Integer Linear Programming Applied to Production Planning

Master's thesis in Complex Adaptive Systems and Engineering Mathematics and Computational Science

JESPER BUSKE

THEODOR JENDLE

DEPARTMENT OF ENGINEERING PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Integer Linear Programming Applied to Production Planning

JESPER BUSKE
THEODOR JENDLE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Engineering Physics
Division of Engineering Physics
Chalmers University of Technology
Gothenburg, Sweden 2024

Integer Linear Programming Applied to Production Planning

JESPER BUSKE
THEODOR JENDLE

Supervisor: Gustav Johannesson, Recog
Examiner: Kristian Gustafsson, Department of Physics, Chalmers University of Technology and University of Gothenburg

Master's Thesis 2024
Department of Engineering Physics
Division of Engineering Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Integer Linear Programming Applied to Production Planning
JESPER BUSKE
THEODOR JENDLE
Department of Engineering Physics
Chalmers University of Technology

Abstract

The Job-Shop Scheduling Problem (JSSP) is a classic optimization problem that has been a focal point in the field of operational research for decades. As industries advance into Industry 4.0, optimizing production planning becomes increasingly crucial to enhance efficiency and competitiveness. This thesis explores the application of Integer Linear Programming (ILP) to an extended version of the JSSP, introducing new constraints and utilizing a heuristic when solving the problem. In this work, we present our mathematical formulation for the extended job-shop scheduling problem. Our approach embeds additional constraints and variables that reflect real-world production scenarios more accurately than traditional JSSP models. The performance of our formulation is evaluated by comparing our results against two benchmarks, where the first benchmark compares the results to a scheduler solely based on heuristics, and the other compares the result to a lower bound of the optimal solution. These comparisons provide insight into the performance of our proposed model. Furthermore, we discuss difficulties associated with solving this NP problem. Expressing the complications of computational complexity and its effects on our extension. This research not only advances the theoretical understanding and exploration of different useful techniques regarding job-shop scheduling but also provides practical tools and insights for optimizing production planning in the era of Industry 4.0.

Keywords: ILP, Optimization, Job-shop Scheduling Problem, GLPK, Time-indexed formulation, Extended Job-shop Scheduling Problem

Acknowledgements

We would like to thank our supervisor Gustav Johannesson at Recog, for arranging the project and providing us with guidance throughout the process of this project. We would also like to extend our gratitude to *The LEGO Group* for providing us with data.

Furthermore, we would like to thank our examiner Kristian Gustafsson at Chalmers.

Jesper Buske & Theodor Jendle, Gothenburg, May 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BnB	Branch-and-Bound
BnC	Branch-and-Cut
EJSSP	Extended Job-Shop Scheduling Problem
GLPK	GNU Linear Programming Kit
GS	Greedy Scheduler
ILP	Integer Linear Programming
JSSP	Job-Shop Scheduling Problem
LP	Linear Programming
MILP	Mixed Integer Linear Programming
NP	Nondeterministic Polynomial time

Contents

1
Introduction

1.1 Background

Production planning plays a crucial role in optimizing manufacturing processes, ensuring efficiency, and minimizing costs. Optimization comes with great benefits, especially during the advancement into Industry 4.0. Traditional approaches to production planning often rely on deterministic models, which may not capture the dynamic and stochastic nature of real-world production environments. This research proposes the application of *Integer Linear Programming*, ILP, a branch of optimization, to enhance production planning strategies by allowing systems to learn and adapt to changing conditions.

There is not any mathematical formulation of what the optimal schedule is that explains the real-world problem. The optimal schedule within manufacturing processes is affected by individual factors, such as how much storage the factory has or what rules the factory has on working hours per operator. To battle this undefined formulation most mathematical models assume that the quality of the schedule is determined by *idle time*, amount of machine downtime, or *makespan*, total completion time. This is a good approximation for exploring the mathematical problem but not as useful in the manufacturing world where this formulation takes too few factors into account. This paper will try to expand the mathematical models that already exist and make them more suited for real-world application.

1.2 Previous Work

In the field of Job shop scheduling problems the usage of ILP and *Mixed Integer Linear Program*, MILP, is not a new thought. For instance, Ríos-Mercado, et al. used a commercial *Branch-and-Cut*, BnC, solver to optimize the makespan of the Flow shop scheduling problem in their paper [1]. The NP-hard nature of the classic scheduling problem has led to the exploration and wide use of heuristics, metaheuristics, and matheuristics in combination with ILP and MILP [2], [3], [4].

One such example of using matheuristic is presented in a paper published by Guzman in 2021 [4]. This paper proposes a matheuristic algorithm to solve the *Job-Shop Scheduling Problem*, JSSP, combining a genetic algorithm with a disjunctive model of the JSSP. Guzman claims that the matheuristic algorithm reduces computational time and propels the problem towards efficiently finding solutions. Furthermore, the

model performs well on large instances of the JSSP, which is a recurring problem when solving the JSSP.

The paper [5] presents a generic ILP model for the flexible job-shop problem with the extensions; due dates, storage capacity, and product re-circulation, when some operations visit a machine or machine group more than once. Stating that the performance of their model was generally good on realistic examples, however expressing difficulties with solution efficiency as the complexity of the problem increases.

In [6], the authors investigate the Job-shop scheduling problem from another point of view by converting the classic ILP problem into a MILP. This reformulation unlocks the possibilities of utilizing, and taking advantage of, powerful methods such as BnC. BnC segments the solution space via cutting planes and relaxes the problem to create lower bounds of the problem, building upon the algorithm *Branch-and-Bound*, presented in section 2.3.1. Furthermore, formulation tightening of the problem is explored and proposed since the constraints constitutes the convex hull of a MILP problem, where they claim an exponential reduction of the complexity and accelerated convergence.

1.3 Problem Formulation

The goal of production planning is to optimize the scheduling of a set of orders. Every order consists of individual operations and the scheduling is done by assigning each operation to a machine and a time index, represented by the grid. The schedules are commonly visualized in the form of a Gantt scheme, Figure 1.1.

Each assignment of an operation comes with a set of rules that need to be fulfilled, so-called constraints. Those constraints are:

- An operation occupies a fixed time period and a started operation can not be interrupted. This means that an operation can not be split into smaller operations, we can not do half the operation today and the rest tomorrow.
- A machine can only perform one operation at a time, which means that the operations can not overlap in the schedule.
- Every operation will have a list of machine types that it can be operated on, this implies that one machine can not necessarily do all the operations.
- Finally, there is a precedence constraint that states; that operations within an order can require a sequence in which operations have to be completed prior to the next one can begin. An example of this relationship between operations is visualized in Figure 1.2.

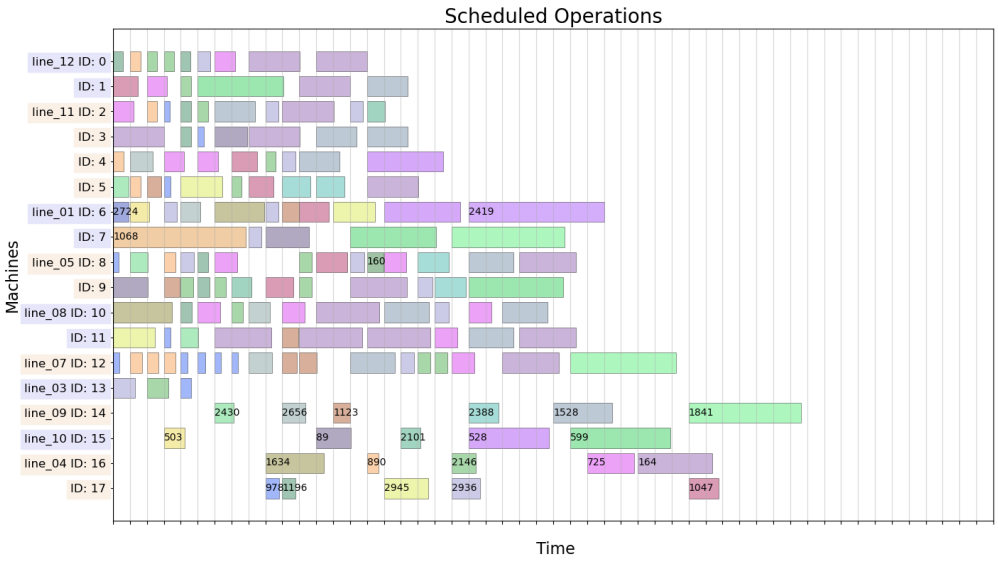


Figure 1.1: This is a visualization of a produced schedule. The rows of the schedule represent a machine, a block represents a scheduled operation, one color represents one order. The operations containing a number represents the final operation of the correspond order. The x-axis represents the time and the grid illustrates the time indices.

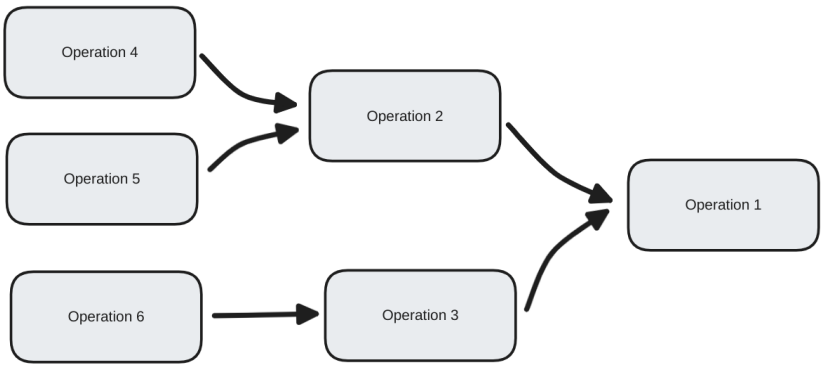


Figure 1.2: This is a representation that explains the relationships between operations within an order. Every box represents an operation, where the connecting arrows explain the logical order in which operations have to be done.

Because schedules are rather subjective, different companies may value different behaviors of a schedule, there are no distinct approaches to evaluating a schedule. In the field of production planning, there exist different problem descriptions that evaluate different scheduling behaviors or factors. This project will handle the following scheduling behaviors:

- *Makespan*, the total time required to process the schedule.
- *Lead Time*, the total time span of an order.
- *Number of Operators*, reducing the number of extra required operators which exceeds a predefined limit.

- *Earliness*, how long before the due date is the order completed.
- *Tardiness*, how long after the due date is the order done.

There are a lot more factors that could be taken into account, but the project chose to limit the problem with these schedule behaviors. The five factors listed above should also have the ability to be weighted differently in the program because the plan is to have multiple application areas for the program.

1.4 Purpose

The primary aim of this project is to develop and analyze an advanced ILP-based model, focusing on the complexities of modern manufacturing. This involves constructing an extended ILP formulation of the JSSP that incorporates additional real-world constraints and variables, using heuristics to reduce computational complexity, and evaluating the performance of the proposed model against benchmarks. This thesis includes the extended model of the JSSP, how well it performs in comparison to a lower bound of the theoretically best schedule and heuristic-based schedulers. This will advance both the theoretical understanding and practical application of ILP in production planning, providing a valuable technique for improving efficiency and competitiveness in the era of Industry 4.0.
Furthermore, the proposed model will not include all real-world intricacies of a manufacturing environment, such as material resources or machine failure.

2
Theory

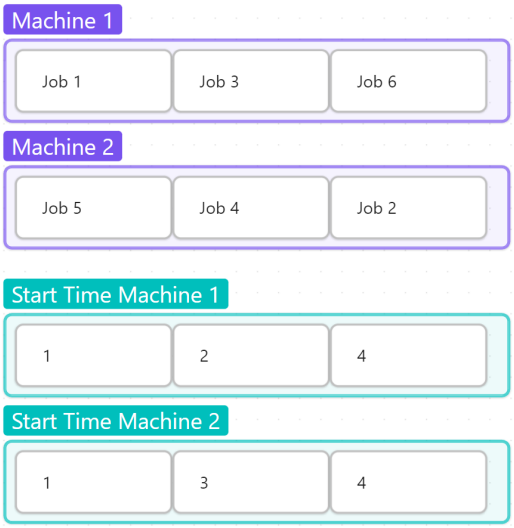
2.1 JSSP

The JSSP is a classic optimization problem in scheduling and operations research. In the JSSP, a set of jobs or orders must be processed on a set of machines. Each of these jobs contains a number of operations, where the operations have a processing time and a specified machine on which they will be processed. A machine can not process multiple operations in parallel and once an operation has started it can not be interrupted. Furthermore, the operations within a job have a sequence dependency, precedence, where certain operations require another operation to be completed prior to it's start of the process. The objective of JSSP is to determine the optimal scheduling of each job on each machine, subject to various constraints, to minimize the makespan. Additionally, the JSSP does not incorporate uncertainties such as machine breakdown nor does it take the absence of material into account.
Numerous mathematical models and formulations have been developed to represent and solve the JSSP. Due to the NP nature of the problem, these models need to capture the underlying characteristics of the problem whilst making it possible to utilize efficient solution algorithms. To tackle the computational complexity issue of JSSP, a wide range of metaheuristics and heuristics is commonly used. Examples of these heuristics are *Local Search Algorithms* - an iterative process of incremental changes to the initial schedule, *Ant Colony Optimization* - mimics the behavior of an ant colony exploration and exploitation, *Genetic Algorithms* - a population and evolution based algorithm to reflect the survival of the fittest [3].

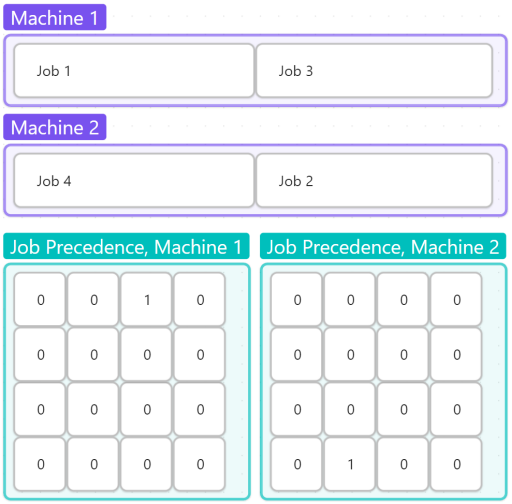
2.1.1 Representation of JSSP

There exist three commonly used mathematical representations of the JSSP, those being *Time-indexed formulation*, *Rank-based formulation* and *Disjunctive formulation* [2]. The rank-based representation targets the sequencing aspect of operations on each of the machines, using the ranks to indicate what order the operations should be processed, Figure 2.1a. This rank-based representation gives fewer variables but needs some help variables to construct constraints.
The disjunctive representation works similarly to the rank-based representation but changes the time representation from an array to a precedence matrix that describes the relationship between operations, stating the job on a machine once the preceding operations are done, Figure 2.1b. Additionally, this representation uses a set of disjunctive constraints, and affine conditions in combination with logical mathematical operators [7]. The most intuitive representation is the time-indexed formulation,

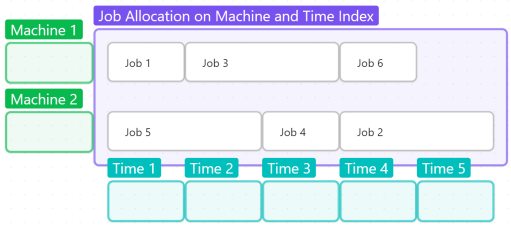
which utilizes a discrete timeline. This representation explicitly describes the positions of each job, both its position on the timeline as well as its assigned machine, Figure 2.1c. Moreover, there exist representations for the constraints that need to be fulfilled for each of the three mathematical formulations. These representations are rather similar to one another, with slight adjustments to be compatible with their representations. That gives them a difference in their complexity scaling depending on what constraint and objective function should be used.



(a) Visual representation of the Rank-based formulation of JSSP. The positions within a machines array indicate the sequence in which the operations are to be executed, whilst the start time array represents the operations start time instance.



(b) Visual representation of the Disjunctive formulation of JSSP. The positions within a machine's array indicate the sequence in which the operations are to be executed, whilst the precedence matrix represents the operations precedence relationship between operations.



(c) Visual representation of the Time-Indexed formulation of JSSP. Each operation assignment is explicitly illustrated in a 3-dimensional matrix consisting of a machine, operation, and time axis. This representation is the most intuitive of the bunch, but also the least compact.

Figure 2.1: Visual representations of the three JSSP formulations; Time-indexed, Rank-based, and Disjunctive formulation.

In Table 2.1, a comparison of the number of decision variables and constraints between the three mathematical formulations is shown. Both the collocation of the three mathematical formulations and comparisons were made by W.-Y. Ku, J.C. Beck [2].

Model	#Variables	#Constraints
Time-Indexed	$\|\mathbb{M}\| \ \|\mathbb{J}\| \ \|\mathbb{T}\| + 1$	$\|\mathbb{M}\| \ \|\mathbb{T}\| + 3\|\mathbb{M}\| \ \|\mathbb{J}\| - \|\mathbb{J}\|$
Rank-Based	$\|\mathbb{M}\| \ \|\mathbb{J}\|^2 + \|\mathbb{M}\| \ \|\mathbb{J}\|$	$\|\mathbb{M}\| \ \|\mathbb{J}\|^3 + 4\|\mathbb{M}\|\|\mathbb{J}\| + \|\mathbb{M}\|$
Disjunctive	$\|\mathbb{M}\| \ \|\mathbb{J}\|^2 + \|\mathbb{M}\| \ \|\mathbb{J}\| + 1$	$\|\mathbb{M}\| \ \|\mathbb{J}\| \ \|\mathbb{T}\|$

Table 2.1: Comparison of the ILP models for the JSSP. $|\mathbb{M}|$, $|\mathbb{J}|$, and $|\mathbb{T}|$ denote the number of machines, the number of jobs, and the number of total time indices, respectively.

A comparison between the three models, the Time-Indexed Model explicitly assigns each operation at a point in time and machine, resulting in a complex method due to the vast amount of variables and constraints. The disjunctive model combines disjunctive constraints to compactly express precedence relationships and sequencing. The rank-based model focuses on the sequencing within a machine.

2.1.2 EJSSP

The JSSP does not cover the majority of real-world features of a manufacturer, for example, due dates and a non-predetermined process sequence of the operations within orders are not considered. Therefore, in this project an extended version of the JSSP will be considered, EJSSP. The reason for this is to reflect a more realistic representation of the production chain within a manufacturing facility. The EJSSP has a trade-off between computational complexity and flexibility/applicability, in comparison to the traditional JSSP. In the EJSSP the following modifications have been implemented:

- Extension of operations - There exists no predetermined processing sequence within an order/job, instead this extended version aims to optimize the scheduling of each operation rather than the scheduling of jobs.
- Extension of precedence - Each operation has a set of other operations that need to be completed before the operation. This set may be empty.
- Extension of machine requirement - Each operation does not have a specified machine to be processed on, instead it has a set of machines on which it can be processed.
- More nuanced schedule behavior - The objective function captures a more complex scheduling behavior, where for example due dates and the number of operators are incorporated/considered. As mentioned previously, it is hard to mathematical and practically define a good schedule.

Similarly to JSSP, multiple formulations of the problem can be adopted to represent EJSSP, however in this project, a formulation inspired by the time-indexed representation will be used. Moreover, since the previously mentioned heuristics and meta-heuristics are dependent on the objective function, and since an ideal objec-

tive function is yet to be implemented, the heuristics are not as effective as their counterpart in the JSSP. Thus new heuristics need to be implemented and utilized.

2.1.3 Other Scheduling Problems

Since the JSSP does not reflect all scheduling problems, other definitions of scheduling problems with their characteristics have been defined. Examples of these are the *Flow Shop-, Open Shop* and *Flexible Job-Shop Scheduling*. The flow shop scheduling problem is a problem where jobs pass through a predefined machine sequence, where each job consists of the same operations. In the open shop scheduling problem, the precedence within a job remains but there exists no machine requirement. Thus, every operation can be processed on any of the machines. Finally, the problem description most similar to that of the EJSSP is the flexible job-shop scheduling problem, introducing operations to a set of machines on which they can be processed. All scheduling problems differ from one another but share a common goal of optimizing the allocation of operations to achieve efficient scheduling outcomes.

2.2 Optimization and Linear Programming

Optimization is the process of selecting the most valuable or preferable outcome of a problem, from a set of available selection alternatives. A selection is defined by assigning a value to the variables of the problem. These selections are called feasible solutions and are limited by certain criteria or rules, so-called constraints. Geometrically the constraints delineate the boundaries of a geometric shape that corresponds to the feasible solution space. The value of a selected element in the feasible solution space is measured by an objective function that captures the underlying features of the problem. The most preferable outcome of the problem is the selection corresponding to the minimum of the objective function, or maximum depending on how the problem is structured.
Linear programming, LP, also known as linear optimization, is a mathematical modeling technique used for the optimization of a problem. As the name suggests, LP is modeled solely by linearity, depicting complex relationships through linearity [8]. Similar to general optimization, LP models are constructed by constraints, an objective function, and non-negative variables called *decision variables*, and additionally the model is specified by parameters. The parameters can be seen as the fixed data of the model, such as a specified capacity. Furthermore, LP models require each constraint and the objective function to be linear in order to utilize mathematical concepts such as *Convexity*. If a problem is convex, then a local minimum is also the global minimum. A problem is convex if the geometric shape of the feasible solution space is a *Convex Set* and the objective function is a *Convex Function*.
In Figure 2.2 an illustration of a convex and non-convex set is presented. In Figure 2.2 an illustration of a convex and non-convex set is presented.
The definition of a convex set C says that the line segment between any two points in the set lies in C, and is mathematically expressed as:

$$\lambda x_1 + (1 - \lambda)x_2 \in C, \quad \forall x_1, x_2 \in C, \ \forall \lambda \in [0, 1] \tag{2.1}$$

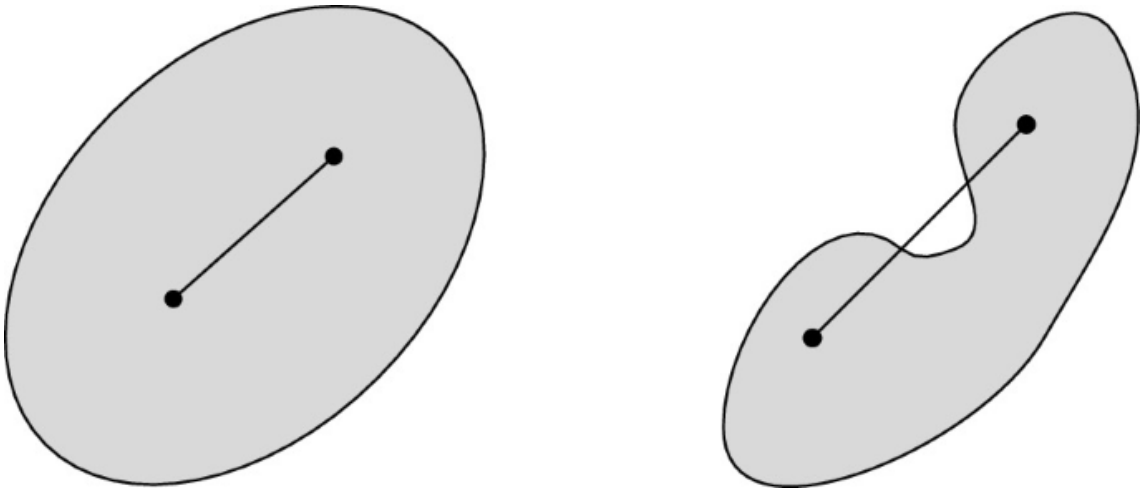


Figure 2.2: The left set illustrates an example of a convex set and the right set illustrates a non-convex set. The picture was taken from [9].

A convex function is defined by a convex domain where the line segment between any two points resides above the curve of the function between them. The convexity is usually shown by utilizing a theorem that states that [10]:

$f(\mathbf{x})$ is convex if and only if it is defined in a convex domain and its Hessian is positive semi-definite:

$$H(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \succeq 0 \tag{2.2}$$

Deriving that the objective function is a convex function and that the feasible solution space, delineated by the constraints, is a convex set is needed in order to prove the convexity of LP. Starting with the objective function, since the second order derivative of any linear function is equal to 0, $f(x) = ax + b \implies f'(x) = x \implies f''(x) = 0$, the objective function is convex. Regarding the feasible region, *Hyperplane* and *Halfspace* need to be defined. Hyperplane is a generalization of a plane, flat surface, in an arbitrary dimension $\mathbb{R}^n$ and is mathematically defined as $\{\mathbf{x} : \mathbf{p} \cdot \mathbf{x} = k\}$ where $\mathbf{p}$ is a non-zero vector in $\mathbb{R}^n$ and k is a scalar. Halfspace is either of the two spaces divided by a hyperplane in $\mathbb{R}^n$ and is defined as $\{\mathbf{x} : \mathbf{p} \cdot \mathbf{x} \leq k\}$ or $\{\mathbf{x} : \mathbf{p} \cdot \mathbf{x} \geq k\}$. Convexity of a halfspace is easily verified. The boundary of a constraint in LP is a hyperplane, where the corresponding feasible region is one of the halfspaces created. By continuously adding and combining all the constraints in LP and their corresponding feasible halfspace, a convex intersection of the halfspaces is constructed, Figure 2.3.
This created intersection is the feasible solution space of the model, and the geometric shape is called a *Polyhedral Set* defined as $\{\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ [12]. With the above

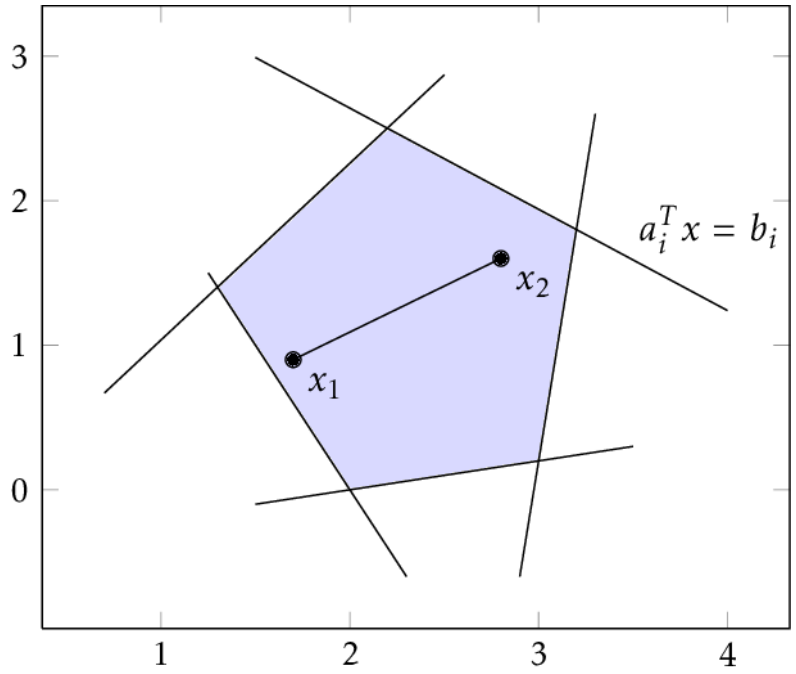


Figure 2.3: Illustration of intersecting halfspaces. The picture was taken from [11].

established, an LP may be represented by its standard form:

$$\begin{aligned} \text{Minimize } z &= \mathbf{cx} \\ \text{subject to } \mathbf{Ax} &\leq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

where $\mathbf{c}$ and $\mathbf{b}$ are defined vectors, $\mathbf{A}$ a defined matrix and $\mathbf{x}$ is the decision variables. The equality $z = \mathbf{cx}$ corresponds to the objective function of the LP, whilst $\mathbf{Ax} \leq \mathbf{b}$ corresponds to the feasible region, the polyhedral set.

2.3 ILP

ILP is a subcategory of linear programming, regarding optimization problems where the decision variables are required to be integers and the constraints as well as the objective function are all linear similar to LP [13]. In contrast to linear programming, ILP requires the decision variables to be integers which for some optimization problems makes it easier to formulate mathematically, but at the expense of increased computational complexity due to the increased number of constraints. ILP problems arise in various fields of engineering, and other professions, where discrete decisions need to be made such as territorial partitioning, cellular networks, and cash flow matching. The mathematical standard form of an ILP is structured as

follows:

$$\text{Minimize } z = \mathbf{cx} \tag{2.3}$$

$$\text{subject to } \mathbf{Ax} = \mathbf{b} \tag{2.4}$$

$$\mathbf{x} \in \mathbb{Z}_{\geq 0} \tag{2.5}$$

where $\mathbf{c}$ and $\mathbf{b}$ are defined vectors, $\mathbf{A}$ a defined matrix and $\mathbf{x}$ is the decision variables with a non-negative integer, $\mathbb{Z}_{\geq 0}$, requirement. The equality $z = \mathbf{cx}$ corresponds to the objective function of the LP, whilst $\mathbf{Ax} = \mathbf{b}$ corresponds to the feasible region, the polyhedral set [14]. To formulate a LP problem as an ILP some tricks exist by creating help variables, or artificial variables, that give some variables the ability to describe a nonlinear behavior. However adding decision variables vastly affects the computational complexity of a model, thus adding unnecessary decision variables should be avoided. Another trick utilized by LP, ILP, and MILP is the *Big M method* which in essence creates a high boundary, this can be used to remove part of the solution space by giving it a high penalty to remove it from logical solutions. The Big M method may be required in some instances to construct a constraint correctly [15].

2.3.1 ILP Solver, GLPK

There are different ways to mathematically solve an ILP problem, all having different strengths and weaknesses. In their essence, these solvers explore the solution space of the constructed problem and locate close-to-optimal solutions. Due to the existence of different solvers, one could use these to utilize different solving methods. Some of these solvers are Gurobi [16], CPLEX [17], and GLPK [18], where this paper will focus on GLPK because it is an open-source solver compared to Gurobi and CPLEX which are commercial solvers. GLPK utilizes the algorithm Branch-and-Bound in combination with *Simplex Method* and *Long-Step Dual Simplex*. Branch-and-Bound is a combinatorial algorithm that partitions the solution space into feasible convex sets, branches, and extracts a local upper and lower bound, bounding the space by using convex relaxation, duality, Lipschitz, or other techniques. From these bounds, global upper and lower bounds are defined which for instance are used to eliminate branches with poor performance. Elimination is determined by comparing the local lower bound to the global upper bound. The branching is repeated until a convergence criteria, based on the difference between the bounds, is met. The complexity of this algorithm is at worst of an exponential nature [19]. Regarding *Linear Programming*, LP, the feasible solution space of a LP is a convex polytope (convex set containing vertices or extreme points), and due to the properties of convex regions, if there exists an optimal solution to the LP, it will be on one of the vertices or extreme points of the polytope. The constraints of the LP constitute the edges or boundaries of the convex polyhedral, where the vertices of the polyhedral correspond to the extreme points. The simplex method follows the edges of the polytope in the direction of the objective function and investigates the reached vertices. This process is iteratively repeated until optimality is met. In the case of an unbounded edge being visited, it is implied that the problem has no optimal solution since the objective value will be improved infinitely. When it comes to the efficiency (amount

of iterations) of the simplex method, the worst-case performance is equal to the total number of vertices, $\binom{n+m}{n}$, where n is the number of variables and m the number of constraints delineating the polyhedral. Due to the fact that the method moves from one vertex to another without ever returning to a previously visited vertex. On the other hand, the average performance is approximately $0.517(m+n)^{1.049}$ [20].

In the instance of applying the Simplex Method to an ILP, the method is used in combination with the relaxation of the model, converting it to a LP. By utilizing the simplex method, tight lower bounds to the problem can be located.
Lastly, the Long-Step dual simplex, a technique applied to LP used for re-optimization when relaxed constraints are added or fixed variables. This method uses the LP's *dual*, another LP derived from the original one, and applies a similar methodology to that of the simplex method. This is useful since by solving the dual, the primal also becomes solved [21],[22].

2.4 Difficulties of scheduling

The reason why these techniques are applied to the problem at hand is due to the NP-problem nature of the problem at hand. NP problems are defined by their computational difficulty, where no efficient solution algorithm has been developed. This means that the problems can not be solved in polynomial time, so-called non-deterministic polynomial time. Problems that explode exponentially usually belong to this class of computational problems. Some other known NP problems include the Traveling Salesman Problem and the Knapsack Problem [23]. The Traveling Salesman Problem has the objective of finding the shortest route given a set of locations that need to be visited and then returning to the start location. The number of solutions to this problem increases factorially, $\mathcal{O}(n!)$, thus brute-forcing the solution becomes impractical at $n = 20$ locations. The Knapsack Problem centers around selecting the most valuable items given a capacity, with the number of possible solutions being exponential, $\mathcal{O}(2^n)$.

Another way to combat the difficulties of NP-problems is the use of heuristics, metaheuristics or matheuristics. Heuristics are a valuable and practical technique usually embedded into the solution methods of NP-problems [24]. Heuristics acts as a rule of thumb, where the approach of solving the problem is not optimized nor produces the optimal solution, but rather an adequate approximation of the solution. The benefits of heuristics are their easy implementation as well as efficiency regarding the time complexity of the problem. Furthermore, heuristics can be used for benchmarking the performance or computational efficiency of a solution method [25]. Metaheuristics, on the other hand, are more nuanced heuristics that efficiently explore a broad extent of the solution space to find optimality. This category of heuristics usually takes inspiration from natural processes, with some examples being *Particle Swarm Optimization*, *Genetic Algorithms* and *Ant Colony Optimization*. Metaheuristics are widely used and a powerful tool when dealing with complex optimization problems in various fields [26],[27]. Finally, matheuristics are

a hybrid optimization method that combines the use of metaheuristics and mathematical programming. This type of heuristics is not as commonly used as the two other types of heuristics mentioned above [28].

3
Method

In this project, the intention is to employ ILP as a primary optimization technique. However, recognizing the computational complexity associated with ILP-based methodologies, heuristics are incorporated into the approach. Heuristics, being efficient and practical problem-solving strategies, provide a valuable complement to ILP by generating near-optimal solutions in a reasonable amount of time. This hybrid approach aims to balance the trade-off between computational feasibility and solution quality. The motivation behind combining ILP and heuristics lies in the need for comprehensive and robust solutions to the Job-shop Scheduling Problem. ILP offers optimality guarantees, while heuristics provide scalability and flexibility in handling large-scale instances and dynamic manufacturing environments. In the subsequent sections, we describe the data, heuristics to be employed, and our proposed methodology. Finally, this chapter contains the mathematical formulation of the EJSSP.

3.1 Data

The data used in this paper was provided by the company *The LEGO Group*, where the data sent was modified and anonymized due to a nondisclosure agreement, thus the data was not extracted from any real-life production chain. The provided data include the operations of 3000 orders. Each operation has the listed information:

- *Execution Time* - time needed to finish the operation, length of the operation.
- *Precedence* - which operation that cannot be started prior to the finish of this operation.
- *Machine Availability* - which machine types the operation can be processed on.

Additionally the data was for a model with 10 different machine types and a total of 18 machines distributed over the machine types. In the data provided due dates were not present and therefore uniformly generated and manually adjusted. This also means that the due dates do not reflect a real production.

The results will be presented from a data set with 25 randomly selected orders, of varying sizes. Containing a total of 156 operations with an average of 7.3 operators per operation.

3.2 Algorithm

As stated in section 2.3.1, the plan is to utilize an ILP-solving package, GLPK, to solve the modeled ILP problem. Due to the complexity of the NP scheduling problem, it is necessary to scale down the solution space fed into the ILP-solver with heuristics. This down scaling is done by not allowing all operations to be optimized at the same time. To minimize the bias of the first-placed operations, the plan is to start with a sparse feasible schedule. The operations position will then iteratively be improved by locking all the operations in place and then unlocking small parts of the schedule and optimizing this subspace. The building-blocks of the down-scaling will be explained in the three following sections; 3.2.1, 3.2.2 and 3.2.3.

3.2.1 Unlocking Operations

By unlocking a set of operations in the schedule, only a fraction of the solution space is considered when solving the ILP, which in turn reduces the complexity. Formulating the unlocked operations as variables and the locked operations as parameters makes it so the solver is only allowed to optimize the unlocked operations, Figure 3.1 for a visualization.

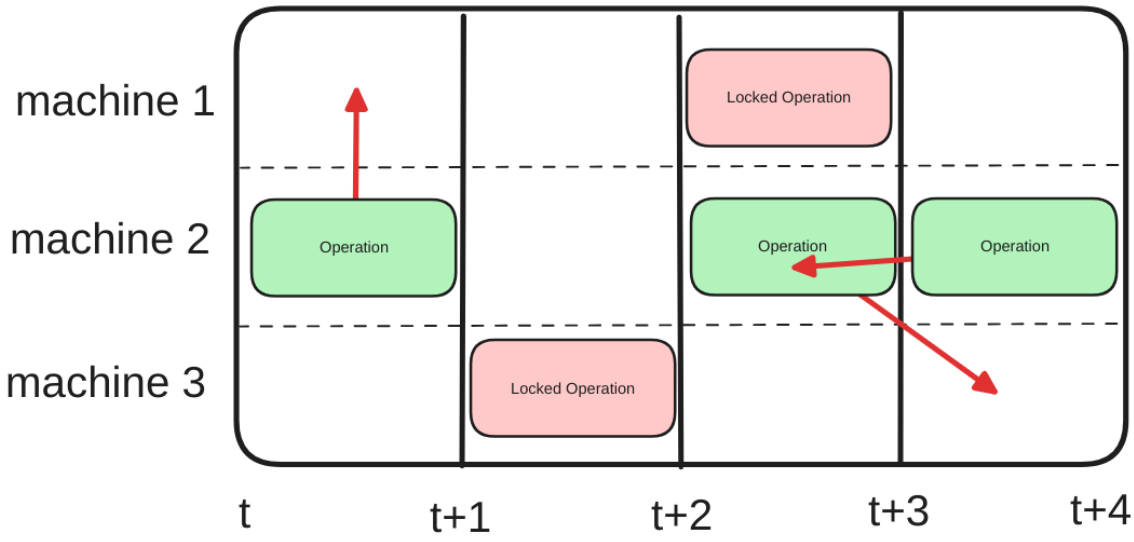


Figure 3.1: Illustrates the unlocking of operations where the red boxes represent locked operations and green unlocked operations. The red arrows indicates changes after an ILP-solve.

Unlocking fewer operations comes with the cost of considering less of the solution space. Unlocking more operations comes with the cost of increasing the complexity. Thus finding a good amount of operations to unlock in each solve, is relevant. The complexity is approximated to depend on more than the amount of unlocked operations and will be explained in section 3.2.1. The unlocked operations will be grouped together in orders, which means that every ILP-solve will have either all operations from one or two orders unlocked. Allowing the solver to optimize with the precedence in mind.

3.2.2 Time Divide

The schedule will be divided into time indices, partitioning the timeline into equidistant time intervals. Depending on the length of the time intervals an operation can expand more than one time interval, where the execution time of the operations will be rounded up to their closest time index. This leads to some operations being represented as bigger than what they actually are, see first operation in Figure 3.2. Larger time indices reduces the number of variables in the ILP-model and thereby lowering the overall complexity. Smaller time intervals have the benefit of representing different lengths of operations more accurately, thereby enabling more optimized schedules. To get the best of both worlds, the algorithm will first optimize larger time units and throughout the optimization process divide each time unit into two, Figure 3.2. When a time division happens, the length and position of each operation is update accordingly, doubling the time index and representing the length in terms of the new time intervals.

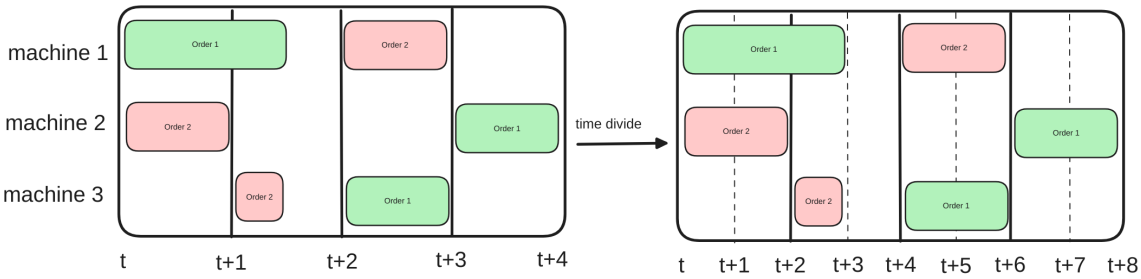


Figure 3.2: Shows an example of a schedule with large time indices followed by the same schedule with each time interval divided in two.

The intent of starting with large time intervals is to lower the complexity when operations are positioned further from their optimal placement and *Limiting Time Periods* should preferably not be used. *Limiting Time Periods* is explained in next section 3.2.3. How many time divides each optimization process should do, will be set to four in all results presented in this report. The initial distance between two time indices is set to the largest operation of the schedule.

3.2.3 Limiting Time Periods

To further reduce the complexity of each solve, a chain of connected time intervals will be solved at a time, Figure 3.3 for a visualization.

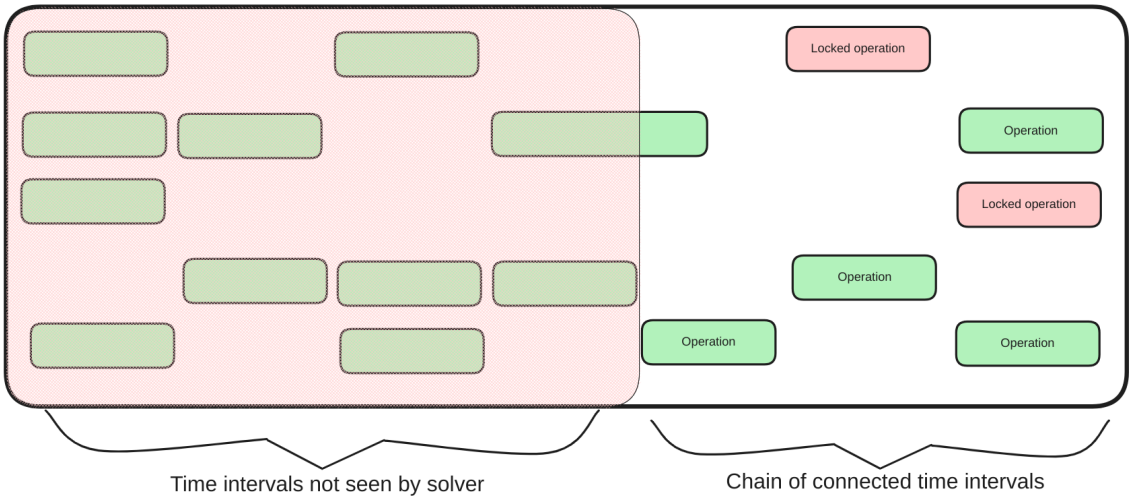


Figure 3.3: Shows an illustration of a schedule with the large red part of the schedule not optimized by the solver.

To avoid breaking constraints, operations that intersects with a time interval endpoint is interpreted as locked with a length modified to not exceed the chain of connected time intervals, Figure 3.4.

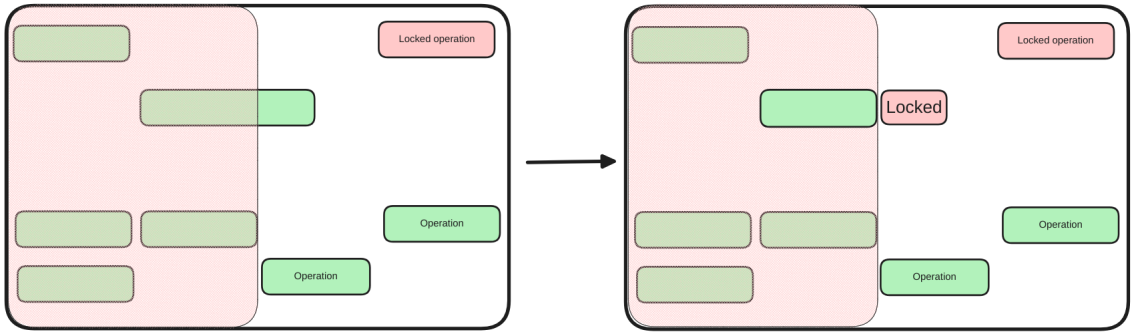


Figure 3.4: Highlight the time interval endpoint of the same illustration as Figure 3.3. Where one operation gets split in two and locked.

Note that the first optimizations should preferably be done without any limiting time periods, due to operations not necessarily being close to their optimal placement. This is the motivation to have large time steps in the beginning of the solve as mentioned in section 3.2.2.

3.2.4 Complexity Approximation

From sections 3.2.1, 3.2.2 and 3.2.3 there are options to reduce the complexity. A time complexity approximation was made to calibrate the complexity of the individual solve. The approximation is based on the assumption that the time complexity grows proportionately to the amount of possible solutions of the sub-problem:

$$t \propto |\mathbb{U}|! \cdot \binom{|\mathbb{M}||\mathbb{T}|}{|\mathbb{U}|} \tag{3.1}$$

Where $|\mathbb{M}|$ is the number of machines, $|\mathbb{U}|$ is the number of unlocked operations, and $|\mathbb{T}|$ is the number of time indices. Equation (3.1) says that the computational time is proportional to how many ways you can distribute $|\mathbb{U}|$ operations on $|\mathbb{M}||\mathbb{T}|$ placements. This approximation does not consider all constraints, such as precedence. Since the value obtained from (3.1) reflects the number of solutions, a maximum number of solutions can be defined. In the presented results the threshold for maximum number of solutions was empirically set to $8.5 \cdot 10^{20}$. The program will unlock two orders and if this leads to a larger number of solutions than the threshold, then it will be split into multiple smaller runs. This splitting will be explained in next section 3.2.5

3.2.5 Time Reduction

If one solve is too complex to run at once, the solve will be split into multiple smaller ones. This is done by limiting the time periods sent to the solver, limiting time periods is explained above. The amount of time indices sent to the solver will be adjusted so that the number of solutions is just under the threshold. Starting from the end of time interval a sub-section is created and solved. The next sub-section will be created from the middle of the last sub-section. This is an iterative process that ends once the beginning of the schedule is solved, Figure 3.5.

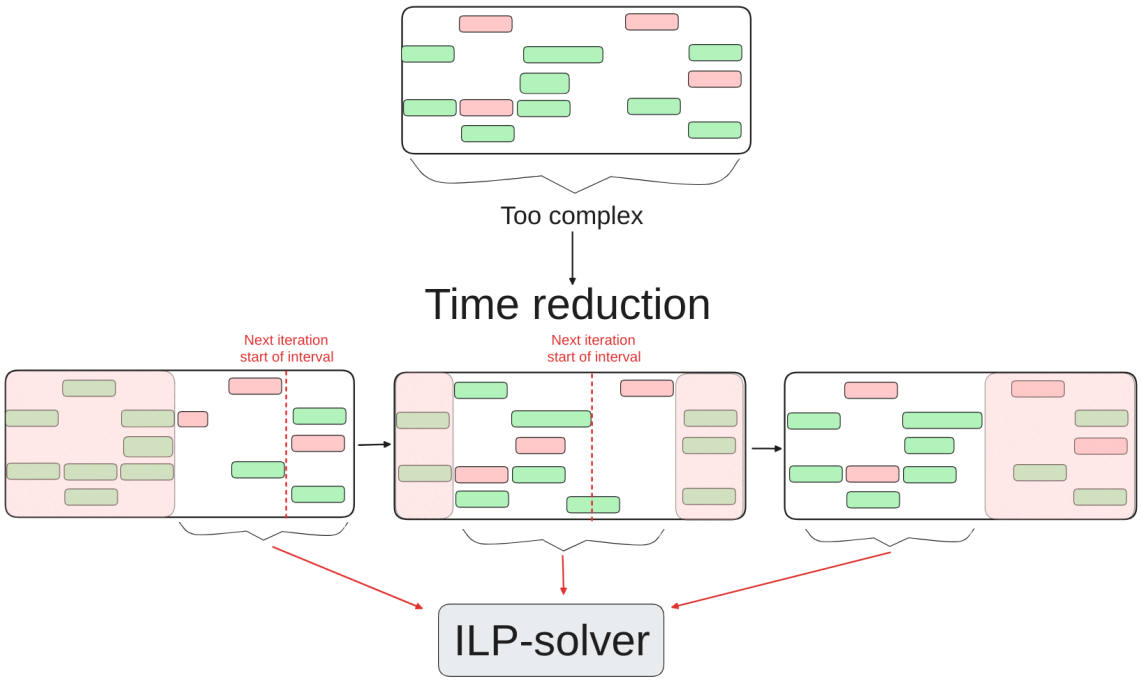


Figure 3.5: Illustration of Time Reduction. Splits the chain of connected time intervals into smaller segments and solves the ILP instances.

3.2.6 Scheduler

The scheduler will unlock two orders at a time and run the full time interval of the schedule. But if the complexity of one unlock is too large, then it will be split into

multiple runs, according to time reduction. For more information the main parts of the scheduler is presented as pseudo code:

Algorithm 1 Algorithm used to schedule

```
for #Divides do
    divide timeline
    pairs ← randomly split list of orders in groups of 2
    for pair in pairs do
        unlock, lock ← data from unlock procedure
        estimate computational complexity
        if problem is too complex then
            perform time reduction
        else
            ilp_data ← convert unlock, lock to required format
            ilp_solution ← ilp(ilp_data)
                                        ▷ Passes the ILP data to the model
            if ilp_solution is an improvement then
                schedule ← ilp_solution
            end if
        end if
    end for
end for
```

3.3 ILP Mathematical Formulation

ILP is a mathematical modeling approach that formulates the problem as a set of linear equations, constraints, and objective functions. The ILP model for the EJSSP utilizes the following sets, variables, and parameters:

Sets

- $\mathbb{M} = \{1, .., \#Machines\}$ - All machines within the manufacturing system.
- $\mathbb{U} = \{1, .., \#Unlocked\ Operations\}$ - All unlocked operations within the given time interval.
- $\mathbb{L} = \{1, .., \#Locked\ Operations\}$ - All locked operations within the given time interval.
- $\mathbb{T} = \{1, .., \#Time\ Indices\}$ - All time indices within the given time interval.
- $\mathbb{O} = \{1, .., \#Orders\}$ - All orders within the given time interval.

Variables

- $x_{mut} = \begin{cases} 1, & \text{If operation } u \text{ is scheduled to start at time index } t \text{ on machine } m \\ 0, & \text{Otherwise} \end{cases}$

  A binary variable where $m \in \mathbb{M},\ u \in \mathbb{U},\ t \in \mathbb{T}$. This variable is initialized to the state of the schedule prior to the current optimization. This helps with an initial start value for the solver, which potentially saves time on a solution, but is not necessary to get the best result.

- $s_o \in \mathbb{N}$ - An Orders long array of integer variables. Help variables that represent the time step of each order's first placed operation. It is used by the objective function to compute the lead time.

- $m \in \mathbb{N}$ - A natural valued help variable that represents maximum operators over the inputted "max operators" weight. Note that this variable is defined so it can not be negative, this is to construct a nonlinear function.

- $n_t \in \mathbb{N}$ - An array of natural valued help variables that represent maximum operators over the inputted "max operators" weight, this at each time index, $t \in \mathbb{T}$. Note that this is the same as $m \in \mathbb{N}$ but for each time index instead of the maximum.

- $e_o \in \mathbb{N}$ - An array of natural valued help variables that represent the earliness of each order, $o \in \mathbb{O}$. This variable is used to make it possible to weigh earliness and tardiness differently.

Parameters

- $V^{um}, \quad u \in \mathbb{U},\ m \in \mathbb{M}$ - Each operations' valid machines, 1 if operation u can be processed on machine m, 0 otherwise.

- $P^{u\hat{u}}, \quad u, \hat{u} \in \mathbb{U}$ - Precedence between operations u and $\hat{u}$. 1 if u needs to be completed prior to the start time of $\hat{u}$, 0 otherwise.

- $B^{ul}, \quad u \in \mathbb{U},\ l \in \mathbb{L}$ - Precedence between operation u and locked operation l. 1 if u needs to be completed prior to the start time of l, 0 otherwise.

- $A^{lu}, \quad l \in \mathbb{L},\ u \in \mathbb{U}$ - Precedence between locked operation l and operation u. 1 if l needs to be completed prior to the start time of u, 0 otherwise.

- $E^u, \quad u \in \mathbb{U}$ - Process time of operation u in number of time indices.

- $LE^l, \quad l \in \mathbb{L}$ - Process time of locked operation l in number of time indices.

- $S^{mlt}, \quad m \in \mathbb{M},\ l \in \mathbb{L},\ t \in \mathbb{T}$ - Locked schedule, 1 if locked operation l is scheduled to start at time index t on machine m, 0 otherwise.

- $F^o, \quad o \in \mathbb{O}$ - Denotes whether the first operation of an order is contained within the time interval $\mathbb{T}$. 1 if the operation is within the time interval, 0 otherwise.

- $L^o, \quad o \in \mathbb{O}$ - Denotes the index corresponding to the last operation of the order o. 0 if the operation is not included in $\mathbb{U}$.

- $O^{uo}, \quad u \in \mathbb{U},\ o \in \mathbb{O}$ - Denotes whether the operation u belongs to the order o. 1 if the operation belongs to the order, 0 otherwise.

- $LO^{lo}, \quad l \in \mathbb{L},\ o \in \mathbb{O}$ - Denotes weather the locked operation l belongs to the order o. 1 if the operation belongs to the order, 0 otherwise.
- $UW^{u}, \quad u \in \mathbb{U}$ - Number of operators required to process the unlocked operation u.
- $LW^{l}, \quad l \in \mathbb{L}$ - Number of operators required to process the locked operation l.
- $D^{o}, \quad o \in \mathbb{O}$ - The due date of the order o, rounded down to the closest time index t.
- MS^{b} - Scalar for the makespan, used to balance the objective function.
- LT^{b} - Scalar for the lead time, used to balance the objective function.
- OX^{b} - Scalar for the maximum amount of operators, used to balance the objective function.
- OM^{b} - Scalar for the mean amount of operators, used to balance the objective function.
- E^{b} - Scalar for the earliness, used to balance the objective function.
- T^{b} - Scalar for the tardiness, used to balance the objective function.
- MS^{w} - factor for the makespan, used to weigh the objective function.
- LT^{w} - factor for the lead time, used to weigh the objective function.
- O^{*} - the preferred amount of operators
- OX^{w} - factor for the extra amount of operators, used to weigh the objective function.
- E^{w} - factor for the earliness, used to weigh the objective function.
- T^{w} - factor for the tardiness, used to weigh the objective function.

Objective Function
The objective function is split into five separate/independent weighted terms corresponding to the behaviors - makespan, lead time, amount of operators, earliness, and tardiness respectively.

$$\min \Big(MS^{w} \cdot ms + LT^{w} \cdot lt + OX^{w} \cdot ox + (E^{w} + T^{w}) \cdot earl + T^{w} \cdot tard \Big) \tag{3.2}$$

where

$$ms = MS^b \sum_{m\in\mathbb{M}} \sum_{u\in\mathbb{U}} \sum_{t\in\mathbb{T}} t \cdot x_{mut} \tag{3.3}$$

$$lt = LT^b \sum_{o\in\mathbb{O}} \Big(- F^o s^o + \sum_{m\in\mathbb{M}} \sum_{t\in\mathbb{T}} x_{m,L^o,t}(t + E^{L^o}) \Big) \tag{3.4}$$

$$ox = OX^b \cdot m + OM^b \sum_{t\in\mathbb{T}} n_t \tag{3.5}$$

$$earl = E^b \sum_{o\in\mathbb{O}} e_o \tag{3.6}$$

$$tard = T^b \sum_{o\in\mathbb{O}} \Big(- D^o + \sum_{m\in\mathbb{M}} \sum_{t\in\mathbb{T}} x_{m,L^o,t}(t + E^{L^o}) \Big) \tag{3.7}$$

Depending on the weights of the model, the ILP minimizes each term or a combination of them. Note that the matrices within the summations of equations (3.3) through (3.7), corresponds to the variable-matrix $\mathbf{x}$ mentioned in equation (2.3), whilst the rest constitutes the vector $\mathbf{c}$.

Constraints
Constraints (3.8) through (3.15) are inspired from: [4], converted from the disjunctive model to time-indexed model.

1. Ensures that all operations are scheduled exactly once.

$$\sum_{m\in\mathbb{M}} \sum_{t\in\mathbb{T}} x_{mut} = 1, \quad \forall u \in \mathbb{U} \tag{3.8}$$

   Searches through every machine and time instance to confirm that the allocation of an operation happens once (equal to 1 at one index and 0 on the rest). This is performed for every operation.

2. Constraint for scheduling an operation on one of its valid machines.

$$\sum_{t\in\mathbb{T}} x_{mut} \leq V^{um}, \quad \forall m \in \mathbb{M}, \forall u \in \mathbb{U} \tag{3.9}$$

   Checks for every time instance if an operation is allocated on a machine that is not allowed to be executed on and if so prohibits that scheduling. This constraint is instantiated for every operation and machine.

3. Constraints to disallow parallel processing within unlocked operations and between unlocked and locked operations, respectively.

$$\sum_{u\in\mathbb{U}} \sum_{t'\in\mathbb{T}'} x_{mut'} \leq 1 + M \cdot (1 - x_{mut}), \quad \forall m \in \mathbb{M}, \forall u \in \mathbb{U}, \forall t \in \mathbb{T} \tag{3.10}$$

$$\sum_{u\in\mathbb{U}} \sum_{t''\in\mathbb{T}''} x_{mut''} \leq M \cdot (1 - S^{mlt}), \quad \forall m \in \mathbb{M}, \forall l \in \mathbb{L}, \forall t \in \mathbb{T} \tag{3.11}$$

$$\sum_{l\in\mathbb{L}}\sum_{t'\in\mathbb{T}'} S^{mlt'} \leq M\cdot(1-x_{mut}), \quad \forall m\in\mathbb{M}, \forall u\in\mathbb{U}, \forall t\in\mathbb{T} \tag{3.12}$$

where $\mathbb{T}' = \{t, .., t+E^u\}$ and $\mathbb{T}'' = \{t, .., t+LE^l\}$.
Loops through the interval of an operation's start to its finish time and confirms that no other operation, locked or unlocked, is assigned to start within that time frame.

4. Precedence constraints within the unlocked operations.

$$\sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}} t\cdot x_{m\hat{u}t} \geq P^{u\hat{u}} \sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}}(t+E^u)\cdot x_{mut}, \quad \forall u,\hat{u}\in\mathbb{U} \tag{3.13}$$

Given two different operations, loops through each machine and time instance to locate the start time of the first operation, and the time instance when the other operation has been fully processed. If and only if there is a precedence relationship between the two operations, the constraint ensures that the start time of the first operation is located at a later time instance than the finished time of the other operation. This is done for each combination of two operations.

5. Precedence constraints between locked and unlocked operations.

$$\sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}} t\cdot S^{mlt} \geq A^{ul} \sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}}(t+E^u)\cdot x_{mut}, \quad \forall u\in\mathbb{U}, \forall l\in\mathbb{L} \tag{3.14}$$

$$\sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}} t\cdot x_{mut} \geq B^{lu} \sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}}(t+LE^l)\cdot S^{mlt}, \quad \forall u\in\mathbb{U}, \forall l\in\mathbb{L} \tag{3.15}$$

This constraint is similar to, number 4, with the distinction that one of the operations is locked and either needs to be executed before or after the unlocked operation.

6. Spillover constraint.

$$\sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}} x_{mut}(t+E^u) \leq \mathbb{T}_{|\mathbb{T}|}, \quad \forall u\in\mathbb{U} \tag{3.16}$$

where $\mathbb{T}_{|\mathbb{T}|}$ denotes the last element of the time interval $\mathbb{T}$.
Checks each operation and ensures that none of the operations are assigned to a time index such that the end of the operation is outside of the time interval $\mathbb{T}$.

7. Constraint used for setting the start time of each order.

$$s_o \leq M + \sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}} x_{mut}O^{uo}(t-M), \quad \forall u\in\mathbb{U},\ \forall o\in\mathbb{O} \tag{3.17}$$

$$s_o \leq M + \sum_{m\in\mathbb{M}}\sum_{t\in\mathbb{T}} S^{mlt}LO^{lo}(t-M), \quad \forall l\in\mathbb{L},\ \forall o\in\mathbb{O} \tag{3.18}$$

Searches through every operation of the order and extracts the start times.

The corresponding help variable in the array is set to the smallest start time of the order. The Big-M technique is used here, to make the constraints where the $x_{mut}O^{uo} = 0$ say that s_o should be smaller than a large number instead of forcing s_o to be 0. Note that the objective function wants the s_o variable to be as large as possible so only having upper bound constrained is viable.

8. Constraint used for setting the maximum amount of extra operators, at a given time index, required to execute the schedule.

$$m \geq \sum_{m \in \mathbb{M}} \Big(\sum_{u \in \mathbb{U}} \sum_{t' \in \mathbb{T}'} x_{mut'} UW^u + \sum_{l \in \mathbb{L}} \sum_{t'' \in \mathbb{T}''} S^{mlt''} LW^l \Big) - O^*, \quad \forall t \in \mathbb{T} \tag{3.19}$$

where $\mathbb{T}' = \{t - E^u, .., t\}$ and $\mathbb{T}'' = \{t - LE^l, .., t\}$.
This constraint loops through the machine-operation planes of every time interval and finds the highest amount of operators at any given time index. Note that the objective function wants the m variable to be as small as possible so only having a lower bound constrained is viable.

9. Constraint for setting the total amount of extra operators required to execute the schedule.

$$n_t \geq \sum_{m \in \mathbb{M}} \Big(\sum_{u \in \mathbb{U}} \sum_{t' \in \mathbb{T}'} x_{mut'} UW^u + \sum_{l \in \mathbb{L}} \sum_{t'' \in \mathbb{T}''} S^{mlt''} LW^l \Big) - O^*, \quad \forall t \in \mathbb{T} \tag{3.20}$$

where $\mathbb{T}' = \{t - E^u, .., t\}$ and $\mathbb{T}'' = \{t - LE^l, .., t\}$.
This constraint loops through the machine-operation planes of every time interval and adds up the number of extra operators at all the time indices. Note that the objective function wants the n_t variable to be as small as possible so only having lower bound constrained is viable.

10. Constraint used for setting the earliness of each order, expressed in a number of time indices.

$$e_o \geq D^o - \sum_{m \in \mathbb{M}} \sum_{t \in \mathbb{T}} x_{m,L^o,t}(t - E^{L^o}), \quad \forall o \in \mathbb{O} \tag{3.21}$$

where $\mathbb{T}' = \{t - E^u, .., t\}$ and $\mathbb{T}'' = \{t - LE^l, .., t\}$.
Loops through each order and computes the difference between the last operations' finished time and the due date of the order. Note that the objective function wants the e_o variable to be as small as possible so only having lower bound constrained is viable. e_o is also defined to have a lowest value of 0 which is used to construct a nonlinear relationship with tardiness.

3.3.1 Balance Part of Scheduling Behaviors

In this project the scoring of a schedule is solely dependent on the behaviors. The behaviors will have the ability to be weighted depending on the area of application. Since some of these behaviors increases alongside the number of time indices, the

priority between behaviors is subject to change when a time division is performed. This undesired inconsistency is highlighted in a simplified case, Figure 3.6.

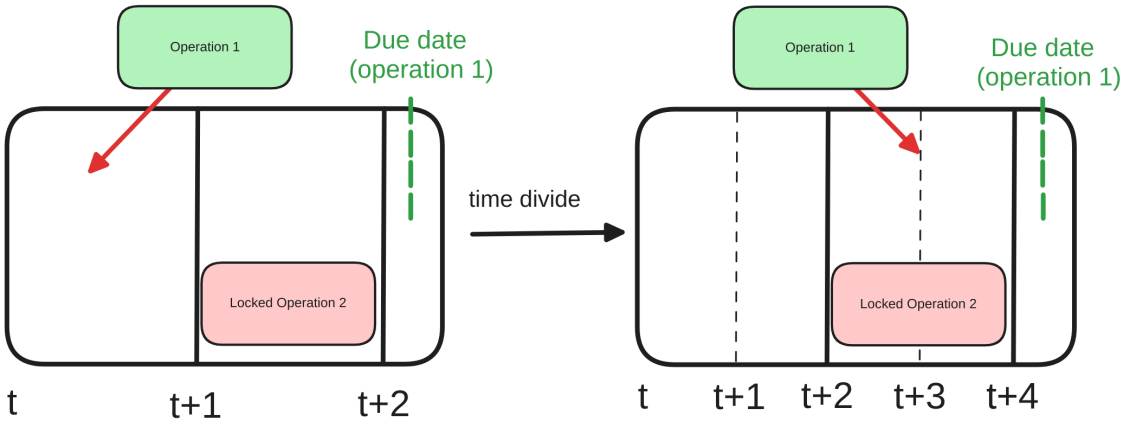


Figure 3.6: The case follows: assume that the weights are; earliness=1, operators=1.5 with optimal amount of operators=1 and 0 for the rest. Furthermore, assume that the operations require 1 operator each, and that the unlocked operation, "operation 1", has its due date placed as the figure illustrates. The unlocked operation is to be placed before and after a time division. Before the division there are two options; to get a punishment from operators which would be 1.5 or the punishment from earliness which would be 1. After the time division the same two options; to get a punishment from operators which would equal 1.5 or the punishment from earliness which now would be 2. This shows an inconsistency in the optimization goal.

To combat this inconsistency, and also to vaguely standardize the objective value of each behavior, balance parameters was introduced:

- Makespan, $MS^b = \frac{\gamma}{|\mathbb{U}|}$
- Lead time, $LT^b = \frac{\gamma}{|\mathbb{O}|}$
- Maximum amount of operators, $OX^b = \frac{1}{O^*}$
- Amount of operators per time index, $OM^b = \frac{\gamma}{|\mathbb{T}|}$
- Earliness, $E^b = \frac{\gamma}{|\mathbb{O}|}$
- Tardiness, $T^b = \frac{\gamma}{|\mathbb{O}|}$

Where γ is the time step size, O^* the desired amount of operators, and the sets, $\mathbb{O}, \mathbb{U}, \mathbb{T}$, regards the complete schedule, not just one ILP-instance.

3.4 Benchmark

To validate the performance of the EJSSP model, two different benchmarks have been developed and used. The first benchmark is done by comparing the EJSSP to a *Greedy Scheduler*. This *Greedy Scheduler*, GS, is reliant on heuristics and has two different scheduling algorithms. The algorithms will be presented in subsection 3.4.1. Due to the heuristics used by the greedy scheduler, it is either only relevant to benchmark against the scheduling behaviors makespan and lead time or against operators and makespan. The other benchmark compares the resulting objective value to a lower bound estimation, denoted T^*, of the theoretical best. The estimation will be presented in subsection 3.4.2.

3.4.1 Greedy Scheduler

As mentioned previously, the greedy scheduler has two algorithms, which can not be used in combination. The first algorithm focuses on performing well regarding makespan and lead time in combination. The other algorithm centers primarily around operators and on makespan. Both of the algorithms iteratively places one operation at a time and use a selection mask, or filter, that defines which operations are allowed to be selected. The selection mask is an element-wise multiplication between two other masks - a precedence mask, ensuring that the precedence constraint is not broken, and a duplicate mask which only allows an operation to be selected once. In the first algorithm, the selection is done by placing the operations of one order at a time. In the second algorithm, a set of viable operations are selected such that the sum of operators is as close as possible to desired amount of operators. This is done for every time index, where previous operations that span into the current time index is considered. After an operation have been selected, the algorithms assigns the operation to a machine and time index. The machine assignment is determined by localizing the machine that is unoccupied as close as possible to the precedence time. Precedence time is the earliest time instance the operation can be placed without breaking the precedence constraint. Regarding the time index assignment, the two algorithms differs slightly. In the first algorithm, the time index is either when the machine is unoccupied or the precedence time of the operation, whichever is the largest. Whilst the second algorithm, also takes the current time index into consideration, assigning the operation to the biggest of these three time indices. The process is repeated until every operation has been assigned. In Figure 3.7 the schedule following the first algorithm is shown, using the data described in the data section, 3.1. The Pseudo code of the algorithm is presented in algorithm 2. In Figure 3.8 the schedule following the second algorithm is shown, using the data described in the data section, 3.1. The Pseudo code of the algorithm is presented in algorithm 3.

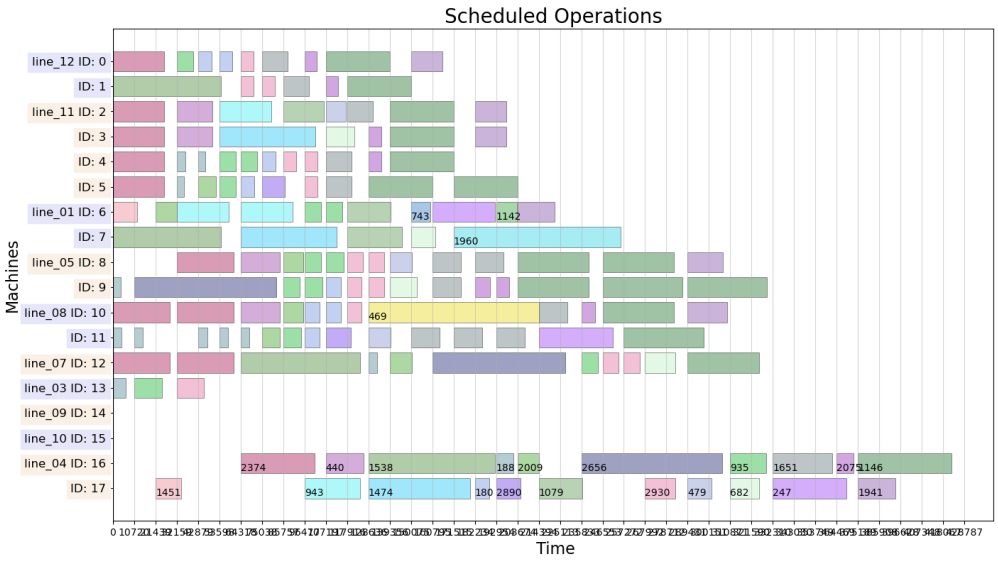


Figure 3.7: Schedule produced by the first algorithm of the greedy scheduler. The rectangles correspond to each individual operation, and the color denotes the order to which it belongs. Each row represents a machine, whilst the grid and columns represent the time indices. The final operation of each order is characterized by containing a number.

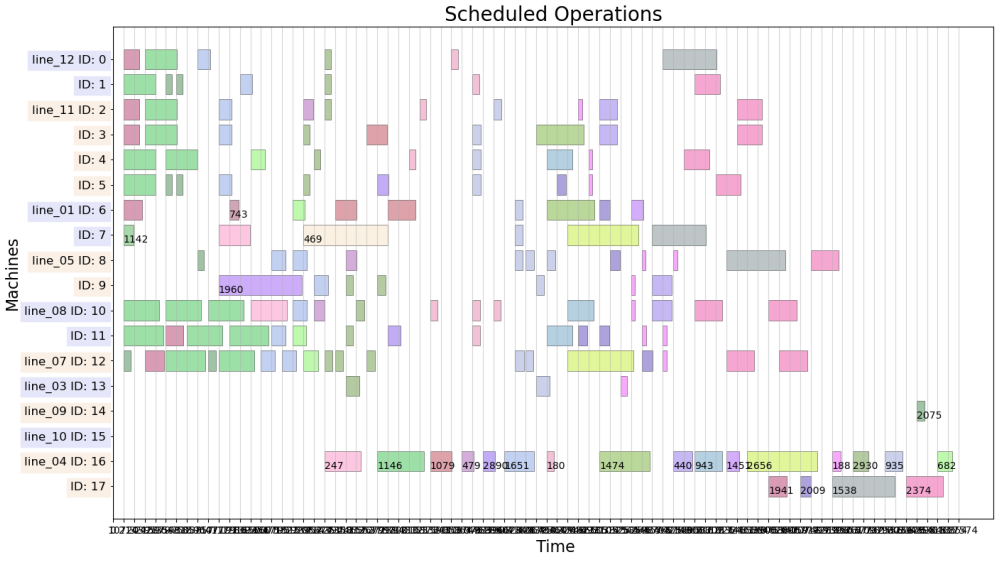


Figure 3.8: Schedule produced by the second algorithm of the greedy scheduler. The rectangles correspond to each individual operation, and the color denotes the order to which it belongs. Each row represents a machine, whilst the grid and columns represent the time indices. The final operation of each order is characterized by containing a number.

Algorithm 2 First algorithm of the greedy scheduler

$\mathcal{S}$:= array of operations
$\mathcal{N}$:= priority array w.r.t due dates
$\mathcal{P}$:= array denoting the number of operations needed to be executed prior
$\mathcal{D} = \mathbf{1}$
$\mathcal{F} = \mathcal{D} \circ \mathcal{P}$
$\mathbb{P} = \mathbf{0}$
$\mathbb{M} = \mathbf{0}$
while $|\mathcal{S}| > 0$ **do**
    $iS \leftarrow \arg\max(\mathcal{F} \circ \mathcal{N})$
    $iP \leftarrow \mathcal{S}_{iS}.parent.index$
    $s \leftarrow \mathcal{S}_{iS}$
    for $m \in \mathcal{M}_{iS}$ **do**
        if $|\mathbb{M}_m - \mathbb{P}_{iS}|$ is smallest **then**
            $M \leftarrow m$
            $tS \leftarrow \max(\mathbb{M}_m, \mathbb{P}_{iS})$
            $tF \leftarrow tS + \mathbb{E}_{iS}$
            $\mathbb{M}_m \leftarrow tF$
            $\mathbb{P}_{iS} \leftarrow tF$
        end if
    end for
    $\mathbb{S}_{iS} \leftarrow [M, tS]$
    $\mathcal{D}_{iS} \leftarrow \mathcal{D}_{iS} - 1$
    $\mathcal{P}_{iP} \leftarrow \mathcal{P}_{iP} - 1$
end while

Algorithm 3 Second algorithm of the greedy scheduler

$\mathcal{S}$:= array of operations
$\mathcal{N}$:= priority array w.r.t due dates
$\mathcal{P}$:= array denoting the number of operations needed to be executed prior
$\mathcal{D} = \mathbf{1}$
$\mathcal{F} = \mathcal{D} \circ \mathcal{P}$
$\mathbb{P} = \mathbf{0}$
$\mathbb{M} = \mathbf{0}$
$\mathbb{T}$:= sets time step size
$iT = 0$
while $|\mathcal{S}| > 0$ **do**
  $spillover \leftarrow spillovers[iT]$
  $[iS] \leftarrow best\_combo\_operations(spillover)$
  for $iS \in [iS]$ **do**
    $iP \leftarrow \mathcal{S}_{iS}.parent.index$
    $s \leftarrow \mathcal{S}_{iS}$
    for $m \in \mathcal{M}_{iS}$ **do**
      if $|\mathbb{M}_m - \mathbb{P}_{iS}|$ is smallest **then**
        $M \leftarrow m$
        $tS \leftarrow \max(\mathbb{M}_m,\ \mathbb{P}_{iS},\ iT \cdot \mathbb{T})$
        $tF \leftarrow tS + \mathbb{E}_{iS}$
        $\mathbb{M}_m \leftarrow tF$
        $\mathbb{P}_{iS} \leftarrow tF$
      end if
    end for
    for $so \in [iT \cdot \mathbb{T}, tF]$ **do**
      $spillovers[so] \leftarrow spillovers[so] + \mathcal{S}_{iS}.operators$
    end for
    $\mathbb{S}_{iS} \leftarrow [M, tS]$
    $\mathcal{D}_{iS} \leftarrow \mathcal{D}_{iS} - 1$
    $\mathcal{P}_{iP} \leftarrow \mathcal{P}_{iP} - 1$
  end for
end while

In the algorithms 2 and 3, $\mathbb{S}$ is a matrix of the schedule with size $2 \times |\mathcal{S}|$ where row one represents the selected machine of the operation, and the second row the start time. $\mathcal{D}$ is a matrix ensuring that no operation can be selected more than once. $\mathbb{P}$ denotes the current precedence time of each operation, whilst $\mathbb{M}$ denotes each machines, $\mathcal{M}$, current end time.

3.4.2 Theoretical Best

As declared previously, T^* is a lower bound of the theoretical best. T^* is evaluated by the five scheduling behaviors individually and defined as:

$$T^* = \sum_{x=1}^{5} w_x \cdot T_x^*$$

Where T_x^* is the estimated theoretical best of a behavior and computed as:

- Makespan, T_{MS}^* - Loops through all the operations and cumulatively adding up the execution times of the operations.
- Lead time, T_{LT}^* - The theoretically best lead time of an order is computed by localizing the longest path from the final operation to one of the leaf operations, operations without any precedence constraint. The traversal of the path is done recursively and the length of the path is defined by its operations' cumulative execution time. This computation is performed for each order and added together.
- Amount extra operators, T_O^* - Set to the number of preferred operators since this is achievable given that the distance between the first and last time index is large enough.
- Earliness, T_D^* - Set to 0 since an order can be completed at the same time index as its corresponding due date.
- Tardiness, T_D^* - Set to 0 since an order can be completed at the same time index as its corresponding due date.

Note that none of these computations considers external effects, such as the presence of other orders. Nor do these computations embed all constraints, an example being the machine constraint.

3.5 Metrics

This section explains the metrics used to present the performance of the model:

- $\Delta\%Z$ - General improvement of the schedule. This metric is percentage-based and compares the objective value of the initial schedule to the objective value of the solution. This is computed by

$$\Delta\%Z = 100(1 - \frac{F}{I})$$

  where F is the objective value of the final schedule and I the objective value of the initial schedule. A value of 100 indicates optimality whilst 0 implies no improvement.
- $\Delta\%T_x^*$ - Comparison between one scheduling behavior, x, and its corresponding term T_x^*. This is computed by

$$\Delta\%T_x^* = \frac{F_x}{T_x^*}$$

  Where F_x is the term of scheduling behavior x in F. A value of 1 indicates optimality and larger values indicate worse performance.
- ΔT_D^* - Denotes the total distance from the completion time of every order to its due date. Earliness and tardiness is not included in $\Delta\%T_x^*$ due to complications with division by zero. This is computed by

$$\sum_{o \in \mathbb{O}} max\Big(w_E(d_o - c_o),\ w_T(c_o - d_o)\Big)$$

  where c_o represents the completion time of an order, w_E is the earliness weight, w_T is the tardiness weight, and d_o represents the due date.

- $\Delta\%T^*$ - Performance of the schedule in relation to T^*. This is computed by

$$\Delta\%T^* = \frac{F}{T^*}$$

  A value of 1 indicates optimality and larger values indicate worse performance.
- $\Delta\%S^*$ - Standardised improvement of the schedule. Measures the progress of the solution from the initial value towards T^*. This standardised improvement is computed as

$$\Delta\%S^* = 100(1 - \frac{F - T^*}{I - T^*})$$

  A value of 100 indicates optimality and 0 indicates no improvement.
- $\Delta\%GS_1$ - Performance of the ILP model in relation to the first greedy algorithm. This is computed by

$$\Delta\%GS_1 = 100(1 - \frac{F}{G_1})$$

  Where G_1 is the objective value of the greedy scheduler. A positive value indicates that the ILP outperformed the greedy scheduler whilst a negative value indicates that the greedy scheduler outperformed the ILP.
- $\Delta\%GS_{1,x}$ - Comparison between one scheduling behavior, x, and its corresponding term $GS_{1,x}$. This is computed by

$$\Delta\%GS_{1,x} = \frac{F_x}{G_{1,x}}$$

  Where $G_{1,x}$ is the objective value of a scheduling behavior in the first algorithm of the greedy scheduler. A value above 1 indicates that the term of GS outperforms the ILP and a value below 1 indicates that the term of ILP outperforms GS. Furthermore, a value of 1 implies that the scheduling methods perform equally well, whilst a value of 0 represents optimality of the ILP.
- $\Delta\%GS_2$ - Performance of the ILP model in relation to the first greedy algorithm. This is computed by

$$\Delta\%GS_2 = 100(1 - \frac{F}{G_2})$$

  Where G_2 is the objective value of the greedy scheduler. A positive value indicates that the ILP outperformed the greedy scheduler whilst a negative value indicates that the greedy scheduler outperformed the ILP.
- $\Delta\%GS_{2,x}$ - Comparison between one scheduling behavior, x, and its corresponding term $GS_{2,x}$. This is computed by

$$\Delta\%GS_{2,x} = \frac{F_x}{G_{2,x}}$$

  Where $G_{2,x}$ is the objective value of a scheduling behavior in the first algorithm of the greedy scheduler. A value above 1 indicates that the term of GS outperforms the ILP and a value below 1 indicates that the term of ILP outperforms GS. Furthermore, a value of 1 implies that the scheduling methods perform equally well, whilst a value of 0 represents optimality of the ILP.

4

Results

In the following chapter, the results of the ILP model applied to EJSSP will be detailed and compared to the implemented benchmarks.

4.1 Comparison and Benchmarking of Results

In this section five Tables will be shown, indicating and quantifying the performance of the models, Tables 4.1, 4.2, 4.3, 4.4, and 4.5. To avoid repetitiveness, the models in Table 4.2 through 4.5 are indexed as follows:

- Model 1 - Tardiness weight: 100, lead time weight: 10, earliness weight: 10 and makespan weight: 1
- Model 2 - Lead time weight: 100 and makespan weight: 10
- Model 3 - Operators weight: 100 and makespan weight: 10
- Model 4 - Tardiness weight: 100, operators weight: 10, earliness weight: 1
- Model 5 - Makespan weight: 100
- Model 6 - Lead time weight: 100
- Model 7 - Operators weight: 100
- Model 8 - Tardiness weight: 100 and Earliness weight: 100

The first column of each Table states which model is being compared, denoted by the index of the model. If a slot in any of the Tables only contains the symbol "-", then the model is not compared against that benchmark due to lack of relevancy. To show the stochastic element of the scheduler, two models was optimized 5 times each. The results are presented in Table 4.1, where the last column of the Table shows the distance from the mean, ΔD, of each run. These runs was conducted on the two models:

- Model 10 - Tardiness weight: 100, earliness weight: 10, lead time weight: 1
- Model 11 - Makespan weight: 100, Lead time weight: 10

Model	Run	Final Obj. Value	ΔD
10	1	1.230	0.389
10	2	2.149	-0.530
10	3	0.913	0.706
10	4	2.500	-0.881
10	5	1.302	0.317
Model	Run	Final Obj. Value	ΔD
11	1	240.564	-11.262
11	2	217.786	11.516
11	3	230.953	-1.651
11	4	236.794	-7.492
11	5	220.413	8.889

Table 4.1: Displaying results from 2 models with 5 runs each, where each runs deviation from the mean is shown in the last column. Green illustrates improvement, and red deterioration.

Model	Start Obj. Value	Final Obj. Value	$\Delta\%Z$
1	474.5	16.7	96.5 %
2	659.6	114.0	82.7 %
3	402.3	224.6	44.2 %
4	545.5	76.7	85.9 %
5	1214.9	213.3	82.4 %
6	538.1	99.5	81.5 %
7	288.8	200.0	30.7 %
8	535.1	8.6	98.4 %

Table 4.2: Displaying a percentage-based improvement for each ILP model, comparing the objective value of the initial schedule to the final schedule of each model. The last column presents the improvement percentage. The weights of the models are presented in 4.1.

Model	$\Delta\%T^*_{MS}$	$\Delta\%T^*_{LT}$	$\Delta\%T^*_O$	ΔT^*_D	$\Delta\%T^*$	$\Delta\%S^*$
1	15.394	1.168	-	1.9	2.515	97.9 %
2	5.628	1.396	-	-	1.710	92.0 %
3	4.969	-	1	-	1.096	90.0 %
4	-	-	1.235	11.684	3.075	90.1 %
5	4.311	-	-	-	4.311	85.9 %
6	-	1.645	-	-	1.645	91.8 %
7	-	-	1	-	1	100 %
8	-	-	-	8.6	-	98.6 %

Table 4.3: Displaying comparisons between the ILP model and the corresponding T^*. Column 2-5 compares each scheduling behavior against its corresponding T^*_x. The second to last column compares the model against T^*. The last column is the percentage-based progress, S^*. Weights of the models can be seen in 4.1.

Model	$\Delta\%GS_{1,MS}$	$\Delta\%GS_{1,LT}$	$\Delta\%GS_1$
2	1.093	0.503	42.1 %
3	0.965	-	3.5 %
5	0.838	-	16.2 %
6	-	0.58	42.0 %
11	0.798	1	18.9 %

Table 4.4: Displaying a percentage-based comparison between the models and the first algorithm of the greedy scheduler. An improvement across the board can be seen. Models that does not primarily focus on the scheduling behaviors makespan or lead time is not presented in the Table due to irrelevancy. Model weights are presented in 4.1.

Model	$\Delta\%GS_{2,O}$	$\Delta\%GS_{2,MS}$	$\Delta\%GS_2$
3	1	0.367	15.9 %
5	-	0.319	68.1 %
7	1	-	0 %

Table 4.5: Displaying a percentage-based comparison between the models and the second algorithm of the greedy scheduler. An improvement for model 3 can be seen, whilst both schedulers find optimum of model 7. Models that does not include the scheduling behavior operators, nor makespan, is not presented in the Table due to irrelevancy. Model weights are presented in 4.1.

4.2 Produced Schedules

In Figures 4.2 through 4.4, three results of different models are shown, each starting their optimization from the same deterministic initial schedule, Figure 4.1. The models are weighted as follows:

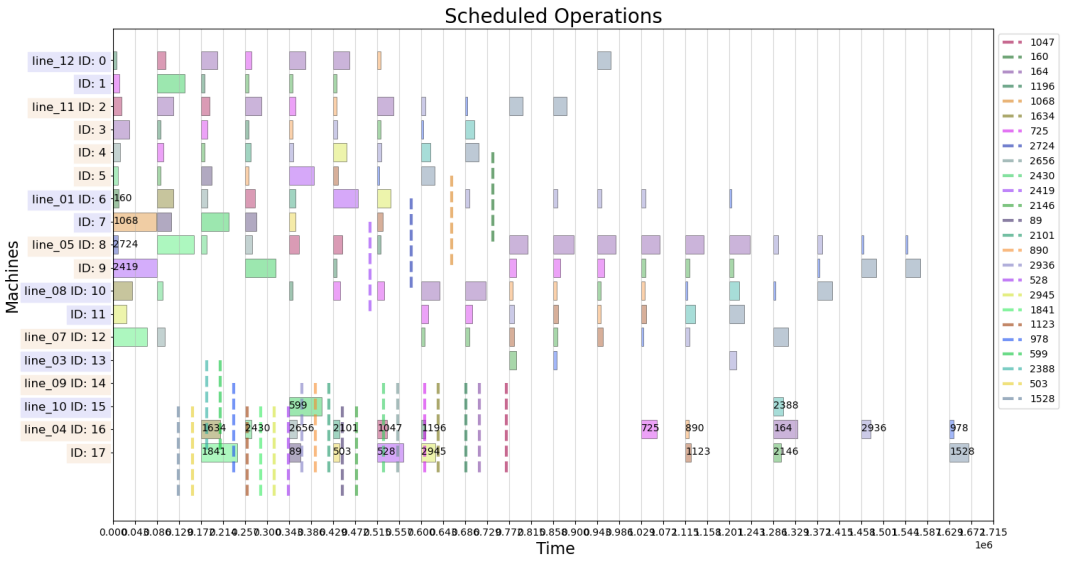


Figure 4.1: Initial schedule produced from the data described in section 3.1. This schedule is sparse, and time is divided once, allowing for a general placement of the operations during the early iterations of scheduling. This schedule is the starting point which all our schedules originate from, given the specified data set of this project.

- Figure 4.2 - Tardiness with a weight of 100, lead time a weight of 10, earliness 10, and makespan a weight of 1
- Figure 4.3 - Lead time with a weight of 100 followed by a makespan weight of 10
- Figure 4.4 - Tardiness weighted by 100, operators a weight of 10 and earliness with a factor of 1

The schedules structured as follows, the rectangles correspond to a individual operation, and all the operations of an order are color-coded. Each row represents a machine, whilst the grid and columns represent the time indices. The final operation of each order is characterized by containing a number. In the case of considering either earliness or tardiness, the due date of each order is present. These due dates are represented by vertically dotted lines and are placed on the same machine as the final operation of the corresponding order. Furthermore, the due dates are also color-coded and presented in a legend for the sake of clarity. Throughout every result, the data described in section 3.1 is used. The values of the time indices are obsolete since the length of a time interval is arbitrary and rather the relation or proportions between operations are the point of interest. Furthermore, since the data provided have been modified and anonymized no real-life connection can be drawn from the length between two time indices.

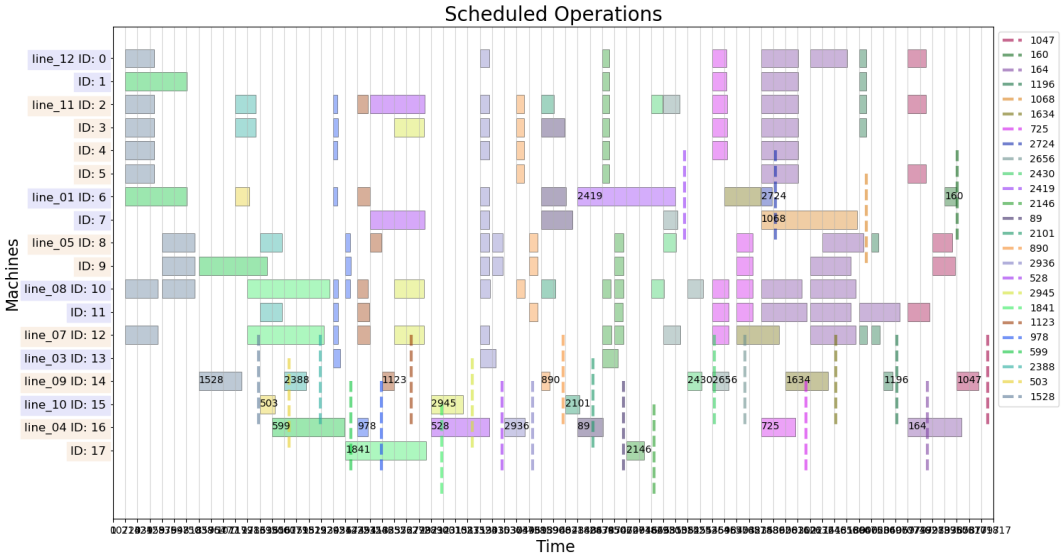


Figure 4.2: The produced schedule of a model with the scheduling behavior weights: tardiness - 100, lead time - 10, earliness - 10, and makespan - 1. This schedule was time divided a total of four times.

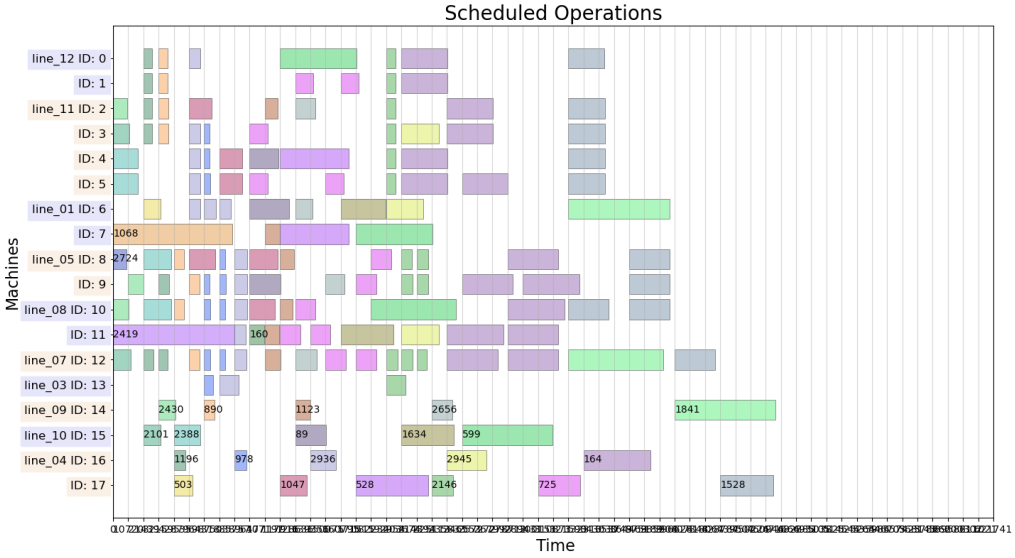


Figure 4.3: The produced schedule of a model with the scheduling behavior weights: lead time - 100 and makespan - 10. This schedule was time divided a total of four times.

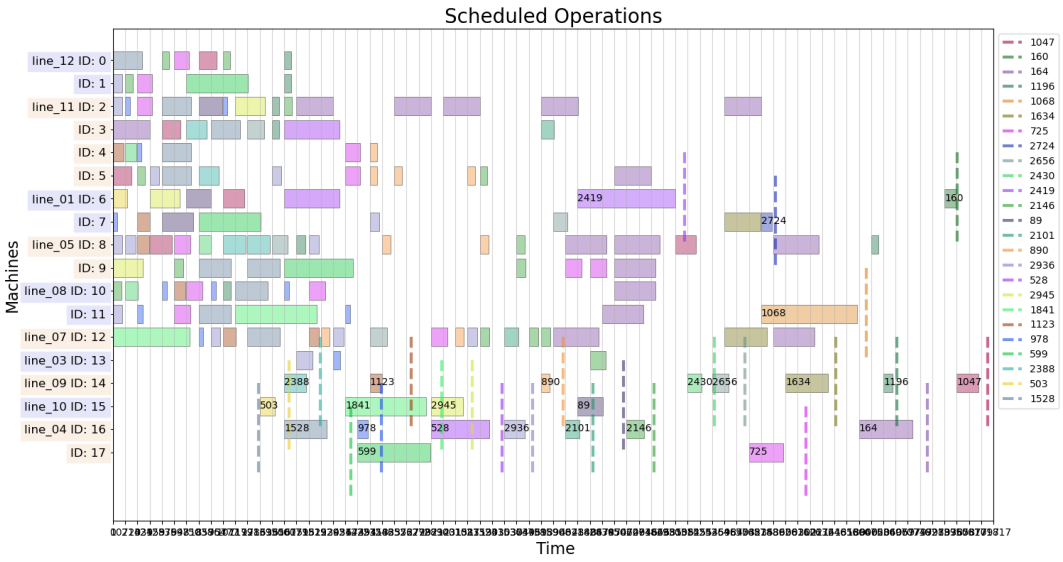


Figure 4.4: The produced schedule of a model with the scheduling behavior weights: tardiness - 100, operators - 10, and earliness - 1. This schedule was time divided a total of four times.

5
Discussion

The discussion will be partitioned into three main topics, performance of the model, complexity discussion, and future endeavors.

5.1 Performance

5.1.1 Metric Analysis

It is hard to measure how well the program performs with different behaviors. Since this is a NP-problem it is not possible to validate whether the solution is optimal nor how close to optimal it is, a common occurrence in optimization problems. Due to the absence of a known theoretical best value, other implementations of metrics had to be done, explained in section 3.5. But these implemented metrics has some inconsistencies and flaws. Starting with, $\Delta\%Z$, has a bias towards behaviors that performs poorly on the initial schedule. Regarding T^*, since it computes every behaviors by themselves, it does not consider that some behaviors counteracts each other, different behaviors forces the operations in different directions. Although T^* is good approximation for the behaviors in a vacuum, the expected deviancy between T^* and the actual theoretical best value increases alongside the number of behaviors. The standardised improvement measurement, $\Delta\%S^*$, is also bias towards behaviors that performs poorly on the initial schedule. The GS metrics are only relevant to compare against some models, however in these instances this metric is fair.

5.1.2 Result Analysis

From Table 4.2, we can notice how some models does not improve very much, throughout the optimization process, whilst other models achieves a distinct improvement. This indicates a combination of the performance of the behaviors on the initial schedule and the performance of the model. In Table 4.2, it can be noted that the models that prioritize operators, model 3 and 7, does not improve as drastically as the models 1, 2, 4, 5 and 8, which prioritizes either tardiness or makespan. This co-aligns with the initial schedule being sparse, resulting in good performance for operators and poor performance for makespan and tardiness on the initial schedule.
In Table 4.3, we see across the Table a great improvement. This is, as mentioned earlier, a bit misleading due to the dependence on the performance on the initial schedule. Thus, not much can be taken from the overall score. But we can compare them between each other. It is noticed from $\Delta\%S^*$ that makespan was the

worst performing behaviour. Where we know that $\Delta\%S^*$ should be bias towards makespans bad initial schedule. Makespan is also consistently the worst performing T_x^*, this could be because the T_{MS}^* is a rougher lower bound than the rest of T_x^*. But most likely this is just because makespan is the hardest problem for the program to get right, once the schedule has been compressed it's hard to move any operation.

In Table 4.4, one can see how each behavior of the model compares against the correspond $GS_{1,x}$. Overall it can be noted that our model performs better, with the exception of model 2 where the greedy scheduler achieves a better makespan. When the weights are included in the comparison, done by comparing the objective values, we see that the proposed solution-method outperforms the greedy schedule for all models. With models 2 and 6 performing more than 40% better than GS. One reason for these models obtaining such an improvement, is due to the GS primarily focuses on performing well makespan, whilst the model 2 and 6 prioritizes lead time. The most interesting and meaningful comparison is against model 11, where both schedulers aim to perform well on primarily makespan and then lead time. In this case we see that our model performs 15.9% better than the GS, which is a considerable amount. These analyses shows the flexibility of the our model, being able to adapt according to the weights set.

In the next Table, 4.5, comparisons against the second algorithm of GS is presented. In this Table it can be seen that our model performs better or equally to the GS on all accounts. Both of schedulers obtain optimality of operators, which can be expected due to it being easily achievable. Expanding the schedule and making it sparse enough, will result in optimality but most likely render the schedule useless in a real-world scenario. However model 3, which aims to distribute the operators equally yet keep the schedule as compact as possible, can be applied to a real-world manufacturing environment. This is also the aim of GS, thus making the comparison against model 3 the most meaningful and interesting one. From the Table we see that our model outperforms GS by 15.9%, implying that our model is indeed preferred.

In Table 4.1, which presents the deviation of the objective value between runs, it can be seen that the difference between the best and worst run of model 10 is 1.6 and 22.8 for model 11. This shows the stochastic nature of the solutions and importance of unlocking sequence, where the sequence of orders is more dependent when all orders share the same optimal placement. When makespan is prioritized, the order of the operations is mostly random because the first unlocked orders will pick the best location in the schedule without analysing the bigger picture of the schedule. This is not totally the case because in these results we choose to unlock two orders at a time, so the schedule have some chance of sorting the orders, before locking itself.

5.1.3 Alternative Metric

To mitigate the weakness in the approximation of T^*, where it does not take into account of behaviors working against each other, an error term can be introduced. This error, e, can be approximated by letting T^* follow the same percental loss as our program when behaviors are combined:

$$T_m^* = T^* + e \tag{5.1}$$

where:

$$e = T^* \frac{z(w_1 + ... + w_n)}{z(w_1) + ... + z(w_n)} \tag{5.2}$$

Where T_m^* is the new approximation to the theoretical best that take mixed behaviors into account, note that this is not longer a lower bound. $z(\mathbb{W})$ represent the objective value from a run with the weights $\mathbb{W} = w_1 + w_2 + ... + w_n$. We choose to mention this in the discussion because this approximation of T_m^* is dependent on what we try to validate, so it has to be taken into consideration if used.

5.1.4 Self-locking

Because one schedule can not be optimized at once the solver has some ways of locking itself. Depending on the type of data and choice of weights. During testing, we have noted some reasons that can lead to the solver not moving some operations where there obviously is a better placement for them, we have chosen to call this phenomenon *Self-locking*. The first self-locking occurs when one order is too big for the program to unlock it at once, when this happens the ability to move any operations in the order is highly compromised. If lead time is high this can even lead to situations where one order is completely stuck. This because the unlocked operations has the priority to stay with the rest of it's order. This sets a limitation on how big orders the program can handle which in our tests were around 6-12 operations in an order. Another self-locking occurs when an order should get past a compact area of operations. The compact area is multiple operations that are well positioned and tightly placed and do not want to be moved by themselves. If a compact area is bigger than the unlocked time period, operations are then not able to pass through, Figure 5.1.

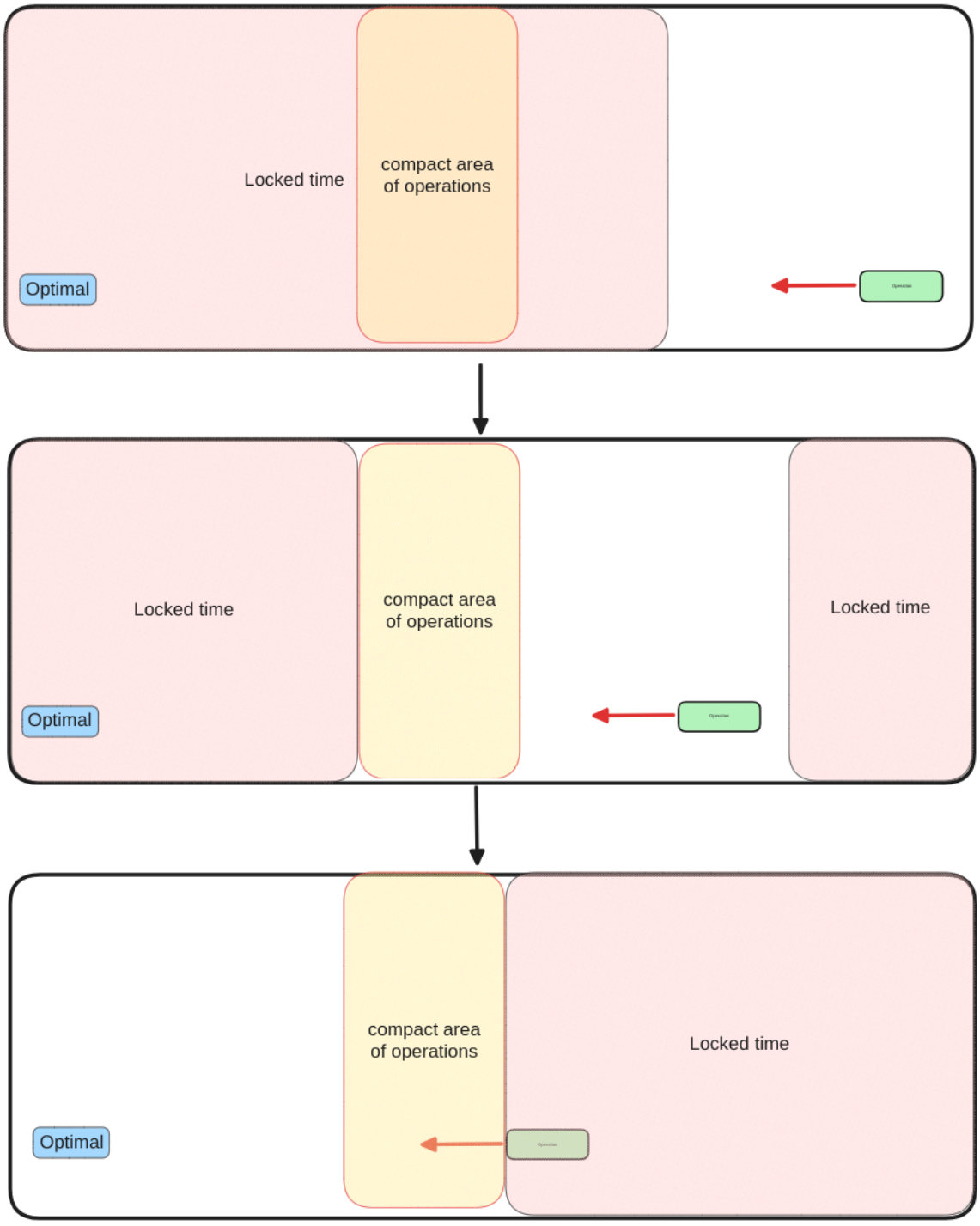


Figure 5.1: Shows how an operation can become locked behind a compact area of operations. Where the green box is the operation and the blue box is the operations optimal position. The red arrow symbolises the force acting on the operation.

To combat this problem the amount of orders that is unlocked was set to be greater than one.

5.1.5 Real-World Modeling

Constructing an objective function has its difficulties. These difficulties includes the absence of quantifying all schedules via one metric, due to schedules having different purposes, and the unpredictable nature of manufacturing environments. Furthermore, a real-world manufacturing environment usually involves nonlinear relationships and depicting these nonlinearities as a linear function is a difficult task. This can lead to a sub-optimal solution of the problem, from a real-world point of view. Additionally, if data from a real-world production chain was provided, an in-depth comparison could have been made. This comparison would benchmark our model against the scheduling method used by *The Lego Group*. From this benchmark more descriptive metrics would have been implemented, giving more substance to the results obtained. It would also give some meaning to the placement of the due dates since our generated locations of the due dates might make the earliness and tardiness schedules to space and therefore to easy. Furthermore, scheduling the real operations data, having a connection to actual time, would represent the problem in a greater manner, where our model rather acts as a proof of concept currently.

5.2 Complexity

Computational and time complexity is a focal point of solutions to NP-problems, in this case the EJSSP, and is frequently discussed. Optimizing time complexity can explore and enable new techniques of solving problems that by nature are difficult. Reducing the complexity is a critical factor for both the performance and applicability of a solution-method. Developing a such new and improved solution technique can either exploring more of the solution space in less time, or derive at the solution at a quicker rate. As the project progressed, we came to realize the importance of computational complexity and adverse effects accompanied by complex problems. Thus, complexity analysis and optimization regarding our proposed model and solution-method to the EJSSP, would have improved the end result. However if this was to be done, the scope of the thesis would be too broad. Furthermore, for a real-world big manufacturer the ability to produce a reliable schedule at a fast rate is a necessity. Enabling to act of combating unforeseeable event, such as machine failure, breakdowns or being short on staff.

As an insight to the complexity of our proposed model and solution-method, the average solution took 4.5 hours to complete. This is substantially longer than state-of-the-art solution-methods given the same amount of machines and operations, see [2] that solved a problem of 15 orders over 20 machines within an hour in one solution instance. A reason for the long solution-times of our model, is the fact that only a minority of the ILP-solves result in a significant improvement, with some solutions not improving the state of the schedule what so ever.

One improvement to our model, would be reducing the overhead time associated with constructing each ILP-instance. We observed that it roughly took 2 seconds for the solver, GLPK, to build the ILP model, whilst the actual time it took to

solve the instance was as little as 0.1 seconds. If the overhead time could be reduced significantly, a lot more iteration of solving ILP-instances could be performed during the same time span. A way of reducing the overhead time is to analyse our model even further and employ constraint skips, when a constraint is obsolete, or even mathematically reformulate the problem (similar to that of the disjunctive model or rank-based model mentioned in earlier). A novel attempt at formulating a more compact, thus less complex and intuitive, formulation was done. The basic idea was to formulate a model of the EJSSP represented by two arrays, one for an operations selected machine and one for the start time. The indices of these arrays, corresponds to a certain operation. This formulation can be performed, where multiple artificial variables needs to be introduced. However, this reformulation was scrapped due to time limit of the project.

Finally, the results could have been better if another solver like Gurobi was used or if the model was implemented in a programming language such as C or C++, which has the benefit of high computational power.

5.3 Future Endeavors

To take this project further it would be good to use real production data to have something to validate against. If we instead see what worked poorly in this project and try to make it better it would be the complexity. Both to lower the amount of variables which is possible if one reformulates the ILP, but also to remove some unnecessary time the solver uses. Where implementation of heuristics or metaheuristics could have enhanced the performance of our solution-method. Heuristics and metaheuristics can be incorporating for two main purposes, selecting what to unlock each iteration or creating the initial schedule. By utilizing heuristics to select what to unlock, each iteration of the optimization process could have a more impactfull improvement on the schedule. This would lead to a faster convergence towards the solution of the problem. In the case of creating an initial schedule, metaheuristics would have been preferred. This due to metaheuristics being more nuanced, which is reflected in their performance, and since the end result of the ILP depends on the initial schedule. The initial schedule from a metaheuristic would be followed by doing the fine-tuning of the schedule via ILP. A metaheuristic that suits JSSP, thus most likely EJSSP as well, is GA and is used by Guzman et al. in [4]. Relaying more on metaheuristics aligns with the goal of creating a flexible scheduling solution capable of handling the difficulties of real-world manufacturing environments.

6
Conclusion

Due to the lack of a theoretical best and not being able to validate against real-world data, it is challenging to provide a definitive verdict on the effectiveness of our scheduler. From our observations, the scheduler performs well on simpler scheduling problems where makespan is not the primary concern. However, optimizing for makespan proves to be the most challenging aspect, as it is difficult to achieve high levels of optimization in this regard. Our program generates schedules that are comparable to those produced by simpler greedy algorithms. While greedy schedulers are less versatile, they are significantly faster to run. The versatility of our ILP-based approach is its primary advantage, allowing it to handle a wider range of scheduling scenarios more flexibly. One of the main issues we encountered throughout our project was the limited number of operations that our ILP model could unlock simultaneously. This limitation may be inherent to the ILP approach, but it could also be a consequence of our model's complexity, which involves a substantial number of variables. Future research should focus on addressing these limitations by exploring alternative optimization techniques, such as integrating metaheuristics or matheuristics, to improve initial schedules and enhance the overall efficiency of the model. Additionally, obtaining real-world data for validation could provide more concrete insights into its performance and practical applicability.

In conclusion, while our ILP-based scheduler shows promise and versatility, its performance is currently constrained by challenges related to makespan optimization and computational complexity. By addressing these issues through continued research and development, we can move closer to achieving a robust and efficient scheduling solution capable of meeting the demands of modern manufacturing environments.

Bibliography

[1] R. Z. Ríos-Mercado and J. F. Bard, "Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups," *Computers amp; Operations Research*, vol. 25, no. 5, p. 351–366, May 1998. [Online]. Available: http://dx.doi.org/10.1016/S0305-0548(97)00079-8

[2] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers amp; Operations Research*, vol. 73, p. 165–173, Sep. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2016.04.006

[3] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models," *Computers amp; Operations Research*, vol. 142, p. 105731, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2022.105731

[4] E. Guzman, B. Andres, and R. Poler, "Matheuristic algorithm for job-shop scheduling problem using a disjunctive mathematical model," *Computers*, vol. 11, no. 1, p. 1, Dec. 2021. [Online]. Available: http://dx.doi.org/10.3390/computers11010001

[5] M. Gomes *, A. Barbosa-Póvoa, and A. Novais, "Optimal scheduling for flexible job shop operation," *International Journal of Production Research*, vol. 43, no. 11, p. 2323–2353, Jun. 2005. [Online]. Available: http://dx.doi.org/10.1080/00207540412331330101

[6] B. Yan, M. A. Bragin, and P. B. Luh, "Novel formulation and resolution of job-shop scheduling problems," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, p. 3387–3393, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1109/LRA.2018.2850056

[7] M. ApS. Disjunctive constraints. [Online]. Available: https://docs.mosek.com/latest/cxxfusion/tutorial-djc-fusion.html

[8] A. V. Team. What is linear programming? definition, methods and problems for data scientists. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/02/lintroductory-guide-on-linear-programming-explained-in-simple-english/

[9] V. S. Borkar and K. S. M. Rao, *Convex Sets*. Springer Nature Singapore, 2023, p. 39–60. [Online]. Available: http://dx.doi.org/10.1007/978-981-99-1652-8_3

[10] G. Gordon. Optimization, convex sets. [Online]. Available: https://www.cs.cmu.edu/~ggordon/10725-F12/scribes/10725_Lecture3.pdf

[11] D. Simon. Intersecting halfspaces. [Online]. Available: https://www.researchgate.net/figure/The-figure-shows-a-polytope-the-shaded-area-which-is-the-intersection-of-five_fig4_269398882

[12] C. Lewis. Linear programming: Theory and applications. [Online]. Available: https://www.whitman.edu/Documents/Academics/Mathematics/lewis.pdf

[13] L. Oesper. Integer linear programming: What? why? how? [Online]. Available: https://www.cs.carleton.edu/cs_comps/2324/integerLinearPrograming/index.php

[14] D. Gribanov, D. Malyshev, and P. M. Pardalos, "Delta-modular ilp problems of bounded co-dimension, discrepancy, and convolution," 2024. [Online]. Available: https://arxiv.org/abs/2405.17001

[15] N. C. A. S. College. The big-m method. [Online]. Available: https://www.nascollege.org/e%20cotent%2010-4-20/DR%20K%20K%20KANSAL/L%209%20Minimization%20The%20Big%20M%20Method%20M%20COM%2019-4.pdf

[16] Gurobi. Gurobi. [Online]. Available: https://www.gurobi.com/

[17] CPLEX. Cplex. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

[18] GLPK. Glpk. [Online]. Available: https://www.gnu.org/software/glpk/

[19] M. P. Stephen P. Boyd, "Branch and bound methods." [Online]. Available: https://web.stanford.edu/class/ee364b/lectures/bb_slides.pdf

[20] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*. Springer International Publishing, 2020. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-39415-8

[21] E. Kostina, "The long step rule in the bounded-variable dual simplex method: Numerical experiments," *Mathematical Methods of Operations Research*, vol. 55, no. 3, p. 413–429, Jun. 2002. [Online]. Available: http://dx.doi.org/10.1007/s001860200188

[22] V. Chvátal. The dual simplex method. [Online]. Available: http://cgm.cs.mcgill.ca/~avis/courses/567/notes/ch10.pdf

[23] T. E. of Encyclopaedia. Np-complete problem. [Online]. Available: https://www.britannica.com/science/NP-complete-problem

[24] N. Meghanathan. Np-complete problems and heuristics. [Online]. Available: https://www.jsums.edu/nmeghanathan/files/2020/01/CSC323-Sp2020-Module-6-P-NP-NP-CompleteProblems-ApproxAlgorithms.pdf

[25] Wikipedia. Heuristic. [Online]. Available: https://en.wikipedia.org/wiki/Heuristic

[26] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, *Metaheuristic Algorithms: A Comprehensive Review*. Elsevier, 2018, p. 185–231. [Online]. Available: http://dx.doi.org/10.1016/B978-0-12-813314-9.00010-4

[27] S. M. Almufti, A. Ahmad Shaban, Z. Arif Ali, R. Ismael Ali, and J. A. Dela Fuente, "Overview of metaheuristic algorithms," *Polaris Global Journal of Scholarly Research and Trends*, vol. 2, no. 2, p. 10–32, Apr. 2023. [Online]. Available: http://dx.doi.org/10.58429/pgjsrt.v2n2a144

[28] S. Ikli, C. Mancel, M. Mongeau, X. Olive, and E. Rachelson, "The aircraft runway scheduling problem: A survey," *Computers amp; Operations Research*, vol. 132, p. 105336, Aug. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2021.105336

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY