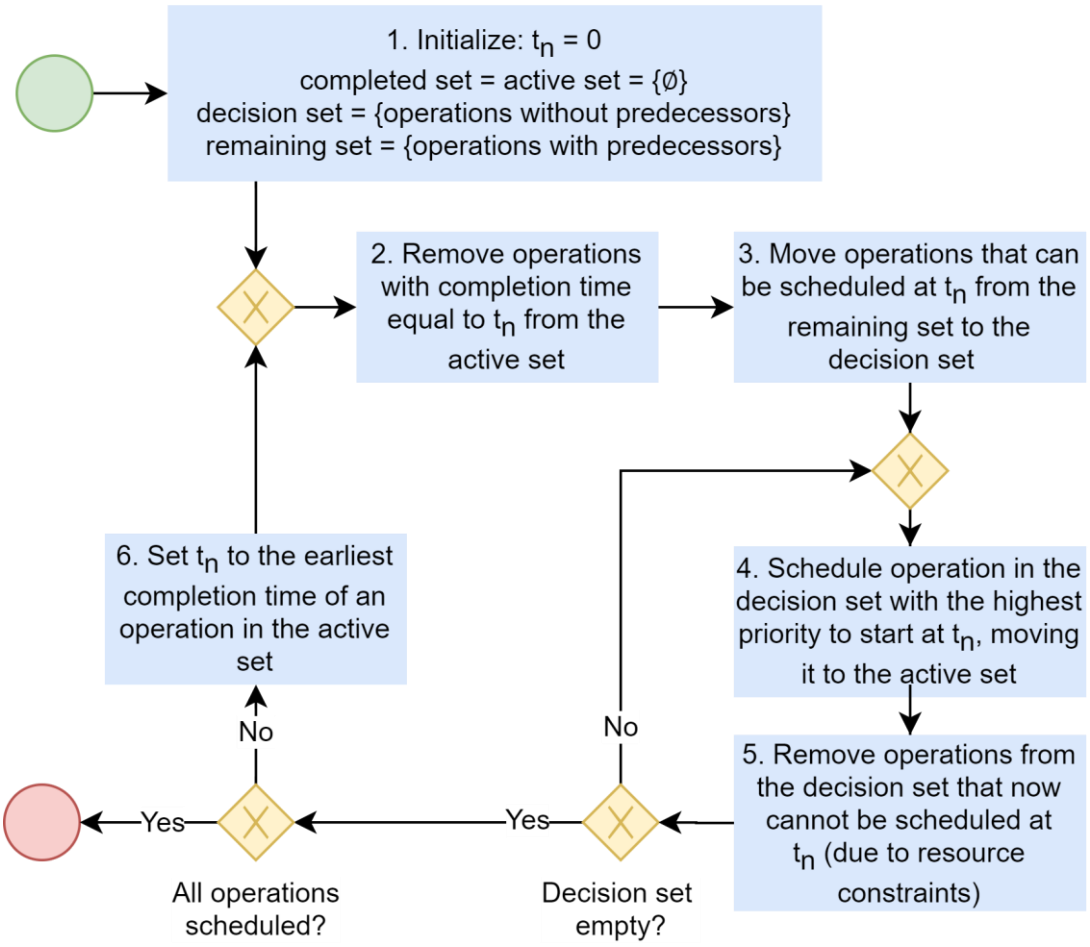

Figure 2.7 | The parallel method generation scheme

2.5 Conclusion

The main methods that are used in literature to solve the JSP can be divided into constructive methods (dispatching rules, shifting bottleneck, branch and bound) and improvement methods (GRASP, simulated annealing, tabu search, genetic algorithm, particle swarm optimization, artificial bee colony). Each method is implemented in multiple different ways, with different parameters and components, resulting in a wide range of options for solving the JSP. These methods result in a priority-sorted list of operations, which can be turned into a functional schedule using a serial or a parallel generation scheme.

3 Current situation

In order to know the starting situation we want to improve, this chapter establishes the JSP model and scheduling method currently used by Syze in Sections 3.1 and 3.2 respectively. Section 3.3 provides some insight into the customers that use the FSP.

3.1 Syze's Job Shop Problem model

Syze's software solves a specific adaption of the Flexible Job Shop Scheduling Problem, also including release dates, due dates and preventive maintenance, as explained in Section 2.1.

In this adaption, jobs consist of multiple operations that can have multiple preceding operations (that must be finished before the operation can start) and multiple succeeding operations (that can start only after the operation is finished). In practice, most jobs converge to a single final operation with which the job is completed, but this is not always the case. This structure is called tree precedence (Chen, Potts, & Woeginger, 1998).

Processing times are equal for all machines an operation could be scheduled on, and machines are grouped in work centers. Each operation must be scheduled on a machine of a specific work center. A machine can belong to multiple work centers.

Operations have queue, setup, processing and wait times in Syze's model. Setup times are fixed and independent of the operation previously scheduled on the machine. Only the setup and processing times require the use of machine capacity, but queue and wait times do have to be considered while scheduling preceding and succeeding operations. All four times require the operation to be available, meaning preceding operations are completed and its release date has passed. Queueing takes place before the setup starts, which represents the preparing of necessary materials. In other words, queue time could be defined as offline setup time, where setup time refers to online setup time. Wait time takes place after processing is done, for example because the products need to cool down before the next operation can take place. Operations are completed when the wait time is completed. Figure 3.1 gives an example of these times.

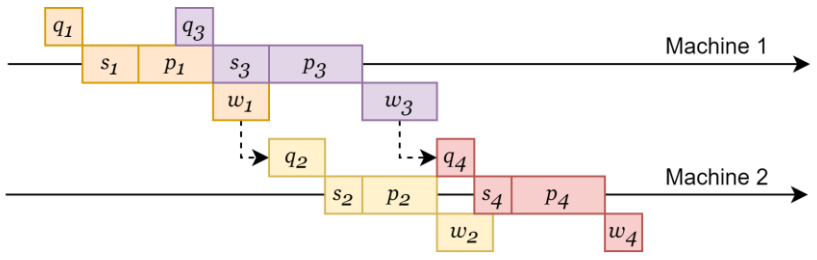

Figure 3.1 | Example of the queue (q_j), setup (s_j), processing (p_j) and wait (w_j) times of operation j.

The objective of the model is to minimize maximum lateness of the resulting schedule. Recall that lateness is the difference between the completion time (which refers to the end of the waiting time) and the due date, which is positive if a job is completed late and negative if a job is completed early.

3.1.1 Mathematical model

This section formally defines the previously explained model in order to bring more clarity to its workings.

3.1.1.1 Sets

$O = \{job\ operations\}$
$R = \{maintenance\ operations\}$
$J = \{operations\} \equiv O \cup R$
$F_j = \{directly\ preceding\ operations\ of\ operation\ j \in O\} \subseteq O$
$I = \{machines\}$
$M_j = \{machines\ operation\ j \in J\ can\ be\ processed\ on\} \subseteq I \quad (|M_j| = 1\ for\ j \in R)$

3.1.1.2 Parameters

$r_j = release\ date\ of\ operation\ j \in J$ $\qquad$ $s_j = setup\ time\ of\ operation\ j \in O$
$d_j = due\ date\ of\ operation\ j \in J$ $\qquad$ $p_j = processing\ time\ of\ operation\ j \in O$
$q_j = queue\ time\ of\ operation\ j \in O$ $\qquad$ $w_j = wait\ time\ of\ operation\ j \in O$

3.1.1.3 Variables

$$X_{i,j,k} = \begin{cases} 1, if\ operation\ j \in J\ is\ the\ k^{th}\ operation\ scheduled\ on\ machine\ i \in I \\ 0, otherwise \end{cases}$$
$S_j = start\ time\ of\ operation\ j \in J\ on\ a\ machine, excluding\ queue\ time$
$E_j = end\ time\ of\ operation\ j \in J\ on\ a\ machine, excluding\ wait\ time$

3.1.1.4 Constraints

$$X_{i,j,k} = 0, \qquad \forall j \in J, i \notin M_j, k \tag{1}$$
$$\sum_i \sum_k X_{i,j,k} = 1, \qquad \forall j \in J \tag{2}$$
$$\sum_j X_{i,j,k} \leq \sum_j X_{i,j,k-1}, \qquad \forall i \in I, k \mid k > 1 \tag{3}$$
$$S_j \geq E_g + w_g + q_j, \qquad \forall j \in O, g \in F_j \tag{4}$$
$$S_j \geq r_j + q_j, \qquad \forall j \in O \tag{5}$$
$$S_j \geq X_{i,j,k} \cdot X_{i,g,k-1} \cdot E_g, \qquad \forall i \in I, (j,g) \in J, k \tag{6}$$
$$E_j \geq S_j + s_j + p_j, \qquad \forall j \in J \tag{7}$$
$$S_j = r_j, \qquad \forall j \in R \tag{8}$$
$$E_j = d_j, \qquad \forall j \in R \tag{9}$$
$$X_{i,j,k} \in \{0,1\}, \qquad \forall i \in I, j \in J, k \tag{10}$$
$$S_j, E_J \geq 0, \qquad \forall j \in J \tag{11}$$

Constraint 1 ensures that operations are feasibly assigned to a machine. Constraint 2 ensures that each operation is scheduled. Constraint 3 ensures continuity in schedules, without gaps in a machine's sequence (i.e., no fifth job can be scheduled on a machine without there being a fourth job). Constraints

4 to 6 set lower bounds on the start time of an operation, relating to precedence relations, release dates and the machine's previous operation, respectively. Constraint 7 sets the end time of operation to their start time plus processing and setup times. Constraints 8 and 9 set the start and end times of maintenance operations, which are given as input and cannot be moved. Constraints 10 and 11 give variables their proper types.

3.1.2 Objective function

The objective function is to minimize the maximum lateness of jobs, determined by looking at the lateness of operations that are not a predecessor of another operation (and therefore the final operation of a job): $\min \max_{j \in O | j \notin F_g \forall g \in O} L_j$, where $L_j = E_j + w_j - d_j$.

3.2 Scheduling method

The FPS uses an Earliest Operation Due Date (ODD) dispatching rule. This method schedules by prioritizing operations with the earliest due date. As stated in the previous section, jobs have a due date. The due date of the last operation of a job is set equal to the job's due date. The due dates for all other operations are determined by moving backwards in the precedence chain, calculating the maximum completion date with which all succeeding operations can theoretically still finish in time, meaning each operation's due date is smaller or equal to the minimum of its successors' due date minus its queue, setup, processing and wait times.

Figure 3.2 shows the steps of ODD as implemented by Syze, which implicitly applies a serial generation scheme (recall Section 2.4.1). In a simple version of the JSP, with one single machine (where

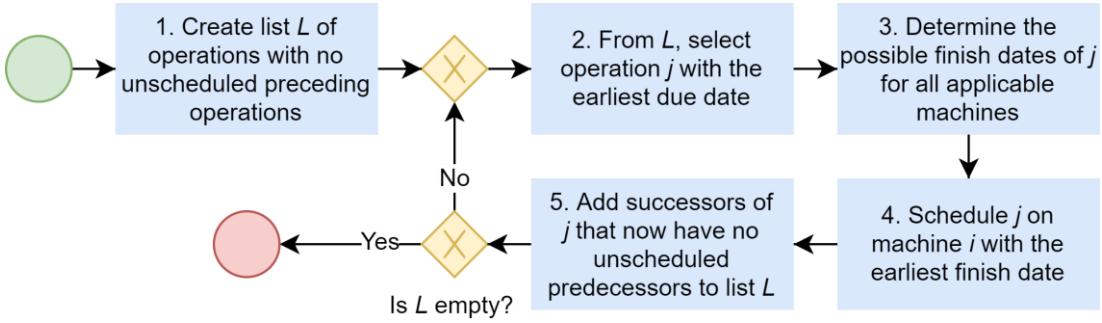

Figure 3.2 | The Earliest Operation Due Date algorithm used to solve the JSP

jobs then consist of a single operation, that has to be completed on that machine) and preventive maintenance as the only extension, ODD optimally minimizes the maximum lateness (Graves & Lee, 1999). However, in JSPs with multiple machines, ODD is outperformed by other methods (Jayamohan &

Rajendran, 2000). Therefore, implementing additional methods in the FPS is likely to lead to better results for Syze's customers.

3.3 Customers

Syze's customer base is diverse, with customers in sectors ranging from automotive to food. This diversity results in the fact that not all customers make use of all constraints mentioned in Section 3.1.1. For this research, we select six customers belonging to different sectors of industry for review. For anonymization, these customers are referred to by their sector, namely Food, Fridges, Kitchens, Power tools, Tannery and Telescopes. We have the data of only a single job shop scheduling instance available for each customer, so we are unable to fit the data's attributes to statistical distributions. Therefore, we instead show some general statistical figures like averages in Table 3.1.

Two out of the six customers use preventive maintenance (downtime), one uses queue times, four use wait times (the Tannery even uses the same wait time for each operation), four use setup times and four use release dates. All customers use work centers and due dates.

Three of the six customers do not use the tree precedence structures mentioned in Section 3.1, instead using a chain structure as in the classis JSP, where each operation has at most one predecessor and at most one successor. For one of these three, Kitchens, all jobs consist of a single operation, meaning there are no precedence relationships at all. When a dataset does include tree precedence relationships, it does so only for a small percentage of jobs; about 1.5% to 3%.

The schedule horizon differs strongly, ranging from two months to three years. Especially the latter is unexpected for an operational schedule, which is meant for the foreseeable future. This unexpected horizon is therefore even more interesting, as it results in a schedule size that is rare in literature.

Worthy of note is that machines can be a part of multiple work centers, which explains why, in the two cases of Telescopes and Power tools customers, the number of work centers is higher than the number of machines. In the other datasets, machines do belong to a single work center.

The Telescopes and Power tools customers stand out because of the high percentage of operations with a single-machine work center, meaning it only needs to be sequenced and not assigned to a machine: 75.3% for Telescopes and 98.9% for Power tools.

Table 3.1 | Attributes of customer data sets

		Food	Fridges	Kitchens	Power tools	Tannery	Telescopes
Schedule time horizon (days)		105	54	70	1145	100	1087
Number of jobs		155	4566	1140	9510	286	4325
Mean job interarrival time (hours)		16.12	0.28	1.45	2.89	8.31	6.02
Mean operations per job		2.1	1.15	1.00	5.74	3.79	15.01
Jobs with a single operation (%)		36.1	95.1	100	11.0	74.8	0.07
Chain precedence jobs (%)		100	97.2	100	98.5	100	97.8
Maximum successors per operation		1	4	0	7	1	3
Maximum predecessors per operation		1	9	0	15	1	3
Number of work centers		10	47	5	250	21	78
Number of machines		21	206	12	245	45	70
Maximum number of machines per work center		9	34	8	2	7	8
Number of downtimes		-	-	-	219	-	101
Mean downtime interarrival time (hours)		-	-	-	125.48	-	258.30
Average length of downtime (days)		-	-	-	462.79	-	13.62
Number of downtimes per machine	*maximum*	-	-	-	1	-	26
	mode	-	-	-	1	-	1
Queue time (hours)	*average*	-	0.69	-	-	-	-
	maximum	-	2.00	-	-	-	-
Setup time (hours)	*average*	-	0.41	0.41	0.68	-	0.71
	maximum	-	0.83	0.42	6.00	-	3.00
Processing time (hours)	*average*	59.45	1.97	4.08	13.44	2.59	10.76
	maximum	1370.88	11.61	137.28	3336.00	24.00	19200.00
Wait time (hours)	*average*	3.64	-	-	31.26	0.25	24.87
	maximum	12.00	-	-	336.00	0.25	240.00
Slack (days)	*minimum*	0	-0.51	-6.92	-22.89	0	-790.86
	maximum	103.88	24.45	4.77	13.03	97.11	456.92
Release dates		no	yes	yes	yes	no	yes

3.4 Conclusion

Syze's implementation of the JSP is a flexible job shop problem with equal processing times on all machines. It is also extended with preventive maintenance, forest precedence structure, release dates and queue, setup and wait times. However, not all customers make use of all these extensions. The ODD dispatching rule is used to solve this FJSP model that has minimal maximum lateness as the objective function.

4 Experimental design

This chapter sets up all needed components for our experiments. First it selects cases that are relevant for Syze's FPS tool in Section 4.1. Section 4.2 then explains how we create multiple instances of scheduling data for the selected cases. Section 4.3 selects the methods to be tested, and Section 4.4 explains how these methods are implemented.

4.1 Selection of cases

In order to be able to recommend customers the usage of a certain method for solving the JSP, we need to know what attributes influence the performance of the scheduling methods. In our experiments, we therefore want to apply the methods to a broad range of JSP cases with differing attributes. The initial cases we select are the six customer cases of Section 3.3, as these are the only data-based cases we have available.

Based on the outcome of these initial cases, we select more cases, in order to test some hypotheses to explain why the initial results are what they are. However, as mentioned in Section 3.3, the attributes of jobs, operations and downtime do not fit any specific statistical distribution. We are therefore dependent on empirical distributions to create realistic data instances for each case. Empirical distributions determine the value of a random variable by randomly selecting a value of that variable in a sample dataset (which are the customer datasets, in our case).

Using these distributions therefore limits us to experiment with cases that are equal or similar to the six customer datasets. Similar cases can be created either by 'switching off' an extension (i.e., removing all preventive maintenance, or all wait times), by scaling some attribute (i.e., multiplying the job interarrival times by 0.66, therefore creating 1.5 times the number of jobs in the same schedule horizon), or by transplanting an extension from one dataset to another (i.e., applying the wait times present in one dataset to another dataset).

4.2 Instance generation

The attributes of customer data in Section 3.3 are based on a single instance of a job shop schedule per customer, referred to as the initial datasets. In order to reliably quantify the performance of the JSP scheduling methods, we require more than one instance, as this increases the reliability of the results. Based on some initial experiments, we find five instances per case enough to be able to reliably compare the performance of several methods.

By applying randomness to the initial datasets (using mostly empirical distributions), we create several new instances per case to be used for quantifying the performance of methods. The remainder of

this section describes how we generate the new instances, split into Section 4.2.1 for jobs and their operations and Section 4.2.2 for machine downtime.

4.2.1 Jobs

Using an empirical distribution for arrival times, we determine either the release date (starting at time 0 for the first job) or, if the case does not include the usage of release dates, the due date of the next job. This repeats until the required number of jobs is generated. This number differs per case.

If the case does include the usage of release dates, the due date of the job is generated by determining the minimal time required for completing a job. This minimal time is calculated by determining the longest path between a start and an end operation of the job, where the 'distance' between a predecessor and a successor is represented by the sum of the predecessor's queue, setup, processing and wait times.

As we observed in the customer datasets that a high percentage of jobs consist of a single operation, we use that percentage as the probability that the number of operations of a job will be one. If this happens to not be the case for a job, the number of operations and their precedence structure is generated, using different processes for tree and chain structures (recall Section 3.1).

When the case includes tree precedence structures, we use the actual percentage of jobs with tree structures in the initial dataset as the chance of a job having a tree structure in the generated instance. Recall that only a small percentage of jobs actually have a tree structure, and most have a chain structure (see Section 3.3).

For chain structures, we determine the number of operations using a uniform distribution instead with the minimum and maximum observed operations per job as boundaries. Using a uniform distribution is a simplification with regards to how this attribute is distributed in the initial datasets. We make this simplification in order to save time that preparing data for an empirical distribution would cost.

If a job is determined to have a tree precedence structure, the number of operations and the precedence structure is determined using a method similar to the method used by Hurink, Kok, Paulus, & Schutten (2011). In it, following the same reasoning as with chain structured jobs, uniform distributions are used for the total number of operations, the number of start operations (without predecessors) and the number of end operations (without successors). Each uniform distribution's boundaries are set to the minimum and maximum value observed in the initial dataset. Each operation's number of predecessors and successors (if this number is not already 0 due to the operation being the first or final of its job) is set

using an empirical distribution. For further details, the reader is referred to the pseudo-code in Appendix 8.1.

The job's due date is determined using the minimal time required for completing the job, multiplied with a slack factor. This slack factor is determined using another empirical distribution and represents the percentual difference between the minimal time required and the time available between the job's release and due dates.

For each operation that is generated in the procedures described above, its attributes (queue, setup, processing & wait times, and the work center it should be processed on) are generated using empirical distributions. Note that this means that the number of machines and how these are divided into work centers is the same as in the initial datasets.

4.2.2 Downtime

Using an empirical distribution for the time between downtimes, we determine the start time of the downtimes, starting at time 0 plus a number generated using that empirical distribution for the first downtime. Then, the downtime duration is determined using an empirical distribution as well. Finally, one machines is randomly selected out of all machines. If the case has at most one downtime per machine (as with the Power Tools customer), the same machine cannot be selected twice for downtime in the instance. This repeats until the required number of downtimes is generated. This number differs per case.

4.3 Selection of scheduling methods

In order to increase customers' acceptance of new algorithms for scheduling, Syze has two criteria that must be met by the algorithms. These are repeatability, meaning that running the same algorithm twice should get the same results, and explainability, meaning the concept of the algorithm is understandable to a nonexpert and not overly complex. Any method, including those that depend on randomness, can be made repeatable, i.e. by using seed values for random number generation. The first criterion therefore does not lead to discounting any of the methods discussed in Chapter 2.

We deem the simulated annealing, genetic algorithm, particle swarm optimization, artificial bee colony, estimation of distribution and harmony search methods to be too complex to meet the explainability requirement. SA is complex as it requires the tuning of four different settings, the optimal values of which depend on the specific case without a way to determine these values beforehand (i.e. based on some formula), instead requiring this to be done manually. Although GA is not in itself a complex method, the multitude of different ways to implement it, as shown by Table 2.4, do make it complex.

Lastly, the branch and bound method is discounted as it is a method for solving optimally, which takes an infeasible amount of time for the size of our instances. The remaining methods that do meet all requirements are then dispatching rules (DRs), shifting bottleneck (SB), GRASP, tabu search (TS) and genetic programming (GP).

Except for DRs, these methods all use an objective function to determine the next step to take (i.e. a local search where the solution is slightly changed if that improves the objective function). Many possible objective functions exist (recall Section 2.1.3), and Syze's scheduler currently uses maximum lateness. We deem the average tardiness a better objective function, as improving an average improves all jobs, whereas improving a maximum improves only the current worst job(s). The advantage of average tardiness over average lateness is that this does not reward finishing a job early, which would result in cost-increase in practice (more warehousing space required for finished products, opportunity costs of material usage, etcetera).

DRs and SB are constructive methods, which solve the JSP once and therefore have an immutable time duration. The duration of GRASP, TS and GP, which are improvement methods, is a parameter to be chosen, where more iterations lead to better results. For these three methods, we examine their results depending on how much time is spent. These improving methods include an initial constructive method (which they then improve upon), and therefore require more computation time than constructive methods do. As some of Syze's customers want to be able to create schedules as fast as possible, others have more computation time available and can therefore also use improvement methods. We therefore discuss both separately. Due to the limited time available for this research, we set the upper bound of the computation time to either two hours, or the time required for a single iteration if that time is reasonably close to two hours.

4.4 Implementation of methods

As mentioned in Section 4.3, we select Dispatching Rules (DRs), Shifting Bottleneck (SB), Tabu Search (TS) and Genetic Programming (GP) as relevant methods for Syze's FPS. We apply these methods to the cases determined by Section 4.1. In order to provide comparable results, we use the same machine for all methods, with 16 GB of RAM and an Intel Core i7-4710MQ quadcore processor with a clock speed of 2.5 GHz. We implement the methods using the Python programming language.

Each method's implementation is based on a specific implementation in literature, though some parameters are tweaked in order to cope with the larger scale of our cases compared to cases used in literature. This section elaborates what changes were made for each of the selected methods.

4.4.1 Dispatching Rules

We implement the ten dispatching rules that Table 2.1 and Table 2.2 establish, as these are deemed well-performing by literature. For the ODD DR, we determine each operation's due date in the same manner as currently implemented in the FPS (recall Section 3.2), by moving backwards in the precedence chain.

The dispatching rules determine the sequence of operations, still requiring the assignment of operations to be established. This is done for each operation individually, where the machine that leads to the earliest completion time of the operation is selected.

4.4.2 Shifting Bottleneck

We adopt the SB implementation of Kalma (2020), including his recommendation to use the SRPT dispatching rule to solve the subproblems. In preliminary experiments, we find that, of all six initial cases, SB can only solve the Food case in reasonable time. Presumably, this is because the largeness and complexity of other cases leads to an increase in the time required for solving the subproblems in SB, even when using SRPT. As these subproblems are repeatedly resolved, the time increase of a subproblem has significant effects on the total time required. Because of this observation, we will only apply SB to cases similar in size as the Food case.

4.4.3 Greedy random adaptive search procedure

We use the GRASP described by Aiex, Binato, & Resende (2003) for our implementation. For the local search part, we have to use different neighborhood operators, as we use a different objective function and have an FJSP, which has to assign machines to operations as well. The assignment is initially done in the same manner as with our DRs implementation.

During the local search, we use a critical insert as the assignment neighborhood operator, meaning a critical operation is inserted into another machine's sequence. The sequence neighborhood operator we use is a critical swap.

Critical operations here do not refer to the changed operation(s) being part of the critical path, but to the changed operation(s) being one of the top 50 most tardy operations, as this is more in line with our goal of minimized average tardiness.

4.4.4 Tabu Search

We implement a concurrent Tabu Search, meaning sequence and assignment changes are evaluated simultaneously (recall Section 2.3.3). The same neighborhood operators are used as with GRASP. The starting solution for the TS is the best solution generated by a DR. The tabu list length is increased over

time in the same manner as in Li, Pan, & Liang (2010), starting at a length of half the to be scheduled operations, and ending at a length equal to the number of to be scheduled operations. We also implement an aspiration criterion, meaning a tabu move is still allowed if it leads to a better objective function value than found so far.

4.4.5 Genetic Programming

Our implementation of GP (recall Section 2.3.7) is based on Tay & Ho (2008), with some small changes. First, the Current Time terminal they use is impractical for our large cases, and as it is not used in any of their best results, we choose to not use it in our implementation. Furthermore, they use both a swap and a shrink mutation. As the Python genetic programming library (called DEAP) does not include a swap mutation operator, we choose not to use it and instead increase the probability of shrink mutation.

Tay & Ho (2008) use the following parameters for their GP: a population size of 100, 200 generations, and selection of the 5 best individuals in each population to copy into the new generation's population. As these values lead to too long computation times for some of our instances, we set the population size to a value which allows for at least two generations (based on the DRs' average computation time) with a maximum of 100. We then select the 5% best individuals. The number of generations is not set in advance, as we instead we keep iterating until the maximum allowed computation time is reached.

4.4.6 Generation scheme

As explained by Section 2.4, the mentioned methods at some point have to translate a sequence of operations into a functional schedule by determining the start and end times of each operation using either a serial or a parallel generation scheme, where the latter gives better results but also requires more computational time. In preliminary experiments we found that the computational time becomes infeasible when applying the parallel generation scheme to larger cases. Presumably, this is because larger cases lead to larger sets to be filtered and searched in, which increases computation times exponentially. On the other hand, serial generation moves from a single operation to the next, leading to a linear increase in computation time. We therefore choose to use a serial generation and implement the version of Bedworth & Bailey (1999).

4.5 Conclusion

Using empirical distributions, we generate five instances for each of the initial customer cases of Section 3.3, to which we apply Dispatching Rules, Shifting Bottleneck, GRASP, Tabu Search and Genetic Programming.

5 Results

This chapter discusses the results of applying the selected methods to the selected cases. Section 5.1 describes the results of the initial cases, and Section 5.2 attempts to answer some questions raised by those initial results using some additional cases, adapted from the initial ones. Section 5.3 concludes by summarizing all results per method.

5.1 Initial results

We first explain how the results are shown. As stated by Section 4.3, we use average tardiness as our objective function. We call the method that achieves the lowest average tardiness for a given instance the "winner" of that instance. In order to easily compare results over all instances, cases and methods, we state the normalized values for average tardiness, as well as for one other key performance indicator (KPI), namely maximum tardiness.

Normalization is done by dividing each KPI value with the corresponding value of the winner of the same instance. A drawback of normalizing in this manner is that when the winner's value lies close to or below zero, other values become either extremely large or negative, respectively. However, neither of those things happens in our results, so we can safely use this methodology.

For the sake of conciseness, we here show the results of only the best methods per case, as these are most interesting to us. We refer the reader to Appendix 8.2 for the results of all methods per case. In the names of the improvement methods (e.g. "TS-120"), the number refers to the computation time in minutes corresponding to the results. This way, we can compare the results of these methods over time.

Each case's results are shown using a boxplot and a table. The boxplots show the spread of the normalized values of average tardiness over each instance. The tables show the averages over all instances of the normalized KPIs for the same methods and cases. In these tables, the number of times (out of the 5 instances) the method is a "winner", as defined above, is given by the "wins" row. Sometimes, multiple methods achieve the same (winning) schedule, which explains why the sum of the "wins" row might be more than 5. For maximum tardiness and the percentage of late jobs, the number of times the method achieves a better value for those KPI's than the winner is given in the row below the respective KPI, called the number of times it is "better than winner". Values of 0 in these rows are replaced by dashes ("-"), as this makes the tables less cluttered and chaotic.

For GRASP, the number of iterations given in the tables refers to the number of times a locally optimal schedule is generated. It should be noted that the last of these might not be locally optimal, as the local search is stopped when the computation time constraint is met. For TS, the number of iterations

refers to the number of search steps taken. For GP, it refers to the number of times a new generation is evaluated.

We decide to use a significance level of 0.05, e.g. meaning an improvement of more than 5% is deemed as a significant improvement. The following sections present the results of each initial case, one by one.

5.1.1 Food

In the Food case, SPT far outperforms the other DRs in terms of average tardiness: the second-best DR (namely SRPT, not shown here, see Appendix 8.2.1) has an 11 times higher average tardiness. However, SB outperforms SPT with a 14% lower average tardiness and is therefore the best constructive method in this case. This performance increase does come at the cost of computation time: SB uses 14 minutes of computation time, whereas SPT only requires 2 seconds. This makes sense, as SB resolves its subproblems multiple times using a DR, instead of merely applying a DR once (recall Section 4.4.2).

As can be seen in Figure 5.1 and accompanying Table 5.1, TS is the best-performing improvement method. Starting from the SPT result of an average normalized average tardiness of 1.27, TS lowers it to 1.05 in an average of 1 minute. That is already better than the result of GP after 120 minutes, not to mention GRASP, which only reaches a normalized average tardiness of 10.46.

In only 2 of the 5 instances, TS keeps finding improvements for 120 minutes. In the other 3 instances it stops finding improvements after 20, 24 and 51 minutes, which is why the number of iterations does not increase linearly over time in Table 5.1.

Interestingly, SB does have a lower percentage of late jobs than TS (an average difference of 1.67 percentage point, corresponding to 2.6 jobs on average), though its average tardiness is 12% higher on average. Also, the maximum tardiness of TS decreases over time, albeit an insignificant decrease of less than 1%. This does, however, show that it is possible that minimizing average tardiness leads to an increase in maximum tardiness.

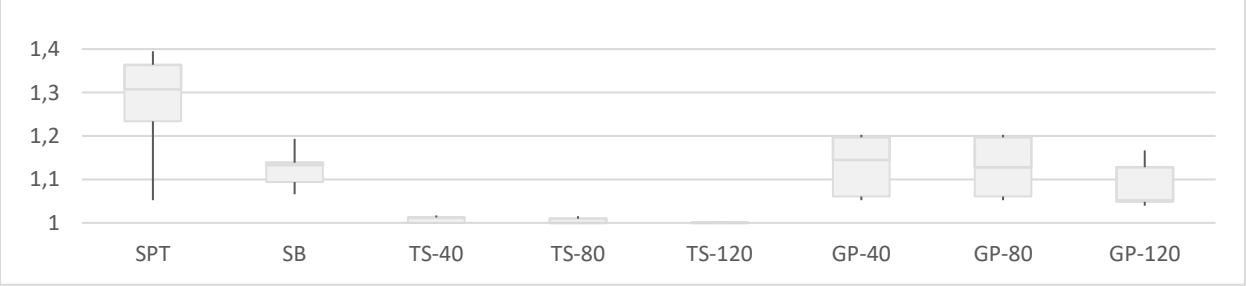

Figure 5.1 | Boxplot of normalized average tardiness for the Food case

Table 5.1 | Results for the Food case

	SPT	SB	TS-40	TS-80	TS-120	GP-40	GP-80	GP-120
avg. tardiness (hours)	288.81	245.86	221.86	221.56	220.81	255.17	254.20	246.45
norm. avg. tardiness	1.27	1.12	1.01	1.00	1.00	1.13	1.13	1.09
wins	0	0	2	3	5	0	0	0
max. tardiness	1.22	1.15	1.00	1.00	1.00	1.08	1.08	1.07
better than winner	-	2	2	1	-	-	-	1
late jobs (%)	19.49	16.41	18.08	18.08	18.08	18.72	19.23	18.59
better than winner	-	4	1	1	-	1	-	2
duration (min.)	0.04	14.23	40	80	120	40	80	120
no. iterations	-	-	88.4	108.6	125.6	24.8	50.4	75.2

5.1.2 Fridges

Of the constructive methods, ODD outperforms all others, with the second-best performer (SCR, not shown here, see Appendix 8.2.2) performing 53% worse in terms of average tardiness. TS does not manage to significantly improve ODD results, even though it performs many improvement iterations, as can be seen in Figure 5.2 and accompanying Table 5.2.

GRASP and GP have an insignificant difference in average tardiness of 4.4%, and we can therefore not deem either the winner based on average tardiness. However, GRASP requires 20 minutes less computation time than GP to get to the similar results, and therefore does have an edge over GP. It is interesting that both GRASP and GP can perform only a single iteration in their computation time, and still outperform other methods, while the strength of these methods in theory lie in their ability to improve using multiple iterations. It is unclear to us why this is the case.

Though GRASP and GP are the (tied) winners for this case in terms of average tardiness, they respectively require 120 and 140 minutes of processing time, meaning GRASP achieves its results faster. We therefore prefer that method for this case.

5.1.3 Kitchens

As can be seen in Figure 5.3 and accompanying Table 5.3, there is no large difference in average tardiness between the best methods for the Kitchens case. The EDD, ODD and SCR case achieve the same results. This is logical for EDD and ODD as these DRs become equivalent when all jobs consist of one operation, which is true in the Kitchens case. Why SCR achieves the same result as well is unclear to us.

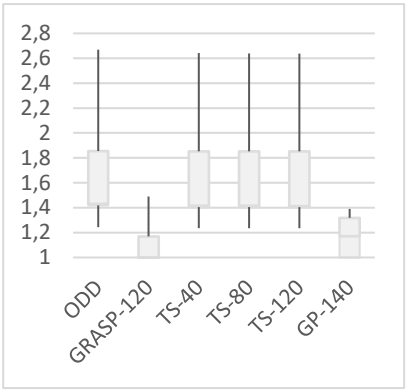

Figure 5.2 | Boxplot of normalized average tardiness for the Fridges case

Table 5.2 | Results for the Fridges case

	ODD	GRASP-120	TS-40	TS-80	TS-120	GP-140
avg. tardiness (hours)	80.34	51.41	79.75	79.69	79.67	49.81
norm. avg. tardiness	1.72	1.13	1.71	1.71	1.71	1.18
wins	0	3	0	0	0	2
max. tardiness	0.48	0.85	0.48	0.48	0.48	2.26
better than winner	5	2	5	5	5	-
late jobs (%)	98.91	98.36	98.91	98.91	98.91	98.55
better than winner	-	2	-	-	-	-
duration (min.)	4.93	120	40	80	120	140
no. iterations	-	1	24	48.8	70.6	1

GP achieves similar results to the mentioned DRs, even performing better in one instance, albeit insignificantly (less than 1% difference). TS performs a single improvement iteration in one minute on the DRs result, slightly improving results. After this, it does not find a better schedule for the remainder of two hours, leading us to believe is easy to achieve a (near-)optimal schedule in this case, which is probably because of its simple job structure wherein each job consists of a single operation.

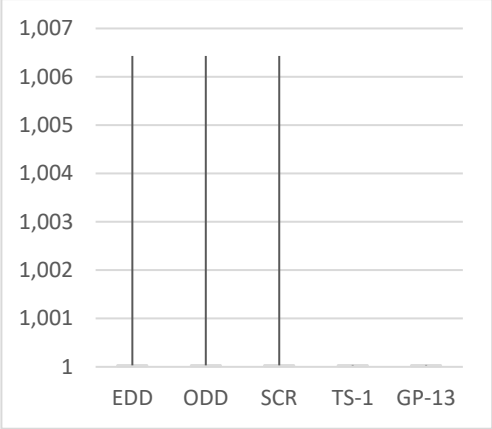

Figure 5.3 | Boxplot of normalized average tardiness for the Kitchens case

Table 5.3 | Results for the Kitchens case

	EDD	ODD	SCR	TS-1	GP-13
avg. tardiness (hours)	0.15	0.15	0.15	0.15	0.15
norm. avg. tardiness	1.00	1.00	1.00	1.00	1.00
wins	2	2	2	5	3
max. tardiness	1.00	1.00	1.00	1.00	1.00
better than winner	-	-	-	-	-
late jobs (%)	2.86	2.86	2.86	2.84	2.84
better than winner	2	2	2	-	1
duration (min.)	0.32	0.42	0.39	1	13
no. iterations	-	-	-	1	1

5.1.4 Power Tools

Figure 5.4 and accompanying Table 5.4 show the results of the Power Tools case. We find ODD and LRPT to perform similarly well, without a significant difference in average tardiness. However, Figure 5.4 does show that LRPT results in a lower spread in average tardiness and Table 5.4 shows that the maximum tardiness of LRPT is 31% lower than ODD's on average. We therefore prefer LRPT as a constructive method for the Power Tools case.

We find that TS is unable to achieve a significant improvement from the DRs (only 2% on average), likely because it is unable to perform a large amount of iterations. GRASP is a clear winner in all 5 instances. GP gets the second-best average tardiness, but it is on average 5 times higher than GRASPs average tardiness and it takes 100 more minutes to achieve that result.

It is unclear to us why GRASP performs so well, especially because only 1 iteration of GRASP is performed within the time limit (with the exception of one instance, where 2 iterations are performed), while the nature of improvement methods is that they achieve better results after iterating multiple times.

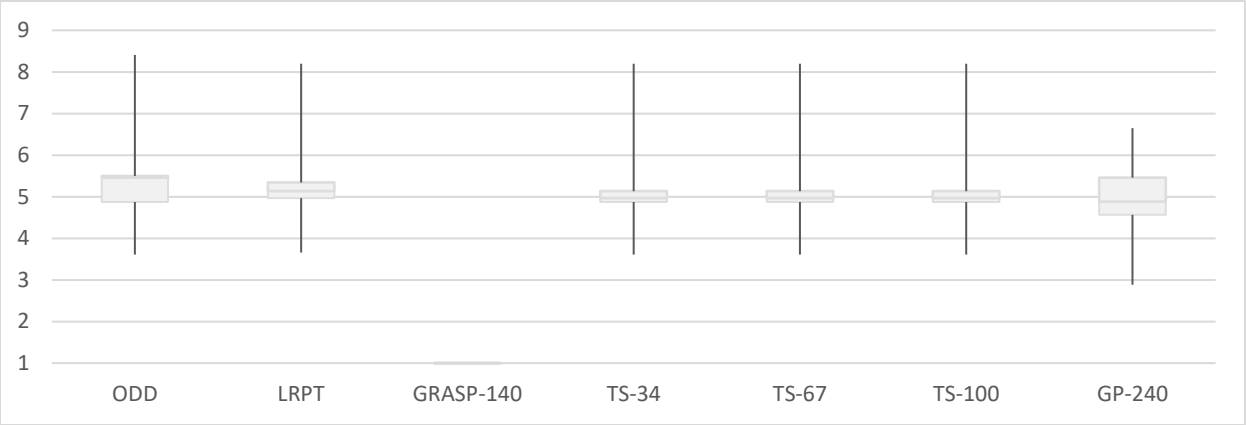

Figure 5.4 | Boxplot of normalized average tardiness for the Power Tools case

Table 5.4 | Results for the Power Tools case

	ODD	LRPT	GRASP-140	TS-34	TS-67	TS-100	GP-240
avg. tardiness (hours)	72948.84	123680.07	246953.63	386632.67	563597.99	693317.57	460705.06
norm. avg. tardiness	5.57	5.46	1.00	5.36	5.36	5.36	4.89
wins	0	0	5	0	0	0	0
max. tardiness	2.30	1.59	1.00	1.52	1.83	1.83	2.52
better than winner	-	-	-	1	-	-	-
late jobs (%)	99.98	99.97	99.65	99.98	99.98	99.98	99.98
better than winner	-	-	-	1	-	-	-
duration (min.)	19.16	17.96	140	34	67	100	240
no. iterations	-	-	1.2	1	2	3	1

5.1.5 Tannery

Of the constructive methods, ODD all others in the Tannery case. SPT (not shown here, see Appendix 8.2.5) takes second place with regards to average tardiness, which is for SPT 66 times higher than for ODD on average.

GRASP is not mentioned in Figure 5.5 and accompanying Table 5.5, as it performs 50 times worse than GP and TS on average. GP and TS perform similarly well, with no significant difference in average tardiness. However, GP takes almost twice as long to get to the same results, with an average 14% higher maximum tardiness and similar number of late jobs on average.

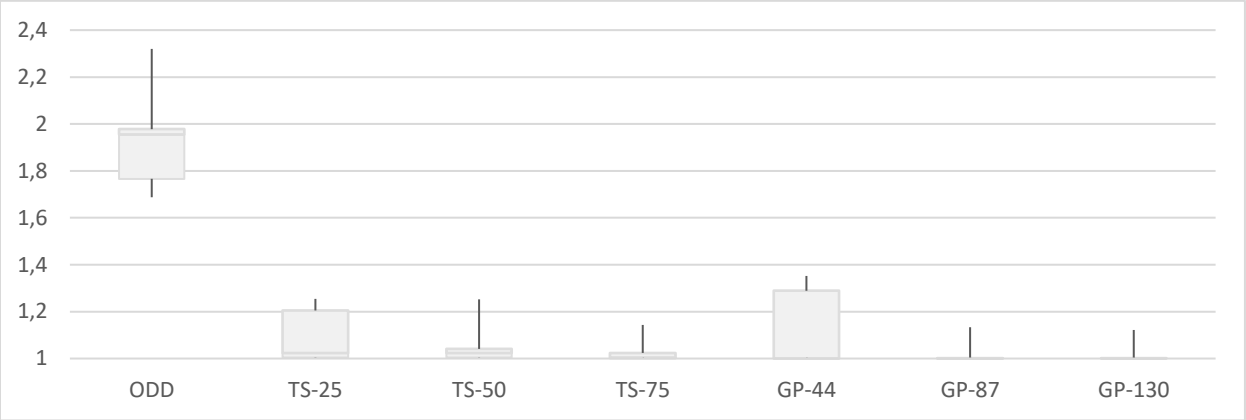

Figure 5.5 | Boxplot of normalized average tardiness for the Tannery case

Table 5.5 | Results for the Tannery case

	ODD	TS-25	TS-50	TS-75	GP-44	GP-87	GP-130
avg. tardiness (hours)	1.49	0.93	0.90	0.84	0.99	0.79	0.79
norm. avg. tardiness	1.94	1.10	1.06	1.03	1.13	1.03	1.02
wins	0	1	1	2	3	4	4
max. tardiness	1.02	0.97	0.96	0.93	1.22	1.08	1.08
better than winner	2	2	2	2	-	-	-
late jobs (%)	4.60	3.83	3.76	3.69	3.55	3.14	3.34
better than winner	1	-	-	-	-	1	1
duration (min.)	0.18	25	50	75	44	87	130
no. iterations	-	15.8	17.2	22.4	7.4	15.2	22

5.1.6 Telescopes

Whereas in the Power Tools case ODD and LRPT performed similarly well, here LRPT performs 28% better than ODD, and is therefore the winner of the constructive methods for the Telescopes case.

As shown by Figure 5.6 and accompanying Table 5.6, TS behaves similarly to the Power Tools case in this case. Again, due to the small number of iterations it performs in the given time limit, it is unable to achieve significant improvement over the DR results.

GRASP seems to be a clear winner, though GP manages to outperform it in one of the 5 instances. However, for that instance it achieves an insignificant 2% lower average tardiness. Overall, then, GRASP achieves the best result, even in terms of maximum tardiness and percentage of late jobs. It does, however, require 220 minutes to perform a single iteration, which is longer than the two hours we aimed for.

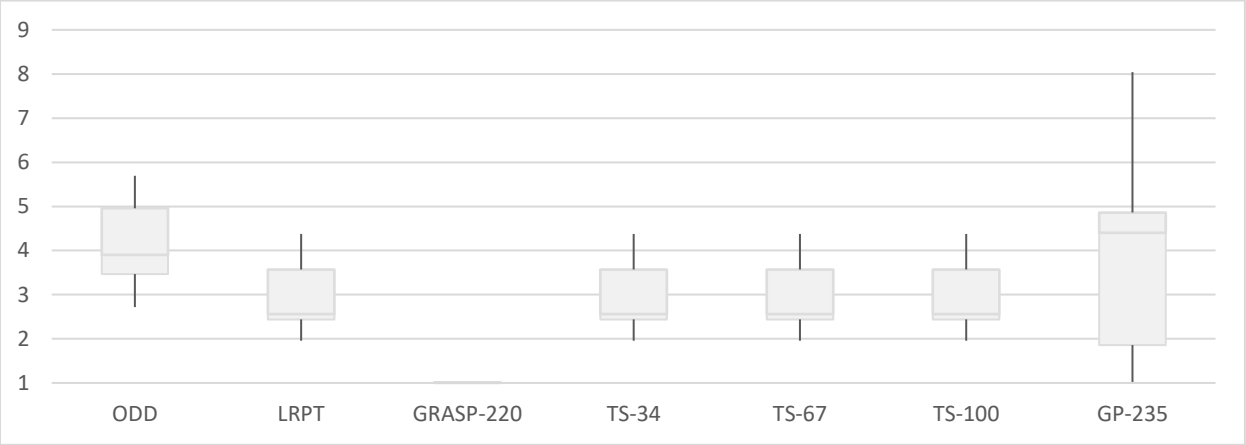

Figure 5.6 | Boxplot of normalized average tardiness for the Telescopes case

Table 5.6 | Results for the Telescopes case

	ODD	LRPT	GRASP-220	TS-34	TS-67	TS-100	GP-235
avg. tardiness (hours)	320898.19	228782.47	85263.45	228782.41	228782.40	228782.40	283536.62
norm. avg. tardiness	4.15	2.98	1.00	2.98	2.98	2.98	4.03
wins	0	0	4	0	0	0	1
max. tardiness	3.49	1.47	0.87	1.14	1.47	1.47	5.83
better than winner	1	1	1	2	1	1	-
late jobs (%)	99.98	99.98	99.78	99.98	99.98	99.98	99.95
better than winner	-	-	1	1	-	-	-
duration (min.)	31.04	32.25	220	34	67	100	235
no. iterations	-	-	1	1	2	3	1.2

5.2 Hypothesis testing

From the initial results, we get three questions that we decide to look further into in this section, namely:

1) Why does SPT outperform ODD in the Food case by 11 times, while ODD achieves good results in all other cases?
2) Why does GRASP perform well in the Fridges, Power Tools and Telescopes cases?
3) Why does LRPT outperform ODD in the Power Tools and Telescopes cases?

We aim to find some answers to these questions by adapting the mentioned cases to see whether or not that influences the results. Due to the limited time available for this research, we have chosen not to adapt the Telescopes case.

5.2.1 Adapted Food case

The Food case stands out because it is the smallest case of all six in terms of jobs and operations. It seems probable therefore that the answer to question 1 is related to the size of the Food case. To test out this hypothesis, we create a case similar to the Food case with 1.5 times the jobs in the same horizon. We found that this increase in jobs made the duration of SB infeasibly long, and we therefore do not discuss that method here.

Figure 5.7 and Table 5.7 show the result of applying ODD, SPT and LRPT to the new case. After increasing the number of jobs, ODD becomes 15 times worse than SPT on average. Without increasing the number of jobs, it was 12 times worse. This increase was the opposite of what we expected, and we conclude that the size of the Food case is not what makes SPT perform better than ODD for this case.

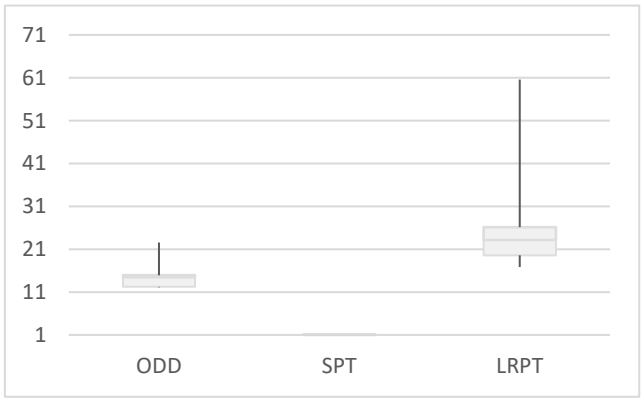

Figure 5.7 | Boxplot of normalized average tardiness for the Food case with more jobs in the same horizon

Table 5.7 | Results for the Food case with more jobs in the same horizon

	ODD	SPT	LRPT
avg. tardiness (hours)	4827.68	351.41	8044.59
norm. avg. tardiness	15.25	1.00	29.26
wins	0	5	0
max. tardiness	1.15	1.00	1.12
better than winner	1	-	2
late jobs (%)	91.79	11.79	96.58
better than winner	-	-	-
duration (min.)	0.09	0.06	0.08

5.2.2 Adapted Fridges and Power Tools cases

With regards to question 2 and 3, we hypothesize the answer is related to either the use of release dates, downtime or maintenance in the cases. We test all three of these hypotheses on the Power Tools case by creating three new cases (one without release dates, one without downtime, and one with 0.67 times the number of jobs in the same horizon) and the first two of these hypotheses on the Fridges case.

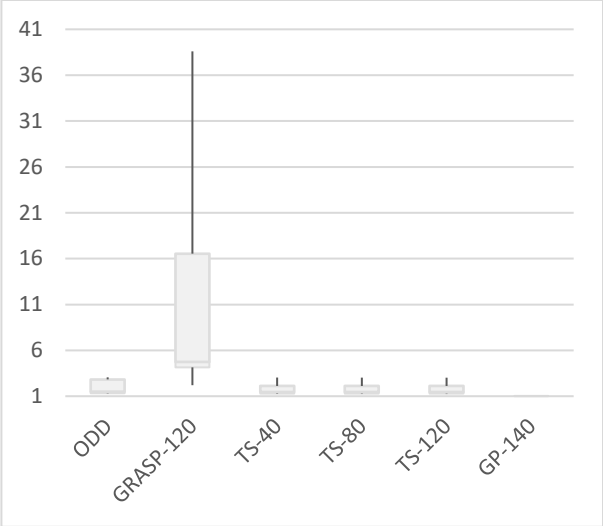

Figure 5.8 | Boxplot of normalized average tardiness for the Fridges case without release dates

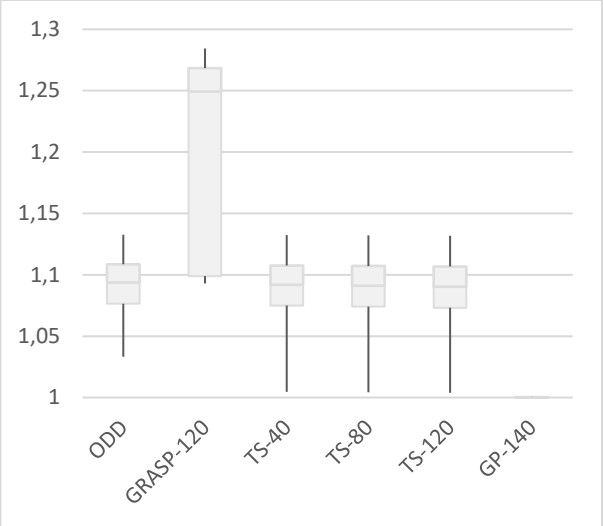

Figure 5.9 | Boxplot of normalized average tardiness for the Fridges case without tree precedence

Table 5.8 | Results for the Fridges case without release dates

	ODD	GRASP-120	TS-40	TS-80	TS-120	GP-140
avg. tardiness (hours)	39.33	63.41	31.59	31.56	31.54	17.23
norm. avg. tardiness	1.99	13.25	1.83	1.82	1.82	1.00
wins	0	0	0	0	0	5
max. tardiness	0.60	3.98	0.71	0.72	0.72	1.00
better than winner	4	1	3	3	3	-
late jobs (%)	19.03	19.89	15.94	15.94	15.93	11.45
better than winner	-	-	-	-	-	-
duration (min.)	4.63	120	40	80	120	140
no. iterations	-	1	22	41	57.2	1

Table 5.9 | Results for the Fridges case without tree precedence

	ODD	GRASP-120	TS-40	TS-80	TS-120	GP-140
avg. tardiness (hours)	35.66	39.24	35.35	35.34	35.33	32.73
norm. avg. tardiness	1.09	1.20	1.08	1.08	1.08	1.00
wins	0	0	0	0	0	5
max. tardiness	0.74	1.38	0.77	0.77	0.77	1.00
better than winner	4	1	3	3	3	-
late jobs (%)	94.19	93.25	94.15	94.15	94.15	93.85
better than winner	-	4	1	1	1	-
duration (min.)	3.58	120	40	80	120	140
no. iterations	-	1	28.8	57.8	86.8	1

Figure 5.8 & Table 5.8 and Figure 5.9 & Table 5.9 show the results for the Fridges cases without release dates and without tree precedence, respectively. For both cases, GP is the winner for all 5 instances, in favor of GRASP that used to win 3 out of 5 instances (recall Table 5.2). These two adapted cases are the only two for which GP is the clear winner, as in other cases it at best has an (insignificant) 1% improvement over TS, which in that case uses less computation time (the Tannery case, Table 5.5).

The Figures and Tables below show the results of the adapted Power Tools cases. With regards to average tardiness, GRASP is still a clear winner in all three adapted Power Tools cases. However, where in the initial Power Tools case GRASP performed on average 5 times better than other methods, this factor changes to averages of 3 and 4 for the adapted cases without release dates and tree precedence, respectively. These results are in line with the adapted Fridges cases, where we also saw a decrease in the relative performance of GRASP when release dates and tree precedence were removed from the case. However, where GRASP loses to GP in the adapted Fridges cases, GRASP still wins in the adapted Power Tools cases.

The third adapted case, where preventive maintenance is removed, the relative performance of GRASP becomes better. Where in the initial Power Tools case GRASP performed on average 5 times better than other methods, removing preventive maintenance increases this average factor to 25.

With regards to computation time, we notice changes in the time used by GP for a single iteration. Initially, this was 240 minutes, which changes to 300, 170 and 180 for the adapted cases without release dates, tree precedence and preventive maintenance, respectively.

We also find that when removing preventive maintenance in the Power Tools case, LRPT no longer outperforms ODD.

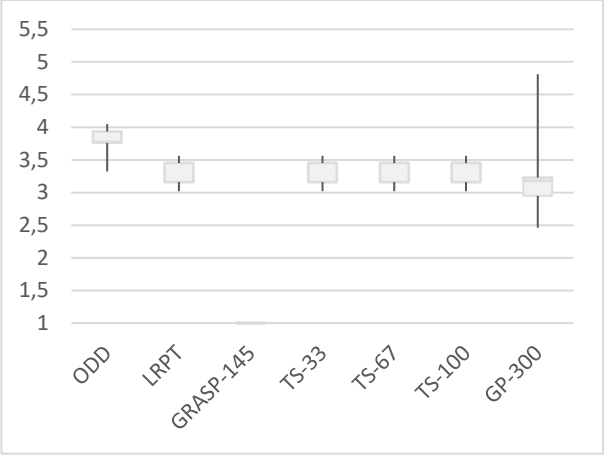

Figure 5.10 | Boxplot of normalized average tardiness for the Power Tools case without release dates

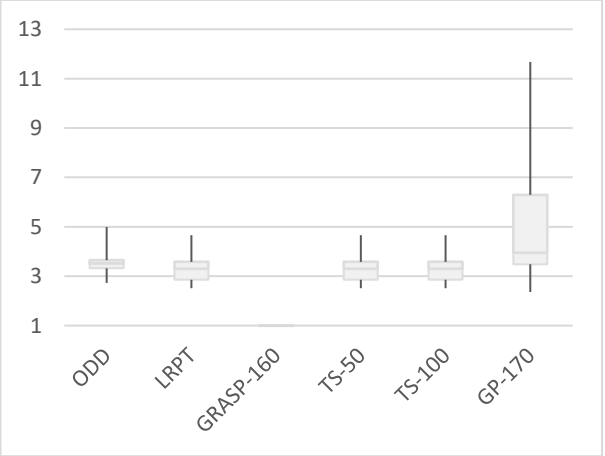

Figure 5.11 | Boxplot of normalized average tardiness for the Power Tools case without tree precedence

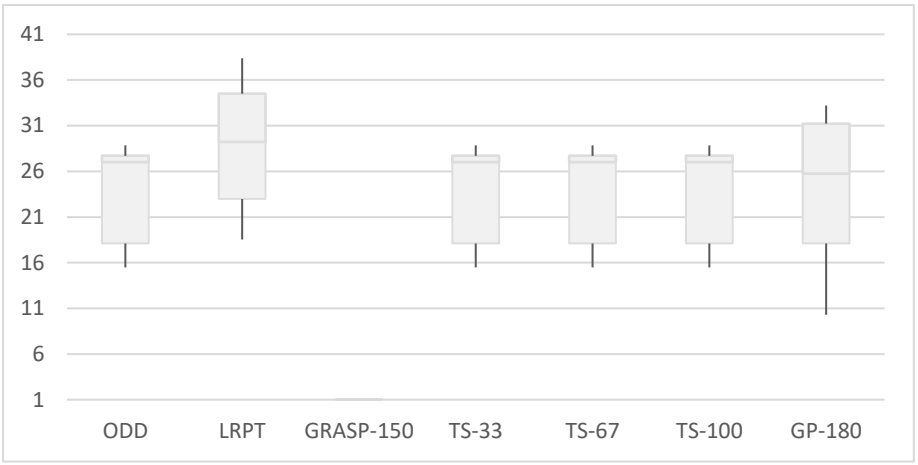

Figure 5.12 | Boxplot of normalized average tardiness for the Power Tools case without preventive maintenance

Table 5.10 | Results for the Power Tools case without release dates

	ODD	LRPT	GRASP-145	TS-33	TS-67	TS-100	GP-300
avg. tardiness (hours)	72728.53	63109.89	19376.94	63109.75	63109.69	63109.63	64326.96
norm. avg. tardiness	3.77	3.27	1.00	3.27	3.27	3.27	3.33
wins	0	0	5	0	0	0	0
max. tardiness	2.35	1.47	1.00	1.47	1.47	1.47	3.58
better than winner	-	-	-	-	-	-	-
late jobs (%)	99.71	99.70	83.20	99.70	99.70	99.70	97.69
better than winner	-	-	-	-	-	-	-
duration (min.)	19.21	18.36	145	33	67	100	300
no. iterations	-	-	1	1	2	3	1

Table 5.11 | Results for the Power Tools case without tree precedence

	ODD	LRPT	GRASP-160	TS-67	TS-100	GP-170
avg. tardiness (hours)	84646.03	78311.40	24173.33	78311.28	78311.20	127261.97
norm. avg. tardiness	3.64	3.38	1.00	3.38	3.38	5.55
wins	0	0	5	0	0	0
max. tardiness	2.24	1.46	1.00	1.46	1.46	4.73
better than winner	-	-	-	-	-	-
late jobs (%)	99.49	98.85	95.98	98.85	98.85	99.32
better than winner	-	-	-	-	-	-
duration (min.)	27.52	26.03	160	50	100	170
no. iterations	-	-	1	1	2	1

Table 5.12 | Results for the Power Tools case without preventive maintenance

	ODD	LRPT	GRASP-150	TS-33	TS-67	TS-100	GP-180
avg. tardiness (hours)	58814.36	71719.41	2663.57	58814.13	58814.02	58813.90	57577.41
norm. avg. tardiness	23.43	28.72	1.00	23.43	23.43	23.43	23.72
wins	0	0	5	0	0	0	0
max. tardiness	2.57	1.99	1.00	2.06	2.57	2.57	2.38
better than winner	-	-	-	1	-	-	-
late jobs (%)	99.98	99.97	99.23	79.98	99.98	99.98	99.97
better than winner	-	-	-	1	-	-	-
duration (min.)	19.65	18.47	150	33	67	100	180
no. iterations	-	-	1	1	2	3	1

5.3 Conclusion

Of the dispatching rules, ODD is most often the best-performing method, namely in 3 out of 6 initial cases. In the cases of Power Tools and Telescopes, LRPT performs better than ODD, with a difference in normalized average tardiness of 0.11 and 1.17, respectively. The other exception is the Food case, where SPT performs 11 times better than ODD. Increasing the number of jobs in the Food case (while keeping its horizon equal) by a factor of 1.5 does not change this behavior, though it does make the SB method infeasible due to the increase in computation time. SB is thus only applied to the initial Food case, where it performs twice as well as SPT. This performance increase does come as a cost of computation time, as SB takes 14.23 minutes while SPT takes 2 seconds.

TS performs well in all cases, which is to be expected as it improves upon the best DR solution. We do see that for smaller cases such as Food, the improvement gained by applying TS is much larger than with larger cases as Power Tools or Telescopes. This is expected behavior as well, as an improvement iteration takes a longer amount of time with these larger cases, therefore allowing for less improvement iterations when keeping computation time equal across cases. TS does not win in these larger cases, as GRASP then achieves better results.

GRASP wins with regards to average tardiness in the Fridges, Power Tools and Telescopes cases, but it requires a large amount of computation time (120, 140 and 220 minutes respectively) to perform a single iteration. This single iteration is then enough for it to achieve a winning result, which is unexpected behavior.

When comparing GRASP and TS, as Table 5.13 does, we do find that TS performs better on average and has a lower standard deviation. This is mainly caused by the Tannery case, where GRASP performs 50

times worse than TS. However, when discounting this case from the average it becomes 2.93, which is still 34% worse than TS's average.

Table 5.13 | Comparison of normalized average tardiness for GRASP and TS

	Food	Fridges	Kitchens	Power tools	Tannery	Telescopes	Average	St. dev.
GRASP	10.46	1.13	1.06	1	50.62	1	10.88	18.10
TS	1	1.71	1	5.36	1.03	2.98	2.18	1.59

It is unclear what makes GRASP perform better than TS in the respective cases. It is unclear why GRASP does outperform TS for half of the cases, while GRASP is only able to perform a single iteration in the given computation time. We do know that not using release dates or tree precedence decreases GRASP's relative performance in the Fridges and Power Tools cases, though it does not necessarily mean that other methods achieve better results than GRASP. Not using preventive maintenance, however, made GRASP's relative performance even better in the Power Tools case.

When adapting the Fridges case to exclude the usage of release dates and tree precedence, we do find that GP overtakes GRASP in terms of relative performance, also only requiring a single iteration. It is unclear why this is. GP further does provide the best average tardiness for the Kitchens and Tannery cases, but with an insignificant difference to TS, and with a longer computation time (13 versus 1 minute and 87 versus 75 minutes for the respective cases).

The difference between methods for the Kitchen case is smaller than 1%. TS finds the winning result for each instance after a single search iteration, after which it can find no further improvements. This suggests a (near-)optimal solution is found very easily for this case. When adapting this case to include 1.5 times more jobs in the same horizon, there are no significant changes in these results, indicating that increasing the number of jobs has no correlation with a better performance of ODD. However, it might also be that a higher increase than 1.5 is required to show changes in the results.

Table 5.14 gives an overview of the best performing methods with regards to average tardiness for each initial case.

Table 5.14 | Best performing methods with regards to average tardiness per initial case

	Food	Fridges	Kitchens	Power Tools	Tannery	Telescopes
Constructive	SB	ODD	ODD	LRPT	ODD	LRPT
Improvement	TS	GRASP	TS	GRASP	TS	GRASP

6 Conclusions & recommendations

This chapter concludes our research by answering the research question in Section 6.1. Section 6.2 then discusses the provided answer by mentioning some limitations. Section 6.3 provides recommendations for future research.

6.1 Conclusions

In order for Syze to be able to decide what scheduling methods to implement, and to give their customers advice on what scheduling method to use, this research aimed to find out what methods perform best in solving different cases of a Flexible Job Shop Scheduling problem. The research question posed in Section 1.3 is:

*"Which **methods** perform **best** in solving a Job Shop Scheduling problem for different **cases** of customers of **Syze's software**?"*

We now have a partial answer to this question, which we elaborate on in this Section.

Based on the criteria of explainability and repeatability, we selected Dispatching Rules (DRs), Shifting Bottleneck (SB), Tabu Search (TS), Greedy Random Adaptive Search Procedure (GRASP) and Genetic Programming (GP) as **methods** to test. The first two of these methods are constructive methods, the others are improvement methods. As the latter use more computation time (having to start from one or multiple starting solution(s) generated by a constructive method, which they then attempt to improve), which is not feasible for all of Syze's customers, we discuss the two types of methods separately. Which method performs **best** is determined by looking at the average tardiness of jobs in the final schedule. We initially selected six customer **cases** to test these methods on, after which we tested some adaptions of these initial cases as well, in order to gain additional insight into the initial results.

The scheduling method currently used by Syze, the Earliest Operation Due Date DR (ODD), is the best-performing constructive method in 3 of the 6 cases. In two of the other cases, which are the only cases that include preventive maintenance, the Longest Remaining Processing Time DR (LRPT) performs best. When adapting one of these cases to exclude preventive maintenance, ODD becomes the best performer, thereby proving that LRPT is better when using preventive maintenance, and ODD is best otherwise. In the last case, SB is the best performer. Due to SB's inability to cope with larger schedules, this is the only case SB can be applied upon. The second-best performer in this case is the Shortest Processing Time DR (SPT). It is unclear why this is the second-best performer, instead of ODD and LRPT that have proven effective in the other cases.

Of the improvement methods, TS performs best for 3 of the 6 cases, while GRASP performs best for the others. On average over all cases, TS does perform better than GRASP. Factors that negatively impact the performance of GRASP are not using release dates and not using tree precedence structures. A factor that positively impacts the performance of GRASP is not using preventive maintenance.

Considering these findings, we deem LRPT, TS and GRASP as methods that should be implemented by Syze in their ERP system. As one DR is already implemented in the software system, implementing another will require less resources than implementing an entirely different method, which makes LRPT the quickest win. After that, we recommend implementing TS, as it on average performs better than GRASP, which is then left as the final method to be implemented. For now, we do not suggest implementing SB due to its inapplicability to larger cases.

6.2 Discussion

A major drawback of our research is the limited availability of customer case data. It is unknown whether the six customer cases are representative of Syze's entire customer base, for what portion of the customer base certain cases are representative, and therefore how to prioritize the implementation of new methods according to what method has added value for most of the customer base.

Due to the limited time available for this research, we were also not able to test all adapted cases that we would have liked to. For example, we would have liked to test an adaption of the Power Tools case with less jobs, to see whether that influences the performance of GRASP. We would also have liked to confirm the results of the adapted Fridges and Power Tools cases by testing out the same adaptions on the Telescopes case, where GRASP also is the best performing method.

Because of the limited time available for this research, we also limited the computation time given to the improvement methods. Giving these methods more time might also influence the results, especially since GRASP and GP could often only perform a single iteration.

Lastly, we decided to only use scheduling methods found in literature for this research. However, none of these methods are used for the average tardiness objective function in their original literature, which underscores a gap between scientific theory and practice. Another approach we could have taken is to design and test new scheduling methods that specifically aim to minimize average tardiness.

6.3 Recommendations for future research

Though we applied SB only to a single case due to the infeasibility of our implementation for larger cases, it was the best performing constructive method in that single case. We recommend looking into a

way to adapt SB to shorten its computation time for large cases, as SB then might win from ODD and LRPT in other cases as well.

In line with the mentioned limitations, we recommend looking further into factors that influence when TS and when GRASP perform better. We recommend the same for factors that influence why in one case SPT performs better than ODD. Knowing these factors will further improve Syze's ability to recommend scheduling methods to their customers. For GRASP, some factors might be the size of cases, the horizon of cases, or the DR used for generating the initial solution. For SPT, some factors might be the size of cases,

GRASP and TS are already the two best performing improvement methods. However, we only tested out a single neighborhood function for these methods, whereas many other functions exist. We recommend testing these as well in order to further improve performance of these methods.

We discounted Simulated Annealing (SA) as a scheduling method due to the four parameters that need to be set depending on the case, making it too complex to meet our explainability requirement. We recommend looking for a way to dynamically set these parameters depending on the instance date (as we did with our Tabu Search (TS) list length). This would remove the complexity and make this a feasible method for Syze's ERP system.

Though we did take some FJSP extensions into account (release dates, queue, wait and setup times, tree precedence and preventive maintenance), there are other extensions we left out of our research, namely variable changeover times and machine-dependent processing times, as we had no data available that included these extensions. We recommend finding data that does include these extensions and testing the selected methods on these as well.

A major advantage of ERP software is easy data-integration. We therefore recommend researching possible usage of other data available in the system for improving the Finite Production Scheduler. Examples of these are taking observed stochasticity in variables such as processing times into account (which requires the tracking of the realized values of these variables), and immediate rescheduling when disruptions, e.g. machine breakdown, are recorded in the system.

7 References

Adams, J., Balas, E., & Zawack, D. (1988, March). The shifting bottleneck procedure for job shop scheduling. *Management Science, 34*(3), 391-401.

Aiex, R., Binato, S., & Resende, M. (2003). Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing, 29, 393-430.

Bagheri, A., Zandieh, M., Mahdavi, I., & Yazdani, M. (2010). An artificial immune algorithm for the flexible job-shop scheduling problem. Future Generation Computer Systems, 26, 533-541.

Baker, J. (1987). Reducing bias and inefficiency in the selection algorithm. In J. Grefenstette (Ed.), Proc. 2nd Int. Conf. Genetic Algorithms and Their Applications (pp. 14-21). London: Lawrence Erlbaum.

Baker, K. (1984, September). Sequencing Rules and Due-Date Assignments in a Job Shop. Management Science, 30(9), 1093-1104. Retrieved from https://www.jstor.org/stable/2631726

Balas, E. (1969). Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm. Operations Research, 17(6), 941-957.

Balas, E., & Vazacopoulos, A. (1998). Guided Local Search with Shifting Bottlenech for Job Shop Scheduling. Management Science, 44(2), 262-275. doi:10.1287/mnsc.44.2.262

Bedworth, D. D., & Bailey, J. E. (1999). Integrated Production Control Systems: Management, Analysis, Design. New York, NY, United States: John Wiley & Sons, Inc.

Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. OR Spektrum, 17, 87-92.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research, 41, 157-183.

Brucker, P., Jurisch, B., & Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics, 49, 107-127.

Carlier, J. (1982). The one-machine sequencing problem. European Journal of Operational Research, 11, 42-47.

Chen, B., Potts, C., & Woeginger, G. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In D.-Z. Du, & P. Pardalos (Eds.), Handbook of Combinatorial Optimization (Vol. 3, pp. 21-169). Kluwer Academic Publishers.

Chiang, T.-C., & Lin, H.-J. (2012). Flexible job shop scheduling using a multiobjective memetic algorithm. Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence (pp. 49-56). Berlin, Heidelberg: Springer.

Chiang, T.-C., & Lin, H.-J. (2013). A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. International Journal of Production Economics, 141, 87-98.

Dauzere-Peres, S., & Lasserre, J.-B. (1993). A modified shifting bottleneck procedure for job-shop scheduling. International Journal of Production Research, 31(4), 923-932.

Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Annals of Operations Research, 70, 281-306.

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. Computers & Operations Research, 22(1), 15-24.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. Annals of Operations Research, 41, 231-252.

Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. Computers & Operations Research, 22(1), 25-40.

Fattahi, P., Mehrabad, M., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing, 18, 331-342. doi:10.1007/s10845-007-0026-8

Gao, J., Gen, M., Sun, L., & Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. Computers & Industrial Engineering, 53, 149-162.

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research, 35, 2892-2907.

Gao, K., Suganthan, P., Pan, Q., Chua, T., Cai, T., & Chong, C. (2014). Pareto-based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling. Information Sciences, 289, 76-90.

Garey, M., Johnson, D., & Sethi, R. (1976). The complexity of flowshop and job-shop scheduling. Mathematics of Operations Research, 1, 117-219.

Gonçalves, J., de Magalhães Mendes, J. J., & Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research, 167, 77-95.

Graves, G., & Lee, C.-Y. (1999, October). Scheduling maintenance and semiresumable jobs on a single machine. Naval Research Logistics, 46(7), 845-863. doi:10.1002/(SICI)1520-6750(199910)46:7<845::AID-NAV6>3.0.CO;2-#

Heerkens, H. (2012). Geen Probleem. Van Winden Communicatie.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. OR Spektrum, 15, 205-215.

Hurink, J., Kok, A., Paulus, J., & Schutten, J. (2011). Time-constrained project scheduling with adjacent resources. Computers & Operations Research, 38, 310-319.

Jayamohan, M., & Rajendran, C. (2000). New dispatching rules for shop scheduling: A step. International Journal of Production Research, 38(3), 563-586. doi:10.1080/002075400189301

Kacem, I., Hammadi, S., & Borne, P. (2002a, February). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Transactions on Systems, Man and Cybernetics, 32(1), 1-13.

Kacem, I., Hammadi, S., & Borne, P. (2002b). Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Mathematics and Computers in Simulation, 60, 245-276.

Kalma, K. (2020, February 10). Applying A Variety Of Heuristics To Solve A Complex Flexible Job Shop.

Kelley Jr., J. E. (1961). Critical-Path Planning and Scheduling: Mathematical Basis. Operations Research, 9(3), 296-320.

Laarhoven, P. v., Aarts, E., & Lenstra, J. (1992). Job Shop Scheduling by Simulated Annealing. Operations Research, 40(1), 113-125. Retrieved from https://www.jstor.org/stable/171189

Li, J., Pan, Q., & Xie, S. (2012). An effective shuffled frog-leaping algorithm for multi-objective flexiblejob shop scheduling

problems. Applied Mathematics and Computation, 218, 9353-9371.

Li, J.-Q., Pan, Q.-K., & Chen, J. (2012). A hybrid Pareto-based local search algorithm for multi-objective flexible job shop scheduling problems. International Journal of Production Research, 50(4), 1063-1078. doi:10.1080/00207543.2011.555427

Li, J.-Q., Pan, Q.-K., & Gao, K.-Z. (2011). Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. The International Journal of Advanced Manufacturing Technology, 60, 1159-1169.

Li, J.-Q., Pan, Q.-K., & Liang, Y.-C. (2010). An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. Computers & Industrial Engineering, 59, 647-662.

Li, J.-Q., Pan, Q.-K., & Tasgetiren, M. (2014). A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. Applied Mathematical Modelling, 38, 1111-1132.

Li, J.-Q., Pan, Q.-K., Suganthan, P., & Chua, T. (2011). A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 52, 683-697. doi:DOI 10.1007/s00170-010-2743-y

Li, X., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. Internation Journal of Production Economics, 174\, 93-110.

Mastrolilli, M., & Gabardella, L. (2000). Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling, 3, 3-20.

Moslehi, G., & Mahnam, M. (2011). A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. Internation Journal of Production Economics, 129, 14-22.

Muth, J. F., Thompson, G. L., & Winters, P. R. (1963). Industrial scheduling. Englewood Cliffs: Prentice-Hall.

Nowicki, E., & Smutnicki, C. (1996, June). A Fast Taboo Search Algorithm for the Job Shop Problem. Management Science, 42(6), 797-813. Retrieved from https://www.jstor.org/stable/2634595

Nowicki, E., & Smutnicki, C. (2005). An Advanced Tabu Search Algorith for the Job Shop Problem. Journal of Scheduling, 8, 145-159.

Ong, Z., Tay, J., & Kwoh, C. (2005). Applying the Clonal Selection Principle to Find Flexible Job-Shop Schedules. ICARIS 2005, LNCS 3627 (pp. 442-455). Berlin Heidelberg: Springer-Verlag.

Pezella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. Computers & Operations Research, 35, 3202-3212.

Rego, C., & Duarte, R. (2009). A filter-and-fan approach to the job shop scheduling problem. European Journal of Operational Research, 194, 650-662.

Saidi Mehrabad, M., & Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithm. International Journal of Advanced Manufacturing Technology, 32, 563-570.

Sha, D., & Hsu, C.-Y. (2006). A hybrid particle swarm optimization for job shop scheduling problem. Computers & Industrial Engineering, 51, 791-808.

Storer, R., Wu, S., & Vaccari, R. (1992, October). New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. Management Science, 38(10), 1495-1509. Retrieved from https://www.jstor.org/stable/2632676

Tay, J., & Ho, N. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering, 54, 453-473.

Wang, L., & Zheng, D.-Z. (2001). An effective hybrid optimization strategy for job-shop scheduling problems. Computers & Operations Research, 28, 585-596.

Wang, L., Wang, S., Zhou, G., & Liu, M. (2012). A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. Computers & Industrial Engineering, 62, 917-926.

Wang, L., Zhou, G., & Liu, M. (2012). An enhanced Pareto-based artificial bee colony algorithm for the multi-objective flexible job shop scheduling. The International Journal of Manufacturing Technology, 60, 1111-1123.

Wang, L., Zhou, G., Xu, Y., Wang, S., & Liu, M. (2012). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 60, 303-315. doi:10.1007/s00170-011-3610-1

Wang, X., Gao, L., Zhang, C., & Shao, X. (2010). A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. Internation Journal of Manufacturing Technology, 51, 757-767.

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Computers & Industrial Engineering, 48, 409-425.

Xing, L.-N., Chen, Y.-W., Wang, P., Zhao, Q.-S., & Xiong, J. (2010). A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems. Applied Soft Computing, 10, 888-896.

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Systemns with Applications, 37, 678-687.

Yuan, Y., & Xu, H. (2015, January). Multiobjective flexible job shop scheduling using memetic algorithms. IEEE Transactions on Automation Science and Engineering, 12(1), 336-353.

Zhang, G., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Systems with Applications, 38, 3563-3573.

8 Appendices

8.1 Tree precedence job generation pseudo-code

```
1 Determine total number of operations T using a uniform distribution
2 Determine the number of start and end operations S and E (without predecessors and
without successors, respectively) using uniform distributions
3 If S + E > T:
4       Go to line 2
5 While unsuccessful and tries < 500(initialized as True and 0):
6       Determine the number of predecessors for all operations (except start operations)
        using an empirical distribution
7       Determine the number of successors for all operations (except end operations)
        using an empirical distribution
8       tries = tries + 1
9       unsuccessful = (total number of predecessors != total number of successors) or
        (maximum number of predecessors <= 1 and maximum number of successors <= 1)
10 If tries >= 500 and unsuccessful:
11      Go to line 2
12 While unsuccessful and tries < 1000 (initialized as True and 0):
13      For total number of predecessors:
14              Select a predecessor (operation that requires an additional successor) and
                a successor (operation that requires an additional predecessor)
15              Connect the predecessor with the successor and remove them from the pool
                of to be selected predecessors and successors
16      tries = tries + 1
17      unsuccessful = loop_in_structure() or disconnected_structure()
18 If tries >= 1000 and unsuccessful:
19      Go to line 2
```

Figure 8.1 | Pseudo-code for the generation of tree precedence structures

8.2 Full results

This appendix shows the results of all selected methods for all initial cases. Below is the same explanation of these results found in Section 5.1.

We call the method that achieves the lowest average tardiness for a given instance the “winner” of that instance. In order to easily compare results over all instances, cases and methods, we state the normalized values for average tardiness, as well as for one other key performance indicator (KPI), namely maximum tardiness.

Normalization is done by dividing each KPI value with the corresponding value of the winner of the same instance. A drawback of normalizing in this manner is that when the winner’s value lies close to or below zero, other values become either extremely large or negative, respectively. However, neither of those things happens in our results, so we can safely use this methodology.

In the names of the improvement methods (e.g. “TS-120”), the number refers to the computation time in minutes corresponding to the results. This way, we can compare the results of these methods over time.

Each case’s results are shown using a boxplot and a table. The boxplots show the spread of the normalized values of average tardiness over each instance. The tables show the averages over all instances of the normalized KPIs for the same methods and cases. In these tables, the number of times (out of the 5 instances) the method is a “winner”, as defined above, is given by the “wins” row. Sometimes, multiple methods achieve the same (winning) schedule, which explains why the sum of the “wins” row might be more than 5. For maximum tardiness and the percentage of late jobs, the number of times the method achieves a better value for those KPI’s than the winner is given in the row below the respective KPI, called the number of times it is “better than winner”. Values of 0 in these rows are replaced by dashes (“-”), as this makes the tables less cluttered and chaotic.

For GRASP, the number of iterations given in the tables refers to the number of times a locally optimal schedule is generated. It should be noted that the last of these might not be locally optimal, as the local search is stopped when the computation time constraint is met. For TS, the number of iterations refers to the number of search steps taken. For GP, it refers to the number of times a new generation is evaluated.

8.2.1 Food

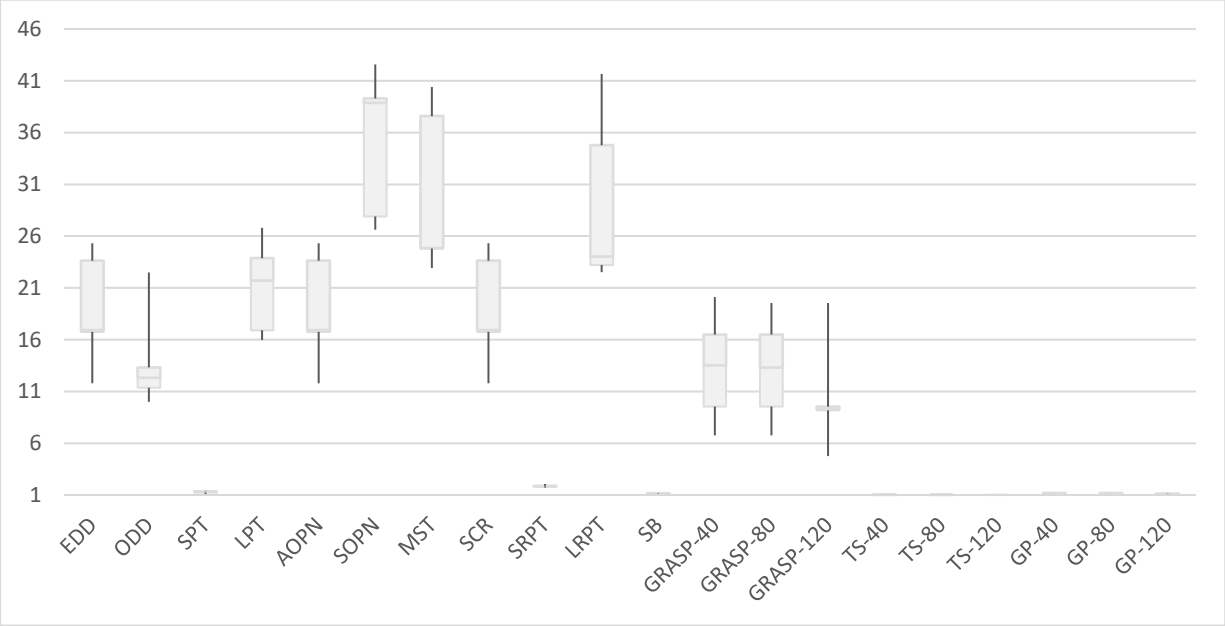

Figure 8.2 | Boxplot of normalized average tardiness for the Food case

Table 8.1 | Results for the Food case

	EDD	ODD	SPT	LPT	AOPN	SOPN	MST	SCR	SRPT	LRPT	SB
avg. tardiness (hours)	3643.7	2951.9	288.8	4256.4	3643.7	6997.1	6235.2	3643.7	403.1	6061.7	245.9
norm. avg. tardiness	18.88	13.90	1.27	21.05	18.88	35.05	30.11	18.88	1.87	29.24	1.12
wins	0	0	0	0	0	0	0	0	0	0	0
max. tardiness	1.60	1.20	1.22	1.62	1.60	1.74	1.47	1.60	1.63	1.38	1.15
better than winner	-	-	-	-	-	-	-	-	-	-	2
late jobs (%)	85.51	86.92	19.49	90.13	85.51	97.31	96.67	85.51	20.38	96.15	16.41
better than winner	-	-	-	-	-	-	-	-	1	-	4
duration (min.)	0.04	0.06	0.04	0.04	0.04	0.05	0.05	0.05	0.05	0.05	14.23

	GRASP-40	GRASP-80	GRASP-120	TS-40	TS-80	TS-120	GP-40	GP-80	GP-120
avg. tardiness (hours)	2834.15	2804.96	2286.38	221.86	221.56	220.81	255.17	254.20	246.45
norm. avg. tardiness	13.29	13.13	10.46	1.01	1.00	1.00	1.13	1.13	1.09
wins	0	0	0	2	3	5	0	0	0
max. tardiness	1.18	1.17	1.11	1.00	1.00	1.00	1.08	1.08	1.07
better than winner	2	2	1	2	1	-	-	-	1
late jobs (%)	78.59	78.59	71.67	18.08	18.08	18.08	18.72	19.23	18.59
better than winner	-	-	-	1	1	-	1	-	2
duration (min.)	40	80	120	40	80	120	40	80	120
no. iterations	1.4	2.2	3.4	88.4	108.6	125.6	24.8	50.4	75.2

8.2.2 Fridges

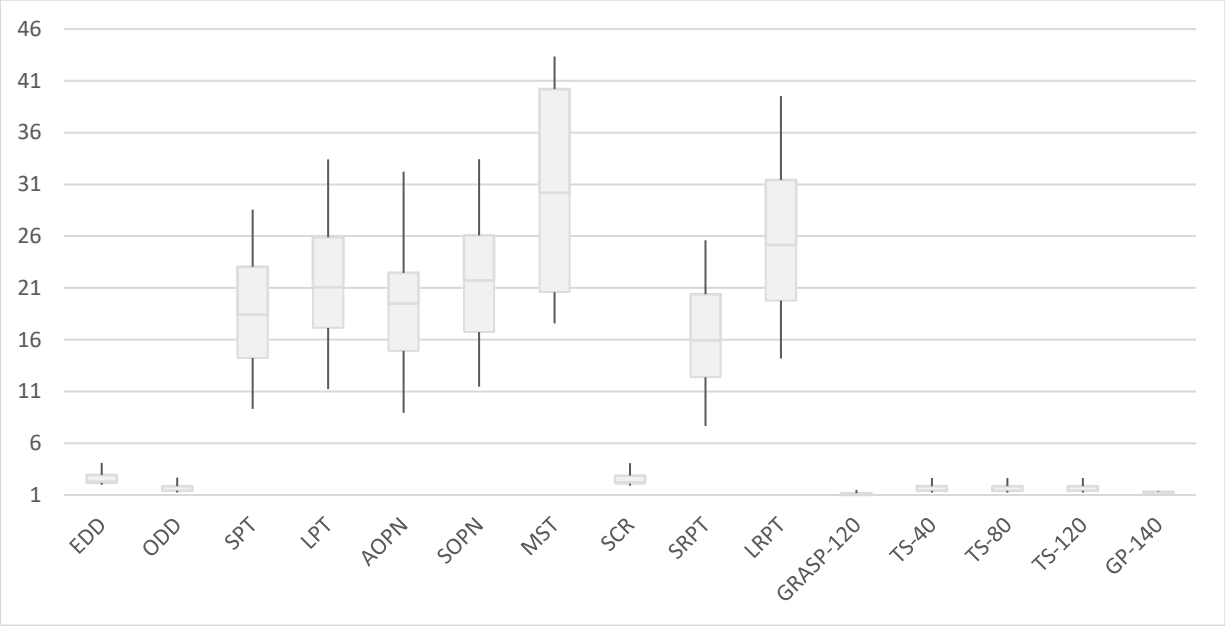

Figure 8.3 | Boxplot of normalized average tardiness for the Fridges case

Table 8.2 | Results for the Fridges case

	EDD	ODD	SPT	LPT	AOPN	SOPN	MST	SCR	SRPT	LRPT
avg. tardiness (hours)	125.51	80.34	748.68	873.58	777.59	878.37	1222.66	123.11	652.65	1046.22
norm. avg. tardiness	2.70	1.72	18.71	21.75	19.61	21.88	30.38	2.64	16.40	26.01
wins	0	0	0	0	0	0	0	0	0	0
max. tardiness	0.79	0.48	3.86	3.91	3.10	4.25	4.17	0.78	4.25	3.54
better than winner	4	5	-	-	-	-	-	4	-	-
late jobs (%)	98.96	98.91	99.40	99.43	99.80	99.76	99.82	98.95	99.34	99.68
better than winner	-	-	-	-	-	-	-	-	-	-
duration (min.)	4.41	4.93	4.42	4.44	4.50	4.62	4.58	4.55	4.76	4.57

	GRASP-120	TS-40	TS-80	TS-120	GP-140
avg. tardiness (hours)	51.41	79.75	79.69	79.67	49.81
norm. avg. tardiness	1.13	1.71	1.71	1.71	1.18
wins	3	0	0	0	2
max. tardiness	0.85	0.48	0.48	0.48	2.26
better than winner	2	5	5	5	-
late jobs (%)	98.36	98.91	98.91	98.91	98.55
better than winner	2	-	-	-	-
duration (min.)	120	40	80	120	140
no. iterations	1	24	48.8	70.6	1

8.2.3 Kitchens

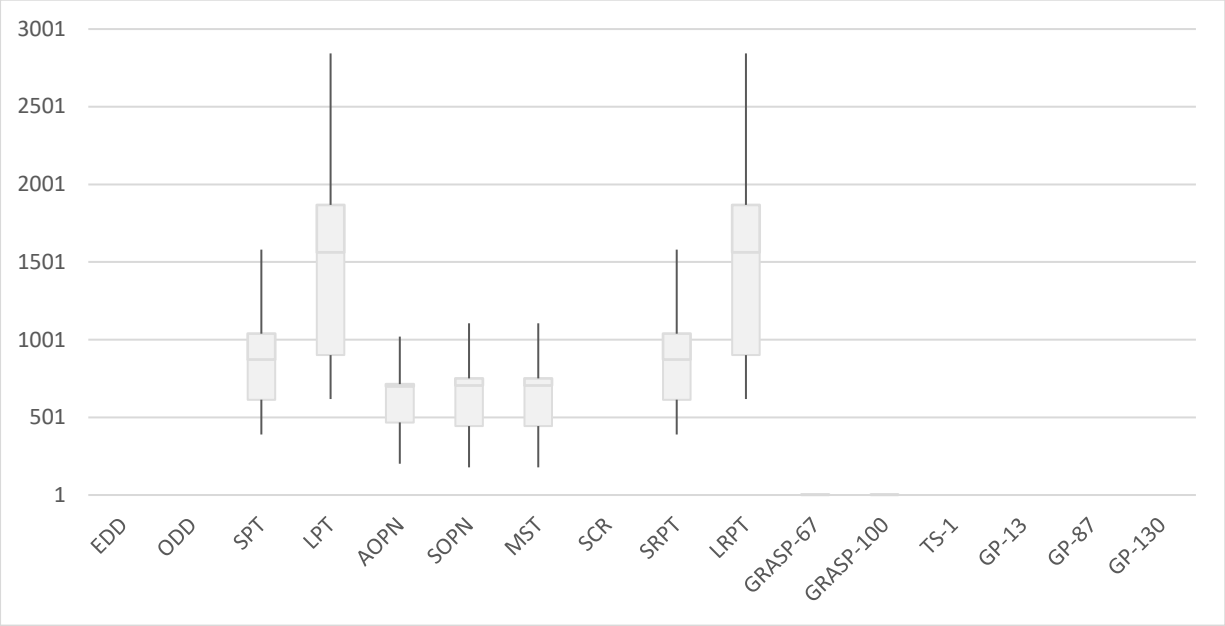

Figure 8.4 | Boxplot of normalized average tardiness for the Kitchens case

Table 8.3 | Results for the Kitchens case

	EDD	ODD	SPT	LPT	AOPN	SOPN	MST	SCR	SRPT	LRPT
avg. tardiness (hours)	0.15	0.15	105.55	176.47	73.86	73.84	73.84	0.15	105.55	176.47
norm. avg. tardiness	1.00	1.00	899.67	1559.64	621.28	637.75	637.75	1.00	899.67	1559.64
wins	2	2	0	0	0	0	0	2	0	0
max. tardiness	1.00	1.00	55.27	61.44	43.67	44.47	44.47	1.00	55.27	61.44
better than winner	-	-	-	-	-	-	-	-	-	-
late jobs (%)	2.86	2.86	18.60	25.89	18.11	17.88	17.88	2.86	18.60	25.89
better than winner	2	2	-	-	-	-	-	2	-	-
duration (min.)	0.32	0.42	0.31	0.31	0.32	0.40	0.39	0.39	0.39	0.39

	GRASP-67	GRASP-100	TS-1	GP-13	GP-87	GP-130
avg. tardiness (hours)	0.17	0.15	0.15	0.15	0.15	0.15
norm. avg. tardiness	1.14	1.06	1.00	1.00	1.00	1.00
wins	0	0	5	3	3	3
max. tardiness	1.01	1.01	1.00	1.00	1.00	1.00
better than winner	-	-	-	-	-	-
late jobs (%)	2.87	2.86	2.84	2.84	2.84	2.84
better than winner	1	1	-	1	1	1
duration (min.)	67	100	1	44	87	130
no. iterations	1	1.4	1	3.4	7.4	11.6

8.2.4 Power Tools

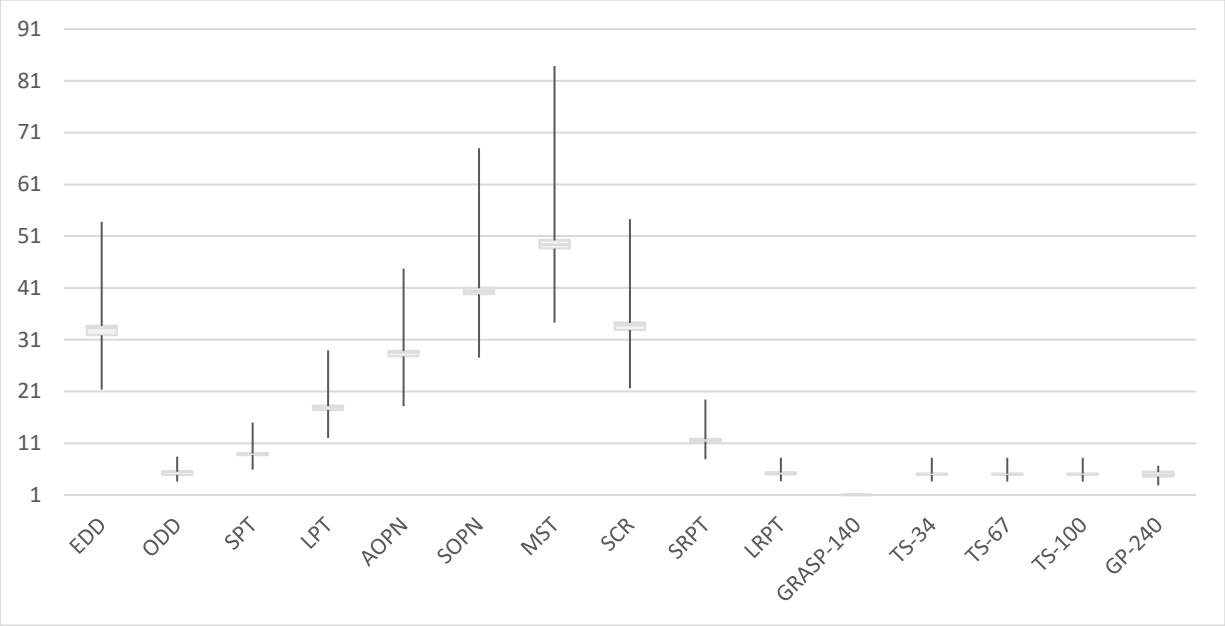

Figure 8.5 | Boxplot of normalized average tardiness for the Power Tools case

Table 8.4 | Results for the Power Tools case

	EDD	ODD	SPT	LPT	AOPN	SOPN	MST
avg. tardiness (hours)	452540.6	72948.8	123680.1	246953.6	386632.7	563598.0	693317.6
norm. avg. tardiness	34.78	5.57	9.52	18.91	29.62	43.31	53.26
wins	0	0	0	0	0	0	0
max. tardiness	16.22	2.30	7.44	9.88	16.34	15.88	15.42
better than winner	-	-	-	-	-	-	-
late jobs (%)	99.99	99.98	99.95	99.98	99.97	99.99	99.99
better than winner	-	-	-	-	-	-	-
duration (min.)	17.50	19.16	20.97	21.07	17.19	17.89	17.54

	SCR	SRPT	LRPT	GRASP-140	TS-34	TS-67	TS-100	GP-240
avg. tardiness (hours)	460705.1	161707.5	71719.4	14146.24	70199.62	70199.53	70199.46	64552.78
norm. avg. tardiness	35.38	12.42	5.46	1.00	5.36	5.36	5.36	4.89
wins	0	0	0	5	0	0	0	0
max. tardiness	16.35	15.72	1.59	1.00	1.52	1.83	1.83	2.52
better than winner	-	-	-	-	1	-	-	-
late jobs (%)	99.95	99.92	99.97	99.65	79.98	99.98	99.98	99.98
better than winner	-	-	-	-	1	-	-	-
duration (min.)	17.32	22.84	17.96	140	34	67	100	240
no. iterations	SCR	SRPT	LRPT	1.2	1	2	3	1

8.2.5 Tannery

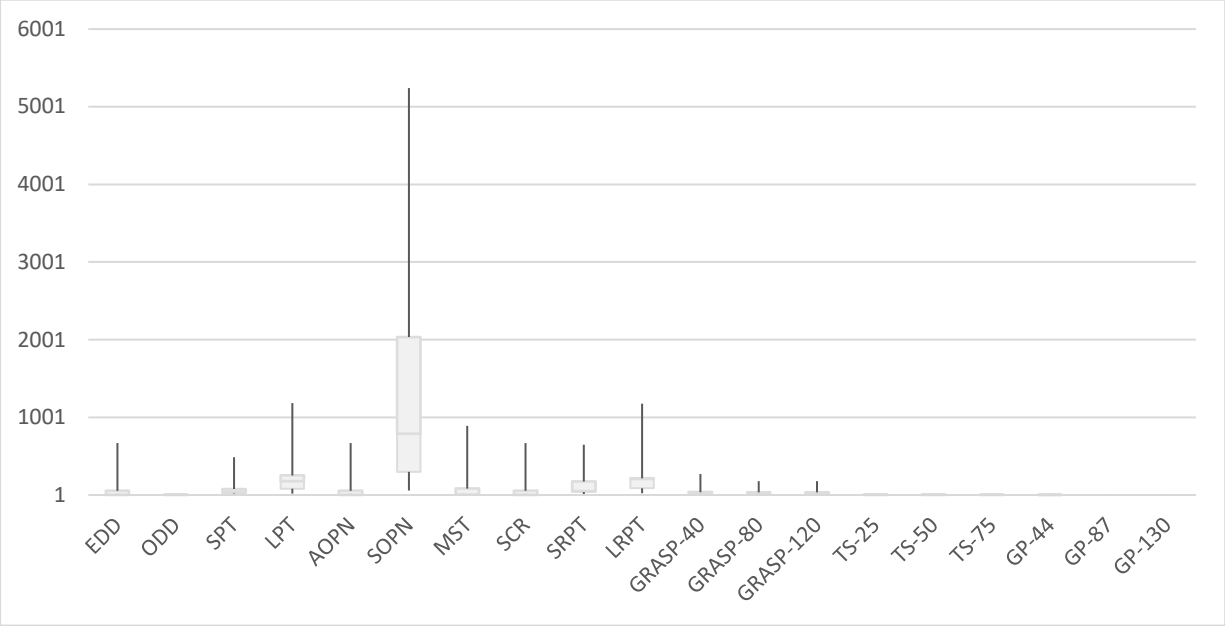

Figure 8.6 | Boxplot of normalized average tardiness for the Tannery case

Table 8.5 | Results for the Tannery case

	EDD	ODD	SPT	LPT	AOPN	SOPN	MST	SCR	SRPT	LRPT
avg. tardiness (hours)	16.13	1.49	23.46	65.40	16.13	293.08	21.82	16.13	33.35	69.16
norm. avg. tardiness	146.81	1.94	127.60	344.07	146.81	1684.51	197.75	146.81	185.78	342.84
wins	0	0	0	0	0	0	0	0	0	0
max. tardiness	10.59	1.02	39.92	45.35	10.59	52.72	10.56	10.59	54.01	24.28
better than winner	-	2	-	-	-	-	-	-	-	-
late jobs (%)	14.70	4.60	7.67	16.24	14.70	47.60	16.45	14.70	7.46	24.04
better than winner	-	1	1	-	-	-	-	-	-	-
duration (min.)	0.13	0.18	0.14	0.14	0.14	0.20	0.19	0.19	0.22	0.21

	GRASP-40	GRASP-80	GRASP-120	TS-25	TS-50	TS-75	GP-40	GP-80	GP-120
avg. tardiness (hours)	14.51	12.66	11.71	0.93	0.90	0.84	0.99	0.79	0.79
norm. avg. tardiness	73.24	53.84	50.62	1.10	1.06	1.03	1.13	1.03	1.02
wins	0	0	0	1	1	2	3	4	4
max. tardiness	17.80	20.21	19.77	0.97	0.96	0.93	1.22	1.08	1.08
better than winner	1	-	-	2	2	2	-	-	-
late jobs (%)	7.39	6.90	6.62	3.83	3.76	3.69	3.55	3.14	3.34
better than winner	1	-	-	-	-	-	-	1	1
duration (min.)	40	80	120	25	50	75	44	87	130
no. iterations	1.6	2.2	2.8	15.8	17.2	22.4	7.4	15.2	22

8.2.6 Telescopes

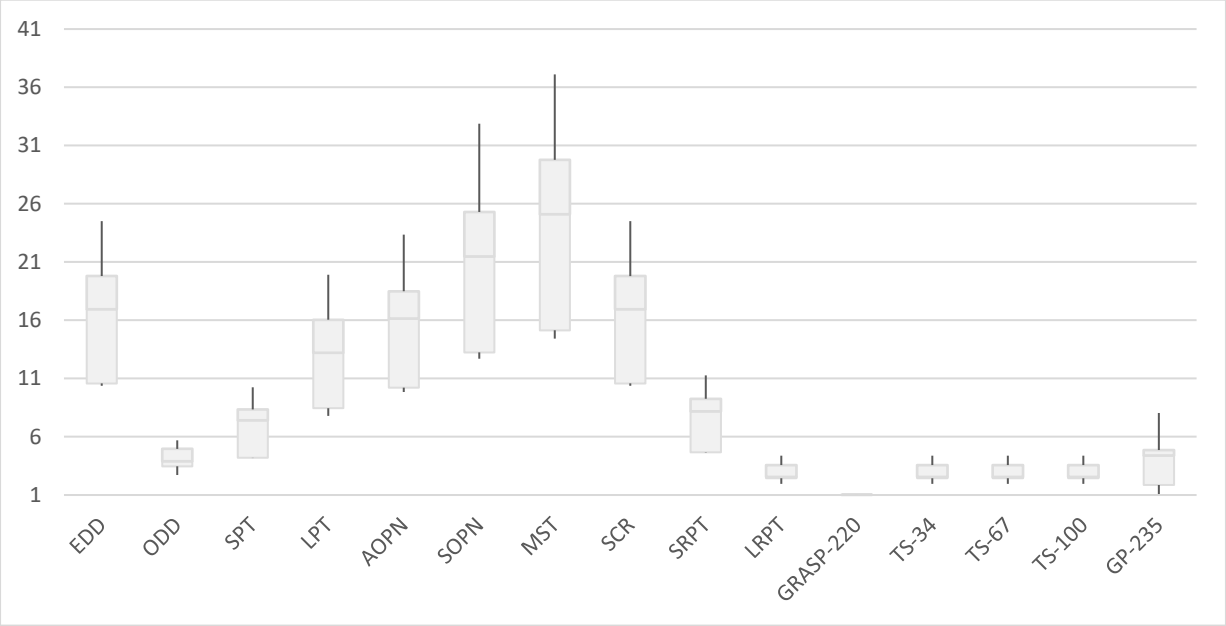

Figure 8.7 | Boxplot of normalized average tardiness for the Telescopes case

Table 8.6 | Results for the Telescopes case

	EDD	ODD	SPT	LPT	AOPN	SOPN	MST	SCR
avg. tardiness (hours)	1225341.5	320898.2	508005.5	967941.1	1165507.3	1557686.9	1792335.8	1225425.6
norm. avg. tardiness	16.44	4.15	6.87	13.08	15.61	21.12	24.31	16.44
wins	0	0	0	0	0	0	0	0
max. tardiness	14.79	3.49	8.31	11.14	14.97	14.59	14.55	14.80
better than winner	-	1	-	-	-	-	-	-
late jobs (%)	99.98	99.98	99.97	99.98	99.98	99.98	99.98	99.98
better than winner	-	-	-	-	-	-	-	-
duration (min.)	28.70	31.04	39.76	37.04	29.01	29.98	29.55	29.01

	SRPT	LRPT	GRASP-220	TS-34	TS-66	TS-100	GP-240
avg. tardiness (hours)	563226.55	228782.47	85263.45	228782.41	228782.40	228782.40	283536.62
norm. avg. tardiness	7.61	2.98	1.00	2.98	2.98	2.98	4.03
wins	0	0	4	0	0	0	1
max. tardiness	14.57	1.47	0.87	1.14	1.47	1.47	5.83
better than winner	-	1	1	2	1	1	-
late jobs (%)	99.97	99.98	99.78	79.98	99.98	99.98	99.95
better than winner	-	-	1	1	-	-	-
duration (min.)	40.20	32.25	220	33	67	100	235
no. iterations	SRPT	LRPT	1	1	2	3	1.2