CHALMERS

Optimal and Suboptimal Scheduling of the Dual Resource Job Shop Problem with Multi-Skilled Workforce

Jacob Andreasson & Robin Persson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg 2026

www.chalmers.se

Master thesis 2026

Optimal and Suboptimal Scheduling of the Dual Resource Job Shop Problem with Multi-skilled Workforce

Jacob Andreasson & Robin Persson

CHALMERS

Department of Electrical Engineering
Chalmers University of Technology
Göteborg 2026

Optimal and Suboptimal Scheduling of the Dual Resource Job Shop Problem with Multi-Skilled Workforce

Jacob Andreasson & Robin Persson

Supervisor: Sabino Francesco Roselli, Department of Electrical Engineering (E2)
Supervisor: Jonas Olsson, Thorlabs Sweden AB
Examiner: Martin Fabian, Department of Electrical Engineering (E2)

Master thesis 2026
Department of Electrical Engineering (E2)
Chalmers University of Technology
SE-412 96 Göteborg
Phone +46 31 772 1000

Written in LaTeX
Gothenburg 2026

Scheduling optimization for a multi-skilled workforce
Jacob Andreasson Robin Persson
Department of Electrical Engineering (E2)
Chalmers University of Technology

Abstract

Efficient production scheduling is important for reducing costs, meeting delivery deadlines, and balancing workloads in manufacturing environments. This thesis investigates scheduling optimization for a multi-skilled workforce derived from the production environment at Thorlabs Sweden AB. The problem is modelled as a dual-resource flexible job shop problem (DR-FJSP), where both machines and workers must be available simultaneously in order to execute some operations, and where workers hold varying qualifications across product types. Five solution approaches are developed and evaluated: a mixed integer linear programming (MILP) formulation implemented in Gurobi, a satisfiability modulo theories (SMT) formulation implemented in Z3, a constraint programming (CP) model using Google OR-Tools CP-SAT, a genetic algorithm (GA), and a greedy priority-rule heuristic. The models incorporate sequence-dependent setup and cleaning times, operation-type constraints, and worker qualification requirements. The approaches are benchmarked on 100 synthetic instances and evaluated on real production instances in terms of solution quality and computational efficiency. The benchmark results show that OR-Tools solved the greatest number (57/100) of instances to optimality with a time limit of one hour. The CP-SAT implementation scaled better than both Gurobi and Z3 on the evaluated benchmark set, though all three exact methods are inferior to the GA in terms of runtime scaling on larger instances. The GA, seeded with heuristic solutions consistently improves upon the heuristic baseline across key performance indicators including makespan, total completion time, tardiness, and lateness, while remaining tractable for production-scale instances.

Keywords: SMT, MILP, JSP, CP, GA, Heuristics, Scheduling, Optimization

Acronyms

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

BFS	Basic feasible solution
CDCL	Conflict driven clause learning
CNF	Conjunctive normal form
CP	Constraint programming
CSP	Constraint satisfaction problem
DR-FJSP	Dual resource flexible job shop problem
EDD	Earliest due date
ERD	Earliest release date
FCFS	First come first served
FJSP	flexible job shop problem
GA	Genetic algorithm
HFS	Hybrid flow shop
JSP	Job shop problem
LP	Linear programming
LPT	Longest processing time
MAC	Machine assignment chromosome
MILP	Mixed integer linear programming
OSC	operation assignment chromosome
RL	Reinforcement learning
RPD	Relative percentage difference
SAT	Satisfiability
SMT	Satisfiability modulo theories
SPT	Shortest processing time
WAS	Worker assignment chromosome

Contents

1

Introduction

The ability to efficiently schedule production orders is critical for the overall performance in a manufacturing environment. Effective scheduling directly impacts a company's cost efficiency by reducing idle times, balances resource use, and minimizes total production time. In practice, many companies rely on heuristic rules that help decision-makers determine which job to process next. Some examples of heuristic rules are earliest due date (EDD) or first-come-first-served (FCFS). While these approaches are fast and easy to apply, they are sub-optimal since they only consider one decision at a time and ignore dependencies between resources, and make no attempt at optimizing any global performance metrics. The resulting schedules are suboptimal and as manufacturing systems grow in complexity, the gap between heuristic and optimized schedules widens. This makes the case for other scheduling methods more compelling. In practice, the cost of suboptimal scheduling compounds across resources; assigning the most urgent order to the fastest available worker may block a future operation that only one other person is qualified to perform. No priority rules can reason about this globally.

Production scheduling has been widely studied within operations research, giving attention to a class of combinatorial optimization problems. One of the most well-known of these is the job shop problem (JSP) [1], in which $J = \{j_1, \ldots, j_n\}$ jobs must be processed on $M = \{m_1, \ldots, m_k\}$ machines. Each job consists of a sequence of operations $O_j = \{o_{j1}, \ldots, o_{ji}\}$, where each operation requires a specific machine for a given processing time. No machine can perform two operations simultaneously, and the operations within the job must be performed in order. The objective is typically to assign each operation to a machine such that the total time to complete all jobs is minimized

While the regular JSP is a good starting point, it does not usually reflect real production systems well; therefore, several extensions of the JSP have been proposed to better capture real manufacturing environments. The flexible job shop problem (FJSP) [1] relaxes the JSP by allowing each operation to be processed on any machine from a set of eligible machines. This variant better represents systems where multiple machines can perform the same task. A further extension is the dual-resource flexible job shop (DR-FJSP) [2], which introduces an additional constraint: a machine and a worker must be simultaneously available for an operation to be-

gin. This aims to capture manufacturing environments where human operators are necessary to proceed such as in highly manual environments.

Both the JSP and DR-FJSP are known NP-hard problems [3], meaning that no general polynomial-time algorithm is known. The number of possible operation sequences grows combinatorially. For a JSP with n jobs and m machines, the number of ways to sequence operations across machines has an upper bound of $(n!)^m$, a figure that renders exhaustive searches computationally infeasible even for moderate instances sizes. For large instances, exact methods quickly become intractable and finding optimal solutions within practical time is a challenge.

There are several methods that have been used to address the job shop problem. Mixed integer linear programming (MILP) [4] formulates the scheduling problem as a mathematical program, using the simplex and branch-and-bound algorithms to minimize an objective function subject to a set of linear constraints. It is a well-established method and has been used in studies such as [5], where a textile manufacturing plant was modelled as a DR-FJSP. However, MILP fails to solve many industrial-sized problems within practical time limits. Another alternative is to use satisfiability modulo theories (SMT) [6], which extend boolean satisfiability (SAT) to more complex formulas including integers and real-numbers. This makes it a very flexible modeling language that has been applied to the JSP and, while capable of handling larger instances than MILP, still suffers from scalability issues, though SMT solvers have nonetheless been shown to be a competitive alternative to MILP solvers [7]. Beyond MILP and SMT-based methods, other declarative optimization frameworks have also been used. Constraint programming (CP) [8] provides another approach to solving different shop problems. A CP problem is modeled as a set of decision variables, where each variable has a domain and is subject to a set of constraints. CP uses a combination of constraint propagation and search to find a feasible solution. In [9] CP was applied to flexible flow shop; a problem variant where in which jobs pass through a fixed sequence of production stages, each containing multiple parallel machines, and must be processed on exactly one machine per stage. [9] shows that CP is able to outperform metaheuristic methods in flexible flow shop scheduling, particularly for small-to-medium sized instances. However, as problem size increase, exact methods become computationally intractable. Where this is the case metaheuristic algorithms offer a practical alternative. Genetic algorithms (GAs) [10] are stochastic, population based algorithms that evolve candidate solutions through mutation, crossover and selection. They are suboptimal algorithms but still capable of generating high quality solutions within practical time frames and have been shown to be useful for DR-FJSP; [11] applied an improved genetic algorithm to the DR-FJSP and found significant improvements in scheduling performance compared to three different evolutionary/metaheuristic algorithms. Also, learning-based approaches, particularly reinforcement learning (RL) has recently gained traction, with methods such as Q-learning being applied to various JSP variants [12]. This approach models the problem as a Markov decision process which lets the agent learn effective strategies to minimize makespan and adapt to dynamic conditions like machine breakdowns [13].

Each of these methods involves a fundamental trade-off between solution quality and computational tractability. Exact methods such as MILP are capable of finding optimal solutions but their applicability is limited to smaller problem instances due to the NP-hard nature of the JSP. Metaheuristic approaches like GAs scale better but offer no optimality guarantees. CP often outperforms metaheuristic methods on small-to-medium sized problems while also being more scalable than MILP for certain problems. The choice of method is therefore not an obvious one, it depends on the structure and size of the problem, the constraints involved, and the acceptable trade-off between schedule quality and computation time. This motivates evaluating multiple approaches on a well-defined problem.

This project is performed in collaboration with Thorlabs Sweden AB, a subsidiary of Thorlabs Inc., a global photonics company that manufactures photonic instruments and systems. The production environment at Thorlabs presents a DR-FJSP: A moderately sized scheduling problem with dual resources and a multi-skilled workforce, meaning that not every operator is qualified to perform every job. The production environment at Thorlabs is mainly manual and organized into work and test benches, where operators assemble and test products. Operations fall into three distinct types based on their resource requirements: machine-only operations, worker-only operations, and dual-resource operations, which require both a machine and a worker simultaneously. The production floor is organized into production pools – physical areas dedicated to the assembly of a specific category of products, where each pool groups together all jobs belonging to that product family. Additionally, workers receive a sequence-dependent setup and cleaning times when switching between jobs or moving between production pools, making the assignment of workers sensitive to the order in which jobs are scheduled. Consequently, the production system can be described by a DR-FJSP with some additional constraints unique to the Thorlabs environment.

Despite the width of available approaches, comparative studies that evaluate MILP, SMT, CP and GA under identical constraints on the same DR-FJSP problem class are, to the best of the authors' knowledge, limited. In [7] SMT and MILP are compared for the JSP, but CP or meta heuristics are not included, and dual-resource constraints are not considered. Industrial validation also seems to be uncommon; where [5] is a notable exception in applying a DR-FJSP model to real manufacturing environment, but the constraint structure differs from the one studied here. In particular, that formulation does not model sequence-dependent pool-crossing and cleaning times, nor the three distinct operation types considered here. This thesis addresses both gaps by implementing five methods – a MILP approach, an SMT approach, a GA, a CP formulation, and a heuristic approach – and evaluating the best performing ones on real production data.

1.1 Aim

This thesis aims to provide practical insights into which methods are most suitable for this class of scheduling problems. The production system at Thorlabs is

modelled as a DR-FJSP with environment specific constraints. Five solution approaches are considered: an SMT formulation (Z3), a MILP formulation (Gurobi), a CP formulation (Google OR-Tools CP-SAT), a priority-rule heuristic, and a genetic algorithm. These algorithms will be evaluated against each other on a set of benchmark instances in terms of solution quality and computational efficiency. Furthermore, the best performing algorithms will be compared using Thorlabs' production data, where they will be compared in terms of computational efficiency and tardiness.

1.2 Research questions

In this thesis, the following research questions will be addressed:

1. How do MILP, SMT, CP, GA, and priority-rule heuristics compare in terms of solution quality, measured as makespan on synthetic benchmark instances, and computational efficiency, measured as solve time and number of instances solved to optimality within a one hour time limit?

2. Which method best balances solution quality and tractability when applied to Thorlabs' production data?

3. Which method produces the best incumbent solution on Thorlabs production instances within a fixed one hour time limit, evaluated in terms of total tardiness?

4. To what extent do optimization-based methods reduce tardiness and makespan compared to a priority-rule heuristic baseline on the DR-FJSP, and does this improvement justify the additional computational cost?

The remainder of this report is structured as follows. Chapter 2 reviews relevant theory and background needed to understand the thesis. Chapter 3 presents the problem formulation, the solution approaches as well as the experiment methodology. Chapter 4 presents the experimental results. Chapter 5 presents the discussion of said results, and Chapter 6 outlines conclusions and future work.

2

Technical background

This chapter provides the theoretical foundation necessary to understand the methods developed and evaluated in this thesis. It covers computational complexity of scheduling problems, which motivates the use of approximate methods alongside exact solvers, as well as the relevant scheduling theory, optimization techniques, and algorithmic frameworks upon which the solution approaches are built.

2.1 Computational Complexity and NP-hardness

While this subject is not critical to understanding the rest of the thesis, it is the underlying reason why methods without optimality guarantees, such as genetic algorithms, are considered when attempting to solve job shop problems.

Computational complexity is the study of solving computational problems in terms of time and space (memory), especially when the problem is large. The complexity of an algorithm is usually expressed in big-$\mathcal{O}$ notation, where $\mathcal{O}(f(n))$ describes how the number of operations grows with the input n. Algorithms whose run time grows polynomially, such as $\mathcal{O}(n^2)$ or $\mathcal{O}(n^3)$ scales well and are considered efficient, while algorithms with exponential complexity, $\mathcal{O}(2^n)$, quickly become computationally infeasible [14]. Figure 2.1 illustrates how exponential growth quickly outpaces polynomial growth even for relatively small input sizes.

Complexity Classes

Problems can be grouped into different classes based on how difficult they are to solve or verify.

P is the class of problems that can be solved in polynomial time, e.g. sorting a list. These problems are considered efficiently solvable [14].

To verify a solution means checking whether a proposed candidate answer is correct, without necessarily having to find the answer in the first place. Some problems are unsolvable in polynomial time, but if a solution already exists, it can be verified efficiently. These problems belong to the class **NP** [14].

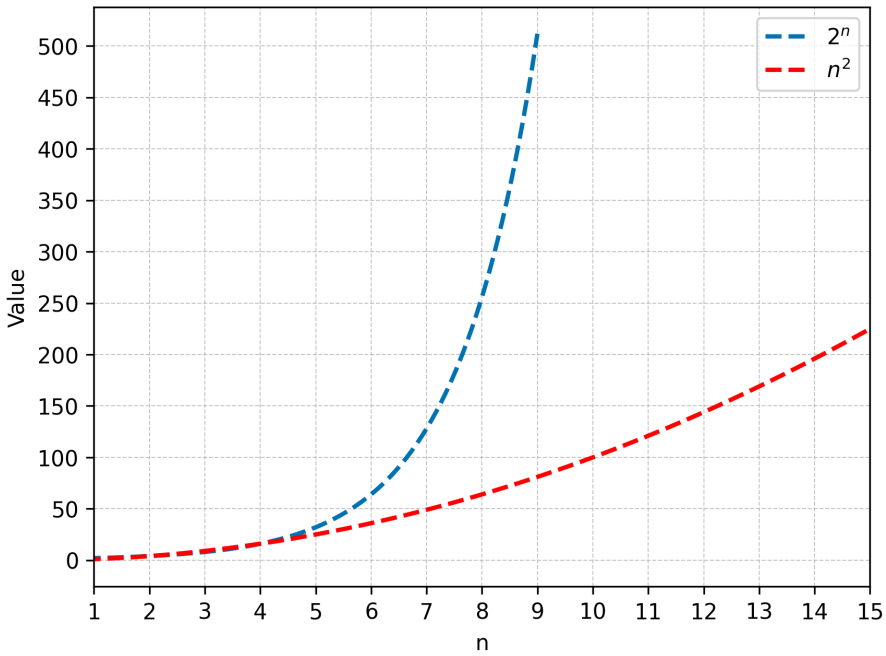


Figure 2.1: Comparison between polynomial growth (n^2) and exponential growth (2^n). Exponential-time algorithms become impractical much faster as input size increases.

Importantly, $\mathbf{P} \subseteq \mathbf{NP}$: any problem that can be solved efficiently can be verified efficiently. Intuitively, this is because if a problem can already be solved in polynomial time, a proposed solution can be verified by solving the problem independently and comparing the result to the proposed one, which is itself a polynomial-time procedure. Whether the two classes are equal, meaning whether every problem whose solution is easy to verify can also be solved efficiently is the famous question of $\mathbf{P} = \mathbf{NP}$ [15].

A problem is **NP-hard** if every problem in NP can be transformed into it via a polynomial-time reduction. This means that an efficient solution to any NP-hard problem would imply an efficient solution to every problem in NP [15]. NP-hard problems are not required to be NP and does not have to be verifiable in polynomial time. This is a characteristic displayed by many optimization problems and is the motivation for considering heuristic and metaheuristic approaches.

NP-complete problems are those that are both in NP-hard and in NP. They are the hardest problems within NP, and a polynomial-time algorithm for any single NP-complete problem would prove P = NP [15].

2.2 Scheduling Theory

The goal of scheduling is to allocate resources to different tasks to be completed over given time periods, while trying to minimize or maximize some objective. The objectives may include maximizing profit, meeting due dates for a customer, or min-

imizing maximum completion time $C_{\max}$ (makespan) of all jobs in a JSP. Minimizing makespan is related to maximal utilization of the system's machines. The completion time C_j, denotes the time at which a job finishes on the last machine it requires processing and leaves the system. The total completion time, $\sum C_j$, measures how long time the tasks spend in the system [16]. Metrics for meeting due dates are tardiness, lateness and number of tardy jobs. Tardiness of a job is defined as

$$T_j = \max(0, C_j - d_j),$$

where d_j is the due date of the job. To measure total tardiness, the sum of the job's tardiness is used, $\sum T_j$ [1]. Lateness is defined as

$$L_j = C_j - d_j,$$

which in contrast to tardiness allows for negative values, meaning that early finished jobs benefit the objective [17]. This allows total lateness $\sum L_j$ to optimize both meeting due dates and finishing jobs early. Maximum lateness $L_{\max} = \max(L_1 \dots L_n)$ measures the worst violation of the due dates [1].

Scheduling plays an important role in manufacturing systems, but also in a wide variety of fields, such as transportation, construction, and computing [1], and the variety of problem types are large [17].

Scheduling Models

There are many standard problem formulations based on manufacturing, and they have many additional constraints such as whether preemption is allowed, precedence constraints, sequence dependent setup times, and recirculation. Here, an overview is given of the problem formulations and properties relevant for this project. Examples of scheduling environments are single machine, parallel machines, flow shops, hybrid flow shops, job shops, and flexible job shops.

Scheduling on a *single machine* is the simplest example of machine environments and serves as a building block for more complicated environments [1]. For a single machine, an optimal polynomial algorithm can be constructed for many cases; however, already in this simple environment, the problem can become NP-hard. Examples of an NP-hard problem on a single machine is optimizing $\sum T_j$, $\sum C_j$, and $L_{\max}$ for jobs with different release dates [17].

Parallel machines scheduling refers to when a job can be processed by m machines. The job j can be processed by any machine and the different machines may or may not have different processing speeds. The job may also have constraints for which machine it can be processed. The only non-preemptive problem that is known to be polynomial solvable is optimizing $\sum C_j$ on unrelated parallel machines with different processing times for each job and job dependent speeds for each machine. Other problems such as minimizing makespan or the weighted total completion time on two parallel machines are NP-hard [1].

The *flow shop* refers to m machines in series, every job has to be processed on all m machines and all jobs follow the same route, meaning that all jobs have to be processed on every machine in order. In the flow shop, a job cannot usually pass another by waiting in a queue for the next machine, but the jobs when dispatched always follow first in first out (FIFO) order on every machine, this is referred to as a *permutation flow shop*. If job sequence change is allowed, it is possible to find an optimal schedule by only making sequencing changes between the first two and the last two machines in a flowshop [1]. The only problem with arbitrary processing times that is polynomially solvable is minimizing makespan in a flow shop with two machines in series: for all other problems with arbitrary processing times, flow shop problems are NP-hard [17].

The flexible or *hybrid flow shop* has c stages in series and at each stage there are m identical machines in parallel. Each job has to pass through all the stages in the shop but only require processing at one machine at each stage. This problem setting is quite complex. However, if the processing times at each stage are the same for a job, one can show that scheduling in non-decreasing processing time is optimal for $\sum C_j$ if every stage has at least as many machines as the preceding stage [1].

The *job shop* is an environment where there are m machines and every job has to follow its own predetermined route. Each operation in the route can be performed by one of the machines and has an associated processing time. There are only a few job shop problems that can be solved in polynomial time [1].

The *flexible job shop problem* (FJSP) generalizes the job shop problem by allowing one or more operations to be processed by alternative machines. The shop is divided into c workcenters that contain a number of identical parallel machines. The jobs have their own routes to follow through the shop's workcenters. In each workcenter, the job requires processing on only one of the machines. The FJSP becomes too hard to analyse already for the case where all processing times for a job are equal on all machines [1].

Scheduling in manufacturing systems

Since the 1950s, tools have been developed to collect and process information in corporations. A *Master production schedule* (MPS) [18] is a plan of the quantities of each product produced in the production line. The MPS builds a higher level schedule of what parts to build, quantity and their due dates based on demand and forecasting [19]. Starting in the 1950s *Material requirement planning* (MRP) was developed for inventory management and bookkeeping. The MRP takes the MPS as input and computes the raw materials and components required to produce the final goods [18]. This early system was focused on ordering materials based on needs. Production of parts, products, and components, is triggered by minimum inventory levels in the MRP. It calculates the production quantities of all items and the start time of the jobs required to meet due dates. Problems with the MRP system was that the capacity was assumed to be infinite, since it did not take into account how

much work was already in the manufacturing plant. In particular, the lead times are assumed to be fixed and depend only on the part number, and not on the load of the plant. Since real world production does not have infinite capacity the fixed lead times, MRP models are an approximation of reality, which has been shown to cause increased congestion and flow time, worsening inventory levels and customer service [19].

Capacity planning is used to determine what resources of labor and equipment are required to achive the MPS. Capacity planning can be performed at three stages of planning, using *resource requirement planning* (RRP), rough-cut power planning (RCPP), and capacity requirement planning (CRP). The RRP ensures that the production output level is feasible in the long term plan, while the RCPP determines MPS's viability. CRP determines whether the different production cells have sufficient production capacity to complete the quantities of specific parts and subassemblies determined by the MRP [18].

Manufacturing resource planning (MRP II) [18] is a computer system that plans and controls materials, resources, and supporting activities to achieve the MPS. It contains the MRP functionality and includes capacity planning, demand management and forecasting functions. It provides long term resource planning and aggregate planning that determines the level of production, staffing, inventory, and overtime in the long term. Further, there is short term control that arranges the jobs to machines using dispatching rules. A drawback is that no dispatching rule works well all the time, since they only work well locally. A good schedule must take the shop as a whole into consideration. However, this leads to very complex computations and the resulting schedules are not always intuitive [19].

An *Enterprise Resource Planning* (ERP) [20] system is an information system that integrates e.g. sales, quality control, product processing, and supply management information, and provides functions to make decisions fast. The ERP system focuses on controlling the entire enterprise, and includes functions for e.g. supply management [19]. Well integrated ERPs help make decisions to reduce e.g. inventory redundancy, meet due dates, and increase productivity [20]. However, implementation of ERP can be expensive and can lead to failure and underutilization [20]. With ERPs one can integrate a *Manufacturing Execution System* (MES) and an *Advanced Planning System* (APS). MES tracks work in process, records process, executes schedules and releases new jobs to the system etc, while APS includes finite capacity scheduling, forecasting, warehouse management etc. [19]. ERPs are often underutilized due to that the system is not able to achieve all business objectives [20].

Digital Transformation in Manufacturing

Industry 4.0 [21, 22] is a sociotechnical concept that collects the modern digital transformation and industrial development. It was coined in 2011 in Germany to describe the "fourth industrial revolution" and a strategic plan to strengthen Germany's competitive position in manufacturing. Different initiatives have been

taken to describe the developments in industry between technology and operation management, Industry 4.0 is one label that has prevailed. There is no universal definition of Industry 4.0, but literature studies attempt to collect the central concepts [21, 22, 23]. Industry 4.0 contains older concepts such as ERP, CAD and computer aided manufacturing but its aim is to describe a new phase in manufacturing with new information and communication technologies [21]. The concept is a mix of political ambitions and technological developments that describes how industry has developed. It encapsulates human, technological and organisational perspectives on industry. Decentralization in decision making, increased autonomy, and new knowledge and capabilities are central aspects. Technological development that characterizes Industry 4.0 are big data, decentralization, self organized entities supported by decision making of artificial intelligence, and flexibility customization and volumes. Customization in industry covers customized products and customized production. Efficiency is another aspect of Industry 4.0, which emphasizes efficiency in the production process, value creation process, digital integration, and intelligent manufacturing processes, in order to make factories ”smart and adaptable”. Integration is another keyword that describes ERPs as flexible and reconfigurable. Manufacturing systems have analytical and simulation based approaches integrated in the business processes. Data management involves new levels of data integration and data processing with help of data science, analytical models, data mining, and big data [22]. It is essential that workers are supported by easily interpretable data.

Smart manufacturing [24, 21] is a concept related to Industry 4.0, attempting to describe the phenomenon. Smart manufacturing deals with greater manufacturing complexities and different performance objectives that require real time information based technologies. This means that the workforce with more advanced training and skills is a key to stay competitive with increasing dynamic management and demand driven product profiles. The workforce will require training in performance oriented decision making [25]. Since manufacturing becomes more automated and complex tools such as computing platforms, control simulation, data intensive modeling and predictive engineering become more relevant in data driven decision making in manufacturing. Like Industry 4.0, Smart manufacturing has no general agreed upon definition but is described as; ”a fully integrated collaborative manufacturing system that responds in real time to meet changes in demand and conditions in the factory, in the supply network and customer needs”, by National Institute of Standards and Technology (NIST) [24].

2.3 Lean manufacturing and JIT

Just in time (JIT) is a manufacturing technique that laid the ground for the Japanese manufacturing success, and specifically associated with the success of Toyota Motor Company [19]. JIT shows that the production environment itself is a control parameter and is characterized by low inventory and flow oriented focus on manufacturing that later expanded into the broader view of Lean manufacturing [19]. The philosophy and tools of Lean are vast; here an overview is given that relates concretely

to production and scheduling. Lean is a socio-technological paradigm and methodology with Japanese roots in the 1970s, that has been widely used since the 1990s. Lean incorporates both technical and social aspects of an organization, and was popularized by Toyota. The Lean philosophy puts customer value in the centre of the process development, it emphasizes producing the product the customer wants, delivering on time and minimizing non-value-adding production activities [18]. The goal is to obtain an "ideal" which can be described as being complete flexibility eliminating waste and standardize work. There are three main concepts central to Lean [26], waste (Muda), overburden (Muri) and unevenness (Mura).

Waste is generally defined as anything that does not add value to the end product [18], examples are waiting times, unnecessary, motion and transportation [26]. The methodology includes creating flow in the system and applying a pull system, responding to the customer's demands when producing. Ideally, the production layout is set up in cells with minimal inventories between stations [26]. Lean recognizes that job shop layout in production leads to complex scheduling routes and seeks to reduce complexity by reorganizing the production line. Changeover, such as fetching tools and moving between machines is seen as waste that should be reduced. Shared resources are recognized as bottlenecks that introduces complexity and should be minimized; however, sometimes shared resources are unavoidable. If shared resources are required, they lay ground for capacity analysis. To create "flow", Lean promotes one piece flow over batch building, valuing that products are leaving the factory earlier, promoting on time delivery [26]. JIT is the concept Lean uses to express that products should be delivered on time and components to be produced when needed. The goal of JIT is to reduce defects, replenish inventory when an item is taken, reduce setup times, breakdowns, handling time and changes is the production plan. JIT also encompasses that sub-assemblies and material should ideally be produced or bought when needed, to minimize inventory. Orders are triggered by *Kanban* signals from a downstream workstation, and similarly, when the supply of required parts (material or sub-assemblies) are ordered upstream the production flow [18]. Kanban production is triggered by demand, if a product is removed from finished goods it signals the station upstream to replace it, and each preceding station is replenishing the parts needed downstream. The layout can be compared to flow shops and hybrid flow shops, with the addition of minimal inventory between stations. The layout aims to even out work distribution, and reduce the need of waiting for a machine or transporting a product [26]. The workers in a JIT system are cross-trained in order to move workers where they are needed, keeping multiple skills sharp by rotating the workers between production cells periodically. Scheduling is a central part of Lean, the tools for setting up the production line are the means to reduce complexity and improve schedule performance [19]. The role of production planning is described as analyzing capacity and routing, with less involvement in execution and monitoring. The use of MRP/ERP is considered bad for execution, based on the vast implementation in the late 1990s an early 2000s showed mixed results for orgainzations at a high cost. However MRP/ERP can be considered useful for planning and analysis to find complex flows and shared resources [26].

JIT and Lean are attitudes, philosophies and priorities that describe an ideal goal of production. When JIT spread from Japan it was evident that the Toyota model could not be copied directly since it was specifically tailored to their manufacturing system. In [19] an historical overview of JIT's migration from Japan to the U.S. is presented. The implementation of JIT is discussed to have been overly simplified with an idealized view of zero inventories and Kanban systems where assumed to be easy to implement. [19] concludes that, since manufacturing environments are different and there cannot be a simple uniform solution for manufacturing, it is up to each firm to find their own appropriate policies.

Industry 4.0 tools supply information and data that enable scheduling research to shift focus to real time and smart scheduling and product flow optimization, and bridge the gap between reality and offline scheduling and mathematical models and building a "smart lean manufacturing system". Such models have been suggested using FJSP with job pools, called multistage parallel flexible job shop to create a real time dispatching rule in a lab setting [27].

2.4 Heuristics

Heuristic algorithms [28] can be used to find approximate solutions when exact optimization methods are computationally impractical. In scheduling, such algorithms are often referred to as *dispatching rules*, *scheduling heuristics* or *priority rules*. These methods can be explained as "rules of thumb" to prioritize assignments of jobs [28]. One class of heuristics is greedy algorithms that constructs schedules by iterative selection of operations and assignments according to some priority rule [1]. Greedy algorithms build solutions iteratively in small steps, making decisions based on the information available at each step. When a greedy algorithm manages to solve a problem optimally or close to optimal, they typically reveal something useful about the structure of the problem itself [29]. Common examples of such algorithms are *shortest processing time* (SPT), *earliest due date* (EDD) and *earliest release date* (ERD) [1]. These algorithms are simple and fast but their optimality is limited to specific problem settings. For example, SPT optimizes the average completion time (or flow time) on a single machine, EDD minimizes maximum lateness and tardiness on a single machine, under the specific conditions such that there are no precedence constraints, all tasks are available at $t = 0$, and there are no setup times [28]. *Longest processing time (LPT)* is used for scheduling on parallel machines by assigning jobs in nondecreasing order of processing time. Although optimality is not generally guaranteed for LPT, its worst case performance has a sound theoretical basis. The worst case performance ratio is

$$\frac{4}{3} - \frac{1}{3m},$$

where m is the number of machines. The LPT rule is optimal under specific conditions, such as the trivial case when the number of jobs n are less then than number of machines $n < m$, or when structural conditions on processing times are met [1]. LPT tends to distribute workload evenly across machines and is therefore often used

as a load balancing heuristic. LPT is also used as a building block to design more advanced scheduling algorithms, such as composite dispatching rule. A simple example from [1] is minimising makespan on a proportionate permutation flow shop where either the first or last machine is the bottleneck. If the first machine is the bottleneck LPT is used and if the last machine is the bottleneck SPT can be used [1].

2.5 Linear programming and the Simplex Algorithm

Linear programming (LP) [30] is a mathematical optimization technique, in which a linear objective function is optimized subject to a set of linear equality and inequality constraints. It is particularly useful for problems that can be modeled linearly, such as transportation planning [31], where linear programming is used to minimize transportation cost by determining optimal routes between suppliers and destinations. Another common application is job shop scheduling [1], where it is used to determine the optimal assignment of a number of jobs to a given set of machines. A linear programming problem can be expressed in its canonical form as:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0, \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the vector of decision variables, $c \in \mathbb{R}^n$ are the objective function coefficients, and $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ defines the constraints of the problem.

The constraints of the problem define what is called the *feasible region*, that is, the region where all possible solutions to the problem exist. In the general case, the feasible region is a convex polytope defined by the intersection constraints $Ax \leq b,\ \forall i\ x_i \geq 0$ [32]. To illustrate this concept, a two dimensional example will be considered. Figure 2.2 shows the lines $y = -x + 7$ and $y = -\frac{1}{2}x + 5$. Together with the constraints $x \geq 0$ and $y \geq 0$ the feasible region is created and shown as the blue-shaded area.

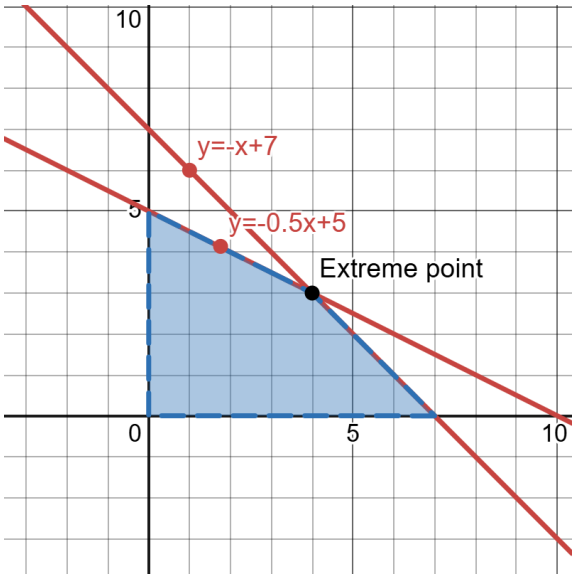


Figure 2.2: Feasible region defined by the lines $y = -x + 7$ and $y = -\frac{1}{2}x + 5$ and non-negativity of x and y.

Since the constraints are linear the resulting feasible region will be (in general) a convex polytope. The convexity of the feasible region is important because it guarantees the fact that, if an optimal solution exists, it will occur at an extreme point [33]. This property is particularly important, as it reduces the search space from a potentially infinite set of feasible points to a finite set of vertices [34]. The significance of the vertex optimality property is demonstrated by the *simplex algorithm* [35], one of the most prominent methods for solving linear programs. Rather than exploring the interior of the feasible region, the simplex method restricts its search to the boundary, specifically the extreme points of the polytope defined by the constraint set.

The algorithm begins at an initial basic feasible solution (BFS), corresponding to a vertex of the feasible polytope. Algebraically, a BFS is characterized by a basis: a set of m linearly independent columns selected from the constraint matrix $A \in \mathbb{R}^{m \times n}$, where m is the number of constraints and n the number of variables. The m variables corresponding to these columns are called *basic variables* and are solved for explicitly, while the remaining $n - m$ *non-basic variables* are fixed at their bounds, typically zero.

At each iteration, the algorithm evaluates the reduced costs of the non-basic variables. To determine whether introducing any of them into the solution would improve the objective value. If a variable with a negative reduced cost is found, it becomes the *entering variable*, and a *ratio test* determines which current variable must leave to preserve feasibility. This is called a *pivot operation*, and geometrically corresponds to moving along an edge of the polytope to an adjacent vertex with a better objective value.

The algorithm terminates when no basic variable has a negative reduced cost, guaranteeing that the current solution is optimal [36]. Despite its theoretical worst-case exponential complexity, the simplex algorithm remains highly efficient in practice for most real-world applications. Note that linear programming itself is of complexity class **P** as demonstrated by the ellipsoid method [37] and then later interior-point-methods [38]. This stands in contrast to integer programming, where the requirement that variables take integer-values makes the problem **NP-hard** in general [39].

2.6 Linear Programming in the Integer Domain

In many real-world applications, decision variables are required to take integer values. Typical examples would be the number of vehicles assigned to a route or number of products produced in a factory. In these cases fractional values are not meaningful. When such integrality requirement is imposed on some or all of the decision variables, the resulting problem is known as an integer programming (IP) problem, or a Mixed Integer Linear Programming (MILP) [40] problem if only a subset of the variables are restricted to integers.

An integer programming problem can be written in the form of:

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \\ & x_i \in \mathbb{Z}, \quad i \in \mathcal{I}, \end{aligned} \tag{2.2}$$

where $\mathcal{I} \subseteq \{1, \ldots, n\}$ denotes the set of variables restricted to integer values.

While the objective function and constraints remain linear, the integrality constraints changes the feasible region. By restricting variables to discrete points, the feasible set is no longer a convex polytope, but rather a set of isolated points contained within such a polytope. Figure 2.3 shows this for previous example. This loss of convexity means that the optimal integer solution might not appear at any vertex, which in turn makes vertex based methods such as the simplex algorithm inapplicable directly. IP problems are NP-hard in general [39], which is the reason they are significantly harder to solve than LP problems.

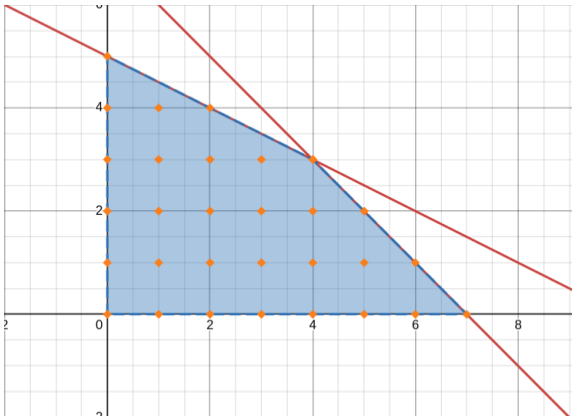


Figure 2.3: Feasible region for the integer programming problem defined by the lines $y = -x + 7$ and $y = -\frac{1}{2}x + 5$ and non-negativity of x and y.

One method to solve IP and MILP problems is the **branch and bound** algorithm. The idea of the algorithm is to systematically divide the search space into smaller sub-problems by recursively solving LP relaxations, that is, the removal of the integrality constraint. The algorithm is executed as follows:

1. **Relaxation:** Ignore the integrality constraints $x_i \in \mathbb{Z}$ and solve the problem as an LP. If the solution is integer-valued, the solution is optimal for the IP. Otherwise at least one variable is a fractional value.

2. **Branching:** Choose a fractional variable x_j and create two sub-problems by adding the constraints

$$x_j \leq \lfloor x_j^* \rfloor \text{ and } x_j \geq \lceil x_j^* \rceil$$

   Where x_j^* is the fractional value from the LP solution.

3. **Bounding:** Solve the LP relaxation of each sub-problem. If the resulting bound is no better than the best integer solution found so far, the branch is pruned.

4. **Pruning:** A branch is pruned when the solution has one of the following properties:

   - its LP relaxation is infeasible,
   - its bound is no better than the current best integer solution, or
   - its LP solution is already integer-valued.

This process continues recursively, forming a tree of sub-problems and is terminated once all branches have been solved or pruned [40].

The branch and bound algorithm is illustrated by the following example. Consider the IP:

$$\begin{aligned} \text{Maximize} \quad & z = 5x + 8y \\ \text{s.t.} \quad & x + y \leq 6 \\ & 5x + 9y \leq 45 \\ & x, y \geq 0 \\ & x, y \in \mathbb{Z} \end{aligned}$$

The solution to the initial relaxation of this problem gives the solution:

$$z = 41.25, \; x = 2.25, \; y = 3.75.$$

But this is not a valid solution since we require x and y to be integer. The procedure of the branch and bound algorithm is shown in Figure 2.4

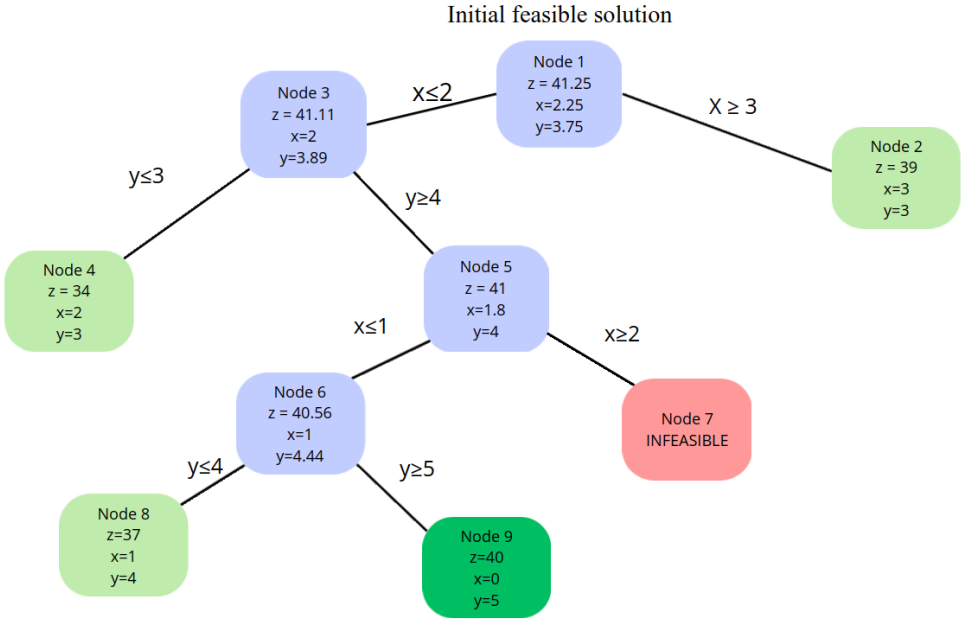


Figure 2.4: Tree of sub-problems created by the branch and bound algorithm

At Node 1, x is chosen as the branching variable and the two sub-problems $x \geq 3$ and $x \leq 2$ are created. At Node 2, an integer solution is found with $z = 39$, meaning that this branch can be pruned. The process continues since the other branch still has potential to yield a better solution. Node 3 yields a fractional solution, but still has potential to give a better result. Node 4 finds an integer solution that is worse than Node 2, but the right hand side of Node 3 might reveal an even better solution. The same argument as for Node 3 goes for Node 5 and Node 6. Node 7 is infeasible meaning it is pruned. Finally, Node 9 is found with the best integer solution, and since the other branches have worse integer solution or are infeasible this becomes the optimal solution for the IP.

2.7 Satisfiability Modulo Theories

Another approach to solving different job shop problems is by using Satisfiability modulo theories (SMT) [6]. To understand SMT it is first necessary to have an understanding of Boolean satisfiability (SAT) [41].

Boolean Satisfiability (SAT)

A SAT problem asks whether, given a propositional formula, there exists and assignment of truth values (*true*, *false*) to each variable such that the formula evaluates to *true*. Propositional formulas are built using logical connectives such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), implication ($\Rightarrow$) and the quantifiers ($\forall, \exists$). A formula that has at least one satisfying assignment is called *satisfiable*, while one that does not is called *unsatisfiable*.

This can be illustrated through the following example:

$$(P \wedge \neg Q).$$

One can find the assignment $P = true$ and $Q = false$, which inserted into the given formula would evaluate it to *true*, meaning that the formula is satisfiable. An example of a unsatisfiable formula could be $b \wedge \neg b$, since no assignment of b can simultaneously satisfy b and its negation.

In practice, SAT problems are commonly expressed in *conjunctive normal form* (CNF) which is a representation where formulas are written as a conjunction of *clauses*, each of which is a disjunction of *literals* (a variable or its negation), like:

$$(P \vee \neg Q) \wedge (\neg P \vee R) \wedge (Q \vee R).$$

CNF is important since many SAT solvers, such as those based on conflict-driven clause learning (CDCL) [42], operate directly on CNF formulas. SAT solvers are able to handle large models efficiently and are often used in hardware verification.

However, SAT solvers are limited to Boolean variables, which makes them efficient for modelling binary constraints but rather inefficient when the problem requires reasoning over integers, real values, or other data-structures.

Satisfiability Modulo Theories

Satisfiability modulo theories (SMT) extends the SAT framework by allowing some variables and constraints to be replaced by *predicates* (binary valued functions over non-binary variables). For example, the constraint $x > 3$ is a predicate over the integer variable x, and evaluates to *true* or *false* depending on the value of x. This

allows the SMT solver to use logical conditions over e.g. linear inequalities. To illustrate, consider the formula:

$$\neg(x \leq 0) \Rightarrow (y > x \vee \neg(y = 5)). \tag{2.3}$$

Let x and y be integer variables. An SMT solver would find that one satisfying assignment would be $x = 2$ and $y = 3$, making the formula satisfiable.

Optimization Modulo Theories

While SMT solvers determine whether a formula is satisfiable, many practical problems require not just any satisfying solution, but the optimal according to some criterion. This leads to optimization modulo theories (OMT) [43, 44] which extend SMT by adding an objective function to be minimized (or maximized) subject to logical constraints.

To illustrate, consider a scheduling problem where x represents the completion time of a job. A set of SMT constraints ϕ might encode precedence, and resource requirements, while the objective function $f(x) = x$ is to be minimized. A regular SMT solver would return any schedule that satisfies all constraints; an OMT solver would return an optimal one.

OMT solvers typically find the optimal solution through an iterative process. The most basic of which is a linear search. Firstly, a satisfying assignment is found, and then an additional constraint is introduced which states that the objective has to be strictly better than the current solution. This process repeats until no satisfying assignment can be found and the last valid solution is declared optimal [43].

A key advantage in OMT compared to classical methods such as MILP is the ability to handle logical constraints. Constraints involving disjunction or implication, which would require significant reformulation using MILP, can be expressed directly in OMT. This makes OMT particularly suited for problems such as scheduling and planning, where both logical and arithmetic constraint arise simultaneously.

2.8 Constraint Programming

Constraint programming [8] is a declarative paradigm for solving complex combinatorial, scheduling, and optimization problems by defining variables, their constraints, and domains.

A *constraint satisfaction problem* (CSP) [8] can be described by a triplet $\langle X, D, C \rangle$, where $X = \{x_1, \ldots, x_i\}$ is a set of decision variables. $D = \{D_1, \ldots, D_i\}$ is a set of domains where D_j is the domain of x_j ($j \in 1 \ldots i$), and C is a set of constraints over a subset of the variables.

Types of Constraints

Constraints in a CSP can be classified by the number of variables they involve. *Unary* constraints restrict the domain of a single variable, while *binary constraints* relate two variables. *Trinary constraints* relate three variables, and so on. Constraints involving three or more variables will be referred to as *higher-order* or *global constraints*.

Global constraints are a defining feature of CP. Rather than decomposing complex relationships into many binary constraints, a global constraint captures high-level structure directly and is equipped with dedicated, efficient propagation algorithms. For example, the `allDifferent` constraint enforces that all variables in a given set take distinct values and can be propagated in polynomial time [45]. For scheduling, constraints such as `NoOverlap` are particularly useful as it guarantees that no two tasks overlap on a single resource.

Constraint Propagation

Constraint propagation is central to the process of solving a CSP. Constraint propagation involves reasoning that explicitly rules out certain values or combinations of values for variables when a subset of constraints cannot be satisfied otherwise.

Constraint propagation is used to reduce the domains while maintaining consistency. When a CSP has one (or more) solution(s) it is called *consistent*. If no solutions exist, it is *inconsistent*, and *solved* if all domains are singletons and together form a valid solution.

There are different levels of consistency [8]. The first is *node consistency* which ensures that every value in a variable's domain satisfies all unary constraint on that variable. *Arc consistency* is the next notion of consistency. A constraint $c(x_i, x_j)$ between two variables x_i and x_j (binary constraint) is arc consistent if, for every value $a \in D_i$, there exists at least one value $b \in D_j$ such that the constraint is satisfied, and vice versa. There are more levels of consistency such as *path consistency* which relates two variables to a third (trinary constraints), and *k-consistency* which means that every consistent assignment if $k-1$ variables can be extended to a k-th.

Propagation alone is generally not sufficient to solve a problem, but it significantly reduces the number of assignments that need to be explored during search.

Search

When constraint propagation cannot determine a unique solution, search is employed. CP solvers typically use *backtracking search* [46], which systematically explores the space of assignments. At each step, the solver selects an unassigned variable, assigns it a value from its domain, and performs propagation. If a contradiction is detected (some variable's domain becomes empty), the solver backtracks and tries a different value.

The efficiency of backtracking search is heavily influenced by variable and value ordering heuristics. The *fail-first* heuristic assigns priority to the variable with the fewest remaining values, aiming to detect infeasibility as early as possible [46]. Value ordering heuristics, such as selecting the least-constraining value, attempt to preserve flexibility for subsequent assignments.

More advanced strategies such as *branch and bound* extend backtracking search to optimization problems by maintaining an upper (or lower) bound on the objective value and pruning branches that cannot improve upon the best solution found so far.

The combination of constraint propagation, search algorithms, and global constraints with dedicated algorithms makes CP a valuable alternative for solving complex combinatorial problems.

2.9 Genetic Algorithms

Besides exact optimization methods, there exists a group of meta-heuristic approaches. One prominent class of such methods are Genetic Algorithms (GAs) [10], which are population-based stochastic search techniques inspired by natural evolution.

A genetic algorithm maintains a population of candidate solutions, where each solution is encoded in a suitable representation (often referred to as a chromosome). Each candidate solution is evaluated using a fitness function that corresponds to the value of the objective function of the optimization problem. The algorithm iteratively generates new solutions by applying a set of evolutionary operators [47]. The size of the population is defined as N.

The algorithm operates as follows [47]:

1. First an initial population of candidate solutions are generated and encoded.
2. Evaluate individuals
   (a) Decode the chromosome
   (b) Evaluate the objective function
   (c) Repeat for the entire population
3. Generate the next generation
   (a) Elitism - an exact copy of the best individual is made

(b) A subset of the current population is selected for reproduction, where individuals with a higher fitness value are typically selected with a higher probability.

(c) The selected individuals are then combined using the crossover operator to generate new individuals (solutions).

(d) A mutation operator introduces (typically) small random variations to individual solutions, which help prevent premature convergence to poor local optima.

(e) The old population is replaced with the new population.

4. Repeat the process from step 2, until termination criterion has been reached.

Each iteration of the steps is referred to as a *generation*, and the set of all generations is referred to as a *run*. At the end of a run the population tends to contain one or more highly fit solutions [48].

Encoding

Chromosomes represent a possible solution to a problem, and its encoding depends on the problem being solved. The chromosomes consists of m digits, referred to as genes. There are for instance, binary encoding, real valued encoding and permutation encodings. The binary encodings are strings of 1s and 0s. They are commonly used in the theoretical study of GA, since they simplify an analytical approach. For practical implementations of GA, binary encoding is usually limiting. Instead, real valued encoding is more commonly used, where each gene is represented by a real number. For combinatorial problems, a permutation encoding can be used in which the genes are represented by integer values and their ordering represents the solution to the problem [47].

Selection

Selection is the process that chooses the individuals that will generate the next generation of individuals. This can be done in different ways; however, the most common are roulette wheel and tournament selection [47]. In roulette wheel selection an individual is selected proportionally to its fitness. Tournament selection resembles more how selection happens in nature. A number k individuals are randomly drawn among the population and are ranked by fitness. The best individual among these is selected with probability p_{tour}, if the individual is not selected the second best individual is selected with probability $p_{tour}(1-p_{tour})$. This continues down the ranked list, so the i-th ranked individual is selected with probability $p_{tour}(1-p_{tour})^{(i-1)}$ [47]. If $p_{tour} = 1$ the selection is deterministic ensuring that the fittest individual is always selected.

Crossover

Crossover is a fundamental operator in GAs; it combines partial solutions of different individuals and assembles new individuals. The function of crossover is to allow for a wide-range search of possible solutions. The crossover operator allows a fit individual to spread quickly across a population. Therefore, crossover may not always be beneficial since it may reduce the diversity of the population quickly, trapping the population in a local optimum. Crossover is therefore performed with a certain probability to cross two selected individuals, if crossover is not performed the individuals are simply copied to the next generation. The crossover operator can be designed in different ways, e.g., *single point crossover* chooses one of the $m-1$ possible points between the m genes in the two parent chromosomes randomly, and then lets the offspring inherit the genes to the left from one parent and the genes from the right in the other, and vice versa for the other offspring [47]. *Multi point crossover* swaps multiple randomly chosen segments between parents.

Mutation

Mutation is important to provide new information to make evolution to work. When a new individual is generated through crossover, it might be mutated. This is done by randomly generating a new gene with probability p_m. Generally, each gene undergoes a mutation with p_m [47].

Elitism

The best individual is very likely to be selected by crossover but it is likely that it is destroyed during crossover. To ensure that the best individual is not lost, an exact copy is always transferred to the next population, known as *elitism* [47]. One can also keep more copies of the best individual(s) without modification. If the population N is even and an odd number k of individuals are kept through elitism, then $N-k+1$ individuals are selected from crossover, and the last individual can for instance be formed through asexual reproduction, using only mutation [49]. Another way of dealing with an odd number of offsprings is to generate one extra and cut it off from the next generation.

Parameter Selection

It is non-trivial to select parameters for the operations of a GA. The parameters have to be selected for the specific problem to be solved [47]. There are multiple methods to search the parameter space; a straightforward approach is to use the Cartesian product of a set of parameters and evaluate GA performance across all combinations. There are other methods that can be employed to find a suitable set of parameters [50], such as sampling a Latin hypercube, random search or using an evolutionary algorithm.

3

Method

This chapter provides a description of the manufacturing environment and the assumptions made to translate it into a DR-FJSP representation. It also presents a canonical model formulation based on the Thorlabs production environment, the design of the genetic algorithm, and the approach of heuristic algorithm. Furthermore, it describes the methodology used to evaluate these approaches and the performance metrics used for evaluation.

3.1 The Manufacturing Environment

To better understand the models developed, it is useful to first establish how the production environment at Thorlabs operates. As mentioned in the introduction, the environment at Thorlabs is mostly manual and organized into work- and test-benches. Workers assemble products at workbenches and then verify their functionality at different test-benches. The production is then further divided into production cells, each containing product pools that are designed to process a category of products. Jobs within a pool may be completed using the workbenches and tools within the pool. The pools locally resemble flow shops and HFS. Note that the local structure does not make the problem trivial, since minimizing total completion time for flow shops of two machines, and minimizing makespan for flow shops of size three are both NP-hard problems [1].

In addition, the production cell is not a collection of independent flow shops. Some product routes are constrained to the pools while some are coupled to machines shared across pools, creating shared bottlenecks. Furthermore, the workers are qualified to process different jobs. If a worker is qualified for a job, they are able to perform all operations within said job. The skills of the workers are diverse, some are specialized at jobs within one or a few pools, while some senior operators are qualified across all pools.

Each job has a route associated with it. The routing of the job specifies the number of operations and resources required to complete the job. The routing is fixed, meaning that the operations must be completed in a given order where each operation can either require a worker, a workbench, or both simultaneously in order to start

execution. Some operations also have a setup time, this could for example be the calibration of a machine. Furthermore, if a worker is assigned to a job in a different production pool, they must first clean their working area before departing.

Problem Description and Formulation

The DR-FJSP is defined as follows:

Let $J = \{j_1, \ldots, j_n\}$ denote the set of n jobs. Each job consists of an ordered sequence of operations

$$\mathcal{O}_j = \{o_{1j}, \ldots, o_{a_j,j}\}$$

,

where a_j is the number of operations in job j, and operations must be processed in this order. Let $M = \{m_1, \ldots m_k\}$ denote the set of k machines, and $W = \{w_1, \ldots, w_l\}$ denote the set of l workers.

Each operation o_{ij} has an associated set of eligible machines $M_{ji} \subseteq M$, that are capable of performing operation o_{ij}.

Because the problem is dual-resource, each operation also has a set of eligible workers $W_{ji} \subseteq W$, that are qualified to perform operation o_{ji}.

The objective is to find an assignment of machines, workers, and start times of operations such that a specified performance measure is minimized. In this report, three experimental settings are considered: the first minimizes the makespan, while the remaining two minimize total tardiness.

To model the Thorlabs production system mathematically, certain simplifications and abstractions of the real world are necessary. The following presents the assumptions that have been made.

- **Job, machine, and worker availability**: All jobs, machines, and workers are assumed to be available from time zero. The worker assumption is made for ease of modeling despite operators working flexible hours in practice.

- **Sequence-dependent setup times**: The static setup times present in the ERP system are replaced by sequence-dependent setup times, producing more realistic schedules by including a time cost of switching jobs or reconfiguring a machine.

- **Uniform worker speed**: All workers are assumed to have equal processing times for any given operation. Uniform times were adopted both to promote

balanced workload distribution and because individual performance data was not made available for this project.

- **No preemption**: Once an operation has begun, it cannot be interrupted in favor of another operation.

Given the production environment and these assumptions, a mathematical model can be formulated which is subject to the following constraints.

1. **Assignment constraints**. Each operation $o_{ij} \in \mathcal{O}$ is classified by type:

$$\tau(j,i) \in \{M, W, BW\},$$

corresponding to machine-only (M), worker-only (W), and machine-worker operations (BW). Each operation must be assigned to exactly one of the resource types it requires, which is drawn from its eligible subsets $M_{ji} \subseteq M$ and $W_j \subseteq W$ or both.

2. **Completion time.** The completion time of an operation is defined as the start time of the operation plus the processing time of the assigned resource.

3. **Precedence**. All operations within a job must be processed in their prescribed order. A setup time is applied whenever consecutive operations are assigned to different machines.

4. **No overlap**. No two operations may occupy the same resource simultaneously.

5. **Pool transitions**. A cleaning time is applied whenever a worker transitions between production pools.

3.2 Model Formulation

The following model has been implemented in Gurobi, Z3, and Google OR-Tools CP-SAT. Since the three solver implementations solve the same underlying problem the different formulations are rather similar. To avoid redundancy, only a canonical SMT-style formulation will be presented and explained. The full solver-specific formulations are given in Appendix A. The following is the canonical formulation of the DR-FJSP for the Thorlabs production environment.

Sets and parameters
J – Set of jobs
O_j – Ordered set of operations for job $j \in J$
$O = \{(j,i) \mid j \in J, i \in O_j\}$ – Set of all operations
M – Set of machines
W – Set of workers
$M_{ji} \subseteq M$ – Set of eligible machines for operation (j,i)

$W_j \subseteq W$ – Set of eligible workers for job j
$\tau(j,i) \in \{M, W, BW\}$ – Operation type label
p^M_{jim} – Machine processing time of operation (j,i) on machine m
p^W_{jiw} – Worker processing time of operation (j,i) by worker w
s^w – Worker setup time between different jobs
c – worker cleaning time between job pools
s^m – Machine changeover time between different machines
G – Sufficiently large upper bound on schedule
$pool(j)$ – Job pool label of job j

Decision Variables
$t_{ji} \in [0, G]$ – Start time of operation (j,i)
$T_{ji} \in [0, G]$ – Completion time of operation (j,i)
$y_{jim} \in \{0,1\}$ – 1 if (j,i) assigned to machine m
$z_{jiw} \in \{0,1\}$ – 1 if (j,i) assigned to worker w
C_{max} – Makespan

Objective

$$\min\ C_{max} \tag{3.1}$$

Constraints

1. Assignment constraints

$$\sum_{m \in M_{ji}} y_{jim} = 1, \quad \sum_{w \in W_j} z_{jiw} = 0 \qquad \forall (j,i) :\ \tau(j,i) = M \tag{3.2}$$

$$\sum_{m \in M_{ji}} y_{jim} = 0, \quad \sum_{w \in W_j} z_{jiw} = 1 \qquad \forall (j,i) :\ \tau(j,i) = W \tag{3.3}$$

$$\sum_{m \in M_{ji}} y_{jim} = 1, \quad \sum_{w \in W_j} z_{jiw} = 1 \qquad \forall (j,i) :\ \tau(j,i) = BW \tag{3.4}$$

2. Completion time:

$$y_{jim} \Rightarrow T_{ji} = t_{ji} + p^M_{jim} \qquad \forall (j,i) :\ \tau(j,i) = M \tag{3.5}$$

$$z_{jiw} \Rightarrow T_{ji} = t_{ji} + p^W_{jiw} \qquad \forall (j,i) :\ \tau(j,i) = W \tag{3.6}$$

$$(y_{jim} \wedge z_{jiw}) \Rightarrow T_{ji} = t_{ji} + \max(p^W_{jiw}, p^m_{jim}) \qquad \forall (j,i) :\ \tau(j,i) = BW \tag{3.7}$$

3. Precedence and machine setup. Let m_{ji} and m_{ji+1} denote the machines assigned to operations (j, i) and $(j, i+1)$ respectively.

$$t_{ji+1} \geq T_{ji} + \begin{cases} s^M & \text{if } m_{ji} \neq m_{ji+1} \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in J,\ i = 1, ..., |O_j| - 1 \tag{3.8}$$

4. Machine no-overlap For every pair of operations $(j_1, i_1) \neq (j_2, i_2)$ sharing an eligible machine m, where neither operation is of type W:

$$y_{j_1 i_1 m} \wedge y_{j_2 i_2 m} \Rightarrow (t_{j_2 i_2} \geq T_{j_1 i_1}) \vee (t_{j_1 i_1} \geq T_{j_2 i_2}) \tag{3.9}$$

5. Worker no-overlap with setup For every pair of operations $(j_1, i_1) \neq (j_2, i_2)$ that share and eligible worker w

$$(z_{j_1 i_1 w} \wedge z_{j_2 i_2 w}) \Rightarrow (t_{j_2 i_2} \geq T_{j_1 i_1} + \delta) \vee (t_{j_1 i_1} \geq T_{j_2 i_2} + \delta) \tag{3.10}$$

Where

$$\delta = \begin{cases} s^w & \text{if } j_1 \neq j_2 \\ 0 & \text{otherwise} \end{cases} + \begin{cases} c^w & \text{if } \mathit{pool}(j_1) \neq \mathit{pool}(j_2) \\ 0 & \text{otherwise} \end{cases}$$

6. Makespan definition

$$C_{max} \geq T_{ji} \quad \forall (j, i) \in O \tag{3.11}$$

Here, (3.3), (3.2) and (3.4) model the assignment constraints, that is, if an operation is of type W, exactly one worker and exactly zero machines are assigned. (3.5) through (3.7) define the completion time of operations. For example, (3.5) states that if an operation (j, i) is assigned to machine m, then the completion time T_{ji} is defined by the start time t_{ji} and the machine processing time p^m_{jim}. (3.8) states that operations within a job must be executed in order and that a setup time is incurred if machine assignments differ between operations. (3.9) states that no two distinct pairs of operations $(j_1, i_1) \neq (j_2, i_2)$ are both assigned to the same machine, one must complete before the other begins. (3.10) does the same thing as (3.9) but for workers with the addition that a setup time is added if the worker switches jobs or switches production pool. (3.11) defines the makespan.

Note that G serves as an upper bound on the schedule horizon and must be chosen sufficiently large to ensure feasibility. If the true optimal makespan exceeds G, valid schedules are cut off and the model becomes infeasible. A safe choice is to set G equal to the worst-case makespan, namely the sum of all operation processing times, corresponding to executing every operation sequentially on a single resource. This guarantees that G is always feasible while remaning reasonably tight.

3.3 Heuristic Approach

The heuristics used is a greedy algorithm to assign resources. The greedy rule is presented in algorithm 1, a more detailed outline is given in Appendix A.3.

Algorithm 1 Heuristic algorithm

```
Input:
Jobs, operations, machines, workers
Processing times and feasibility constraints
Output:
Schedule S, performance metrics
Initialize availability tables
Sort jobs according to EDD, SPT, or LPT
Sort workers by increasing flexibility
for each job j do
    Determine feasible workers
    Select earliest available worker
    if multiple workers are available then
        Select worker with lowest flexibility
    end if
    for each operation o in job j do
        Determine feasible machines
        Select earliest available machine
        if multiple machines are available then
            Select machine with lowest flexibility
        end if
        Compute earliest feasible start time
        Compute completion time
        Add assignment to schedule
        Update job, machine and worker availability
    end for
end for
Evaluate schedule performance
return Schedule, Metrics
```

In algorithm 1, the jobs are sorted based on different priorities such as “Created date”, “Start date”, “End date” and total processing time. “Created date” corresponds to the day the order is received, “Start date” when the ERP suggests the order to be processed, and “End date” the deadline for completion. A ”Created date” sorting in non-decreasing order corresponds to a first in first out priorty rule, ”End date” sorting in non-decreasing order generates an EDD schedule, and “Start date” in non-decreasing order mimics the dispatching of the ERP. The total processing time of each job is used to generate SPT and LPT dispatching rules by sorting in non-decreasing and non-increasing order of processing time respectively. Workers are sorted in non-decreasing order of flexibility, meaning the worker with

least qualification will be assigned first and the most flexible worker assigned last. This means that highly flexible workers are reserved for jobs with fewer alternatives. After sorting, jobs are scheduled forward in time, keeping track of which and when machines and workers are busy. For each job, its operations are immediately scheduled in sequence. At every operation, the worker's previous operation is considered and cleaning time is assigned.

Genetic Algorithm Design

The genetic algorithm used is inspired by [51]. This section describes how encoding, decoding, initialization, crossover and mutation are implemented using the Python package pymoo [52]. The Wu-style implementation [51] was chosen due to the similarity in problem formulation, with the main difference that [51] models learning effects and this project models sequence-dependent changeover times. The implementation uses the simplified GA of [51], without VNS, in order to obtain a simpler analysis of the operators of the problem in this project.

Encoding

An individual is encoded as a string of integers, representing operation sequence, machine, and worker assignment. The chromosome is considered split in three sequences, first operation assignment chromosome (OSC), machine assignment chromosome (MAC), and worker assignment chromosome (WAC), shown in Figure 3.1.

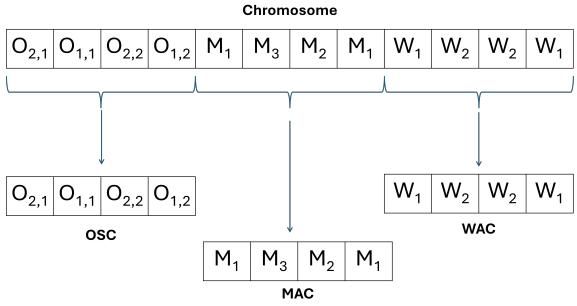


Figure 3.1: Illustration of the encoding of an individual of two jobs and two operations, showing semantic view of the OSC, MAC and WAC.

OSC, MAC, and WAC are concatenated into one sequence representing one individual. The length of the OSC is equal to the number of operations in all jobs. Each operation is represented by the job index and the operation number in a job is identified by the number of occurrences of the job number. The MAC has the same length as the OSC and represents the machine assigned to the corresponding operation in the OSC by machine index. Similarly, for the WAC each position encodes which worker is assigned to the corresponding machine and worker in the OSC and MAC, by worker index. The encoding with indices is shown in Figure 3.2.

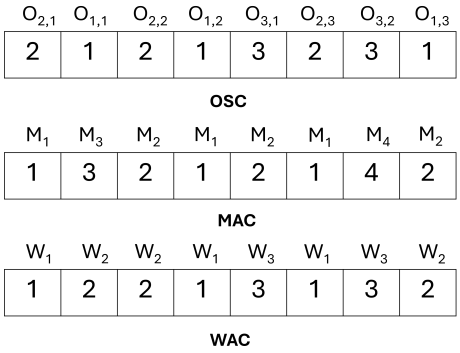


Figure 3.2: The encoding of the OSC, each gene representing operation by job number and operation is represented by the occurrence of the job number in the sequence, MAC genes are assigned machine indices and WAC genes are assigned worker indices.

Initialization

The population is initialized by seeding with precomputed heuristic schedules converted to chromosomes and initializing the remaining population randomly. The heuristic dispatching rules used are EDD, LPT, and SPT. The EDD heuristic is the benchmark used for evaluation, meaning that the GA contains the schedule it is supposed to perform better than. The heuristic algorithm is adapted such that it encodes the assignments into a chromosome that is run through the decoder to account for sequence-dependent transit times. The heuristic algorithms creates a schedule without considering changeover times and then the decoder adds them given the produced schedule, accepting additional changeover time. Thus, the heuristic rule ignores changeover times, since they depend on future assignment, when later encoded the sequence-dependent changeover times are accepted given the heuristic schedule. These heuristic rules add up to four seeds, using two different tie breakers in EDD, SPT, and LPT; consequently the proportion of randomly generated individuals increase with population size. The array is then shuffled to generate a random OSC chromosome. The MAC and WAC chromosomes are generated for each operation in the OSC, by randomly selecting an eligible machine and worker, respectively.

This seeding strategy is compared to a uniform seeding strategy, where each of the heuristic initialization methods and the random initialization strategy contributed to 20 % of the population. This gives the population the ability to reach more diversity across generations, although the initial population has multiple identical individuals from the same heuristic. The encoding and decoding ensure feasibility of every initialized individual.

Decoding

The decoding is performed by stepping through the positions of the OSC, MAC, and WAC; followed by assigning feasible start and finish time, from the processing times of each operation. Setup times for switching machine between jobs and switching

between job pools is accounted for by checking the next assignment of the selected worker and adding the corresponding changeover time. Meanwhile, tables keep track of when it is possible for the machine and worker to start a new operation. The decoding algorithm is described in Algorithm 2, a more detailed outline is given in the Appendix A.4.

Algorithm 2 Decoding algorithm

Input:
OSC,MAC,WAC
Processing times and setup parameters
Output:
Schedule S
Performance metrics
Initialization:
Initialize availability tables for jobs, machines, and workers.
Initialize previous assignments and operation counters.
Initialize performance metrics.
for each position $i \in$ OSC **do**
    Decode chromosome genes:
    Get assignments:
    $j \leftarrow$ job at position i
    $o \leftarrow$ next operation of j
    $m \leftarrow$ machine assignment from MAC
    $w \leftarrow$ worker assignment from WAC
    Evaluate feasibility and timing:
    Determine required resources and processing time
    Compute machine changeover setup
    Compute worker changeover setup
    start_time $\leftarrow$ max(job free time,
                machine free time,
                worker free time)
    finish_time $\leftarrow$ start_time + operation duration
    Update state:
    Add operation $(j, o, m, w, \text{start\_time}, \text{finish\_time})$ to schedule
    Update makespan
    if o is the final operation of j **then**
        Update total completion time metrics
        Update tardiness metrics
        Update lateness metrics
    end if
    Update tables
end for
return Schedule, Metrics

Selection

Tournament selection is used, choosing the best individual among s randomly selected individuals, where s is the tournament size. The tournament is deterministic i.e. $p_{tour} = 1$, meaning that the best individual is always selected among the tournament candidates. The tournament size s, is investigated in the parameter search.

Crossover

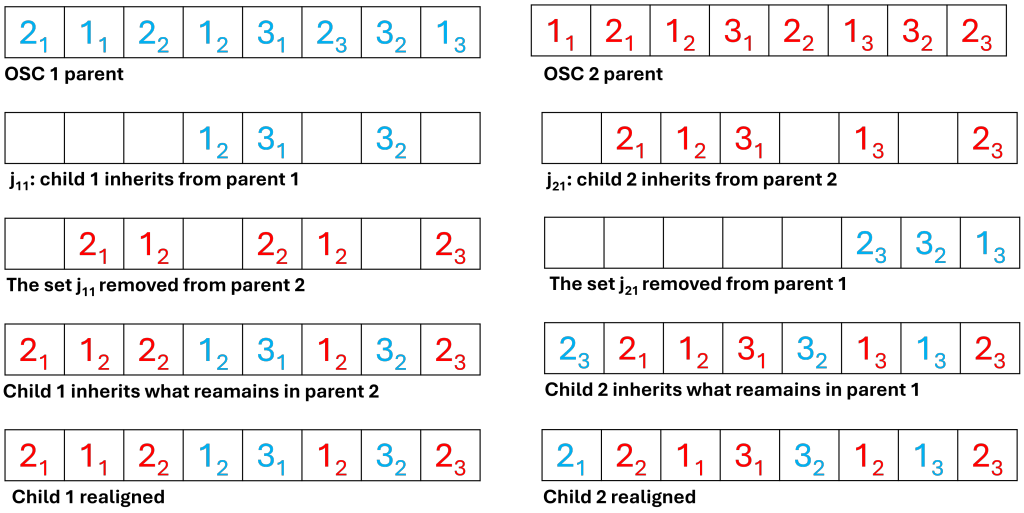
The crossover is performed by splitting the individuals into OSC, MAC, and WAC, then a crossover operation is performed for each respective part. The crossover operation is performed with probability c_p.

OSC Crossover

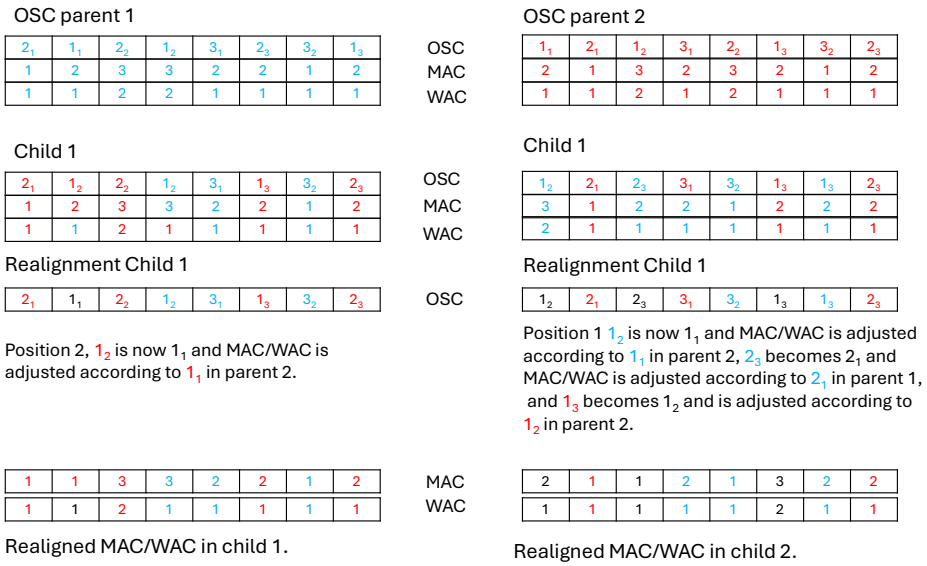
The two parents are randomly divided into two sets, j_{11} and j_{12} for parent 1 and j_{21} and j_{22} for parent 2. This is done by shuffling all indices of the OSC, and selecting a random point that determines which operations go into which set. Offspring 1 inherits a subset j_{11} from parent 1 and offspring 2 inherits a subset j_{21} from parent 2, the position is kept the same from parent to offspring. The sets j_{12} and j_{22} are not inherited by any offspring. Then the genes that exist in both parent 2 and offspring 1 are eliminated in parent 2 from left to right disregarding occurrence number, correspondingly for parent 1 and offspring 2. Then offspring 1 fills the remaining genes in parent 2, preserving the original order and filling the missing positions from left to right, and the same for offspring 2 and parent 1. Note that the occurrence number is not the same in the offspring and the parents, as shown in figure 3.3a. Then MAC and WAC is adjusted such that they align with their corresponding OSC genes, preserving feasibility, the process is shown in 3.3b. The realignment is done by tracking the occurrence number of the parent, if an operation is inherited such that it ends up before its preceding operation, these MAC and WAC genes are adjusted to their corresponding gene in the parent, preserving the fixed routing order.

MAC Crossover

For MAC the same multipoint crossover is used as in [51]. The operation is displayed in Figure 3.4. Parent 1 and Parent 2 are copied to offspring 1 and offspring 2, respectively. A random vector of zeros and ones is generated of the same length as the MAC. All positions in the random vector that equal 0 are interchanged between the offspring genes, WAC is changed accordingly, to preserve feasibility. However, since OSC is unchanged, each MAC and WAC gene is checked if it is feasible; if either MAC or WAC is unfeasible, they are randomly reassigned a new feasible assignment, thus repairing infeasible offspring. In contrast to [51], the shortest processing time is not sampled to repair infeasible assignments, since no assigned machines and workers differ in processing times for a given operation in the problem instances.



(a) The crossover operation for OSC.



(b) Realigning MAC and WAC to preserve feasibility.

Figure 3.3: Crossover for OSC, a toy example of three jobs, 3 machines is used. The operations are subscripted with their occurrence number for clarity.

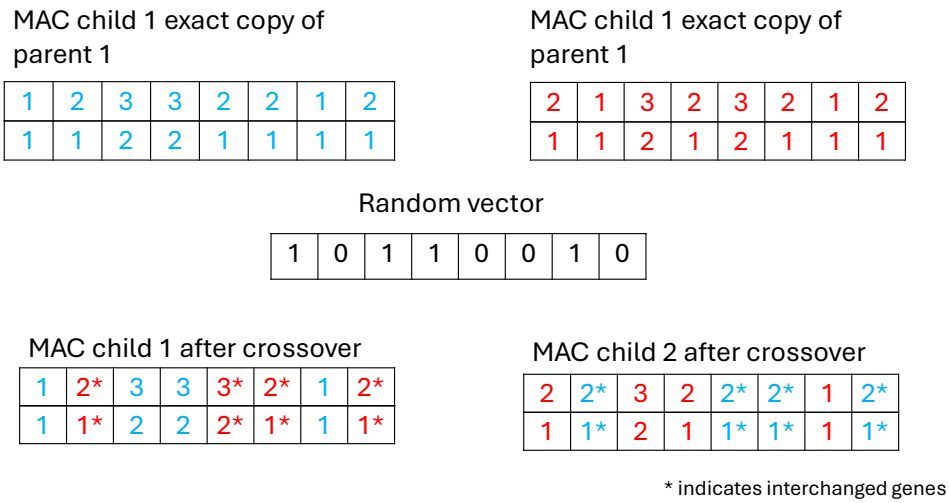


Figure 3.4: Schematic representation of the crossover for the MAC, using 3 machines and two workers.

WAC Crossover

The WAC crossover is performed by choosing two intersection points. The genes are interchanged at these points. If the swap results in an illegal assignment, a new feasible assignment is selected randomly, repairing the infeasible offspring.

Mutation

The probability for an individual to be selected for mutation is p_m. In contrast to [51], we also give each mutation operator a probability in order to investigate the effect of each operator on convergence. The respective mutations for OSC, MAC, and WAC then occur with a certain probability, but it is enforced that at least one will be mutated if an individual is selected for mutation.

OSC Mutation

Two genes in the OSC are selected and interchanged with probability p_{osc}. Then MAC and WAC are adjusted accordingly to preserve feasibility. If an individual is selected for mutation the OSC is mutated with probability p_{osc}.

MAC and WAC Mutation

A gene is randomly selected with probability p_{mac} and a new feasible assignment is sampled. The new assignment is not enforced to select a new machine, in contrast to [51]. Like the OSC mutation, the MAC mutation is selected with probability p_{mac}. The WAC is mutated in the same way as the MAC but occurs with probability p_{wac}.

Repairs and Realignment

Crossover and OSC mutation introduces swapping genes that may break feasible ordering of the chromosome. In order to preserve the feasibility of the chromosome, realignment is performed by the other chromosomes. The occurrence number is tracked from the parents and if the ordering is changed in the resulting offspring, the MAC and WAC assignments is adjusted such that it corresponds to the same as the parent.

MAC and WAC crossover can break feasibility by their design; therefore, infeasible assignments are repaired by resampling a new random feasible assignment. No heuristic rule for resampling machines is used, as in [51], since the operations have the same processing time on any machine. Since MAC and WAC crossover introduces repairs by design, the effect is investigated in the parameter search with those operators turned off.

3.4 GA Parameter Search Methodology

The parameter settings influence the performance of the GA. Therefore, a parameter search was conducted to see how population size, crossover probability, mutation

Instance	Number of jobs	Number of machines	Number of workers	Worker/machine ratio
A	37	22	18	82%
B	28	18	15	83%
C	37	14	17	121%
D	34	19	16	84%
E	48	21	17	81%
F	42	20	16	80%

Table 3.1: Instances used for parameter search, sampled from production data. Number of worker and machines reflect the total number of feasible machines and worker across all jobs in the instance.

probability, and tournament size affects convergence speed and best fitness. An initial search and refinement followed by a refined stage was planned. However, the results in the initial parameter search motivated a more extensive investigation of the repair operations effects on convergence, and whether the GA was primarily driven by mutation-based local improvements. The parameter search was conducted for both seeding strategies.

The search was conducted in stages, first an initial search was performed to see which parameters had the most effect on final fitness. Then interaction analysis were performed to analyze how the combination of parameters affect the final fitness. For searches varying multiple parameters, the Cartesian product of the chosen parameters was investigated, unless otherwise specified.

A budget of 10 000 evaluations of the objective function was set as termination criteria, making the number of generations comparable across runs independent of population size and number of generations. Under a fixed evaluation budget the number of generations is given by,

$$n_{gen} = \frac{B}{N},$$

where n_{gen} is the number of generations, B the evaluation budget, and N the population size. Consequently, smaller population sizes result in a larger number of generations under the same computational budget, potentially increasing the degree of iterative refinement. Since the objective is to construct a scheduler for a real production environment, production data was selected for tuning rather than benchmark data. The initial parameter search was run across six different medium size instances generated from company data and each instance was repeated across five different random seeds, using lateness as objective function. Lateness was chosen as the objective function since it aligns with the goals of the company, while continuing to reward schedules with improved flow time below zero tardiness. However, lateness allows the GA to prioritize early orders at the expense of more tardy jobs, experimental runs showed that the number of tardy jobs decreased with lateness and produced less flow time. The instances were randomly selected from the dataset and are summarized in Table 3.1.

To make the results comparable across instances, the fitness is presented as the

difference of the heuristic baseline's lateness and the GA's best lateness. To compare performance between instances, the median fitness of the different instances is used.

The configurations of the parameter searches are shown in Table 3.2.

Stage	**Fixed parameters**	**Varied parameters**	**Purpose**
Crossover search	$N = 50$, $s = 6$, $p_m = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.3$	$c_p = \{0, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9\}$	Identify effect of crossover.
Population search	$c_p = 0.2$, $s = 6$, $p_m = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.3$	$N = \{5, 10, 25, 50, 75, 100\}$	Identify effects of population size.
Pressure search	$N = 50$, $c_p = 0.2$, $p_m = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.3$	$s = \{2, 4, 8, 10, 12\}$	Identify effects of tournament size
Interaction analysis fixed heuristic seeding	$p_m = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.3$	$N = \{5, 10, 25\}$, $c_p = \{0.1, 0.2, 0.3\}$, $s = \{4, 6, 8\}$	Identify interaction effects of parameters
Interaction analysis with uniform heuristic seeding	$p_m = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.3$	$N = \{5, 10, 25\}$, $c_p = \{0.1, 0.2, 0.3\}$, $s = \{4, 6, 10\}$	Identify interaction effects of parameters
Mutation search	$N = 10$, $c_p = 0.2$, $s = 6$, $p_{osc} = p_{mac} = p_{wac} = 0.3$	$p_m = \{0, 0.15, 0.3, 0.5, 0.7, 0.8, 0.9, 1\}$	Identify effects of mutation probability
Mutation configuration search	$N = 10$, $c_p = 0.2$ $s = 6$,	Cfg: 1 (p_m 1.0, p_{osc} 0.4, p_{mac} 0.4, p_{wac} 0.3); 2 (1.0, 0.3, 0.2, 0.7); 3 (1.0, 0.6, 0.6, 0.6); 4 (0.8, 0.7, 0.2, 0.2); 5 (0.8, 0.3, 0.2, 0.7); 6 (0.8, 0.2, 0.2, 0.2); 7 (0.8, 0.6, 0.6, 0.6); 8 (0.6, 0.3, 0.6, 0.5); 9 (0.5, 0.3, 0.6, 0.5)	Test combination with OSC/MAC/WAC

Table 3.2: Configurations of parameter search progression. Parameter notation: N population size, s tournament size, c_p crossover probability, p_m mutation probability.

3.5 Experimental design

This thesis consists of three experiments. *Experiment I* is a benchmark study that evaluates solution quality and scalability across the five solution methods on a set of synthetic instances. *Experiment II* applies the best performing methods from Experiment I to production data from Thorlabs' ERP system with end-of-day due dates (see below). The methods are evaluated in terms of total tardiness. *Experiment III* repeats Experiment II but due dates are set throughout the day.

Experiment I - Benchmark

To evaluate the scalability of each method, a set of 100 synthetic instances are generated, ranging in size from 3 jobs, 3 machines and 3 workers up to 20 jobs, 15 machines, and 12 workers, with processing times sampled uniformly from the interval $[1, 99]$. The instances are divided into two sets, A and B, that differ primarily on the number of production pools per instance, with five instances generated per configuration. The full set of parameter configurations is shown in Table 3.3.

Set	Size	# Operations	# Machine options	# Worker options	# Pools
A	3×3×3	2	2	2	1
A	5×5×5	3	3	2	1
A	7×4×3	4	4	2	2
A	10×5×5	4	3	3	2
A	10×10×5	4	3	3	2
A	12×8×6	4	4	4	2
A	13×8×6	4	4	3	2
A	15×10×8	5	5	4	3
A	16×10×8	5	5	4	3
A	20×15×12	5	8	8	4
B	3×3×3	2	2	2	2
B	5×5×5	3	3	2	2
B	7×4×3	4	4	2	2
B	10×5×5	4	3	3	3
B	10×10×5	4	3	3	3
B	12×8×6	4	4	4	2
B	13×8×6	4	4	3	3
B	15×10×8	5	5	4	4
B	16×10×8	5	5	4	4
B	20×15×12	5	8	8	5

Table 3.3: Parameters used for synthetic instance generation.

The objective was set to minimize makespan. Each exact method is given a one-hour limit per instance to prove optimality. If optimality cannot be proven within the time limit, the best incumbent solution is recorded but excluded from optimality counts. The GA is given the same one-hour limit but also uses a convergence-based termination criterion. That being, if no improvement in objective function value is found within 500 consecutive generations, the algorithm terminates.

The results are reported as the mean and standard deviations in both makespan and time for each set of five instances with the same configuration. To reduce the impact of stochastic variations in the GA's objective value and solution time, each instance was run five times. For each instance, the mean and standard deviation were calculated across the five runs. Subsequently, the average of these means was computed for all instances sharing the same configuration. The heuristic approach follows the LPT dispatching rule and the same metrics as for the other methods are

reported.

Experiment II - Production Instances with End-of-Day Due Dates

Based on the results from Experiment I, only OR-Tools CP-SAT solver and the GA were selected for further evaluation, as they demonstrated the best combination of solution quality and scalability, which is shown in section 4.2. In addition, the heuristic dispatching approach was retained as a benchmark because it is representative of the scheduling procedure currently used at Thorlabs. Consequently, experiments II and III compare the performance of these three methods on production instances derived from Thorlabs' ERP system. The remaining methods were excluded due to inferior scalability.

In Experiment II, 21 production instances from Thorlabs' production data was created by drawing jobs and due dates from a two-day period from Thorlabs' ERP system. The size of each instance can be observed in Table 3.4.

Table 3.4: Dataset Instances Overview

Instance	Jobs	Machines	Workers
1	7	15	16
2	10	15	16
3	17	19	16
4	20	12	16
5	21	20	16
6	24	19	16
7	32	20	16
8	26	16	16
9	28	20	16
10	33	20	16
11	35	19	16
12	36	20	16
13	37	20	16
14	41	20	16
15	42	20	16
16	45	21	16
17	46	21	16
18	52	24	16
19	60	25	17
20	64	24	16
21	68	25	17

The objective is to minimize total tardiness. Due dates were extracted from the ERP system and converted into minutes relative to the start of the planning horizon, with all jobs due at the end of the respective day. Consequently, due dates were set to 480 minutes for jobs due on the first day and 960 minutes for jobs due on the second

day. It should be noted that the ERP data assume identical processing times across all eligible machines for a given operation.

Each method retains the stopping criteria defined in Experiment I. The reported performance measures are total tardiness and solution time. The GA is run five times per instance, and the reported objective value and runtime correspond to the average across these runs. This is done to reduce stochastic variations.

Experiment III - Production Instances with Intra-Day Due Dates

Experiment III uses the same production instances, solution methods, objective function, and evaluation procedure as Experiment II. The only difference is the treatment of due dates. Instead of assigning all jobs due dates corresponding to the end of the day, due dates are generated uniformly from the interval $[60, 960]$ minutes. This creates a distribution of due dates throughout the planning horizon and allows the methods to be evaluated under a wider range of delivery requirements.

All experiments were performed on an Intel Core Ultra 7 155H with 32 GB RAM running Windows 11. Solver versions used were Gurobi 13.0.1, Z3-4.16, and OR-Tools 9.15. Each solver was allocated a single thread to ensure a fair runtime comparison. Otherwise default settings.

4

Results

We present results of the parameter search of the GA and its implications on the GA's performance. The results from Experiment I are presented, where a comparison between Gurobi, Z3, Google OR-Tools, the GA, and the heuristic approach is made in terms of solution quality and computation time is made. Lastly, the results from Experiment II and III, where selected methods were applied to Thorlabs' production data presented.

4.1 Parameter search

The parameter search revealed properties of the GA that were unexpected, and analysis of the parameters effects of different GA implementation is presented.

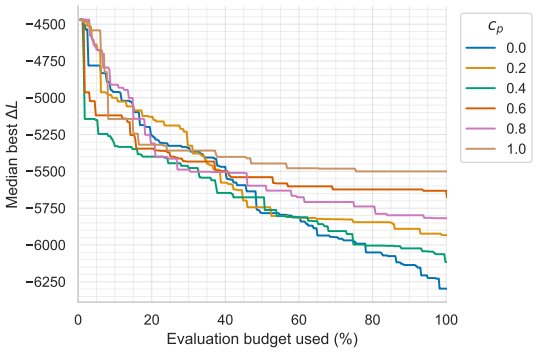
Parameter Search for First Seeding Strategy

Population size, crossover probability, and tournament size were varied separately, keeping the remaining parameters constant across runs for a fair comparison. Figure 4.1 displays the comparison between the active and deactivated MAC/WAC crossover operators.

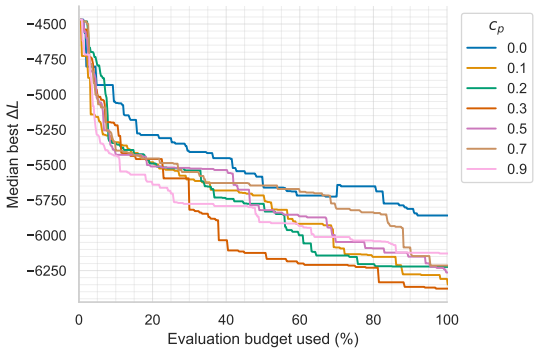
The crossover search shows that $c_p = 0$ reaches the best objective value after running 100% of the evaluation budget, as shown in Figure 4.1a with the original design of the GA. The GA without MAC/WAC crossover operators shown in Figure 4.1b, that $c_p = 0.3$ reaches the best objective value after 100% of the evaluation budget and also exhibits the fastest convergence at 40% of the evaluation budget. However, the objective value at 100% of the evaluation budget of Figure 4.1b is only marginally better than $c_p = 0$ in Figure 4.1a.

When varying the population size, figures 4.1c and 4.1d show that small populations are better for convergence. Population sizes in the range of $5 \leq N \leq 25$ reach the best objective values in both GA settings. Specifically, $N = 5$ converges to the best objective value. With MAC/WAC crossover deactivated we see in Figure 4.1d that $N = 5$ improves the objective value faster than other population sizes, compared with other population sizes at 40% of the evaluation budget.

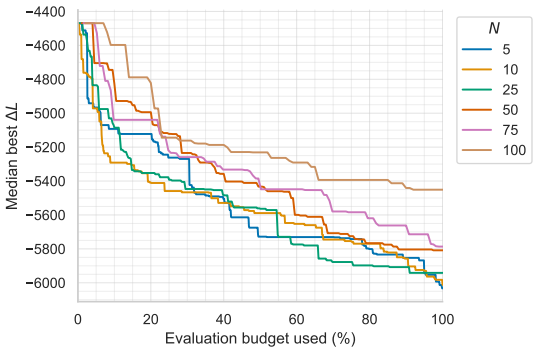
The search of the tournament size s is shown in Figure 4.1e, and Figure 4.1f. The results show no clear pattern regarding which value of s produces better convergence, for instance $s = 12$ reaches the best objective value in Figure 4.1e, but the difference from the final objective value of $s = 2$ is small.



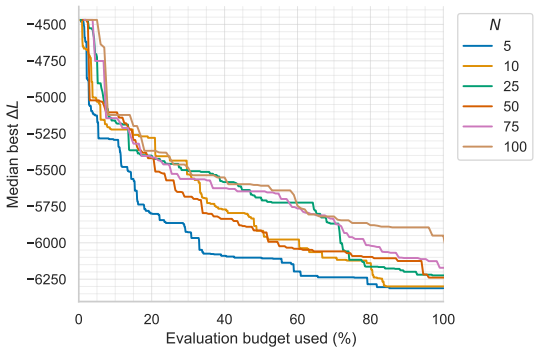
(a) Crossover probability search with all crossover operators.



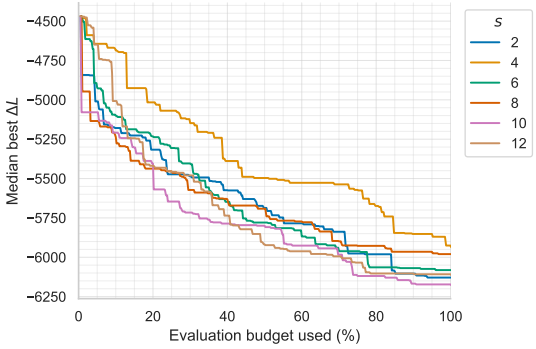
(b) Crossover probability search without MAC/WAC crossover operator.



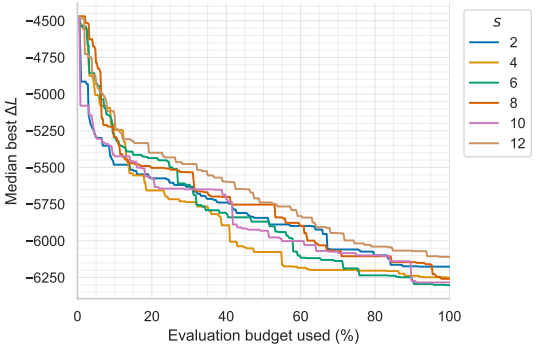
(c) Population size search with MAC and WAC crossover operators.



(d) Population size search without MAC and WAC crossover operators.



(e) Tournament size search with MAC and WAC crossover operators.



(f) Tournament size search without MAC and WAC crossover operators.

Figure 4.1: Parameter search of crossover, population and tournament size compared with and without MAC/WAC-crossover and initial seeding strategy.

Effect of Seeding Strategy

The parameter search was repeated for the uniformly seeded initialization. A comparison was made between configurations with and without MAC/WAC crossover

operators, and the results are shown in Figure 4.2. The results for crossover probability are compared between activated and deactivated MAC/WAC operators, Figure 4.2a shows that $c_p = 0.2$ converges to the best final fitness and in Figure 4.2b $c_p = 0.4$ reaches the best final fitness. At 40% of the evaluation budget, configurations without MAC/WAC crossover operators achieve better fitness overall in Figure 4.2. In contrast to Figure 4.1a, Figure 4.2a has the best achieving crossover probability of $c_p = 0.2$, and not $c_p = 0$. The best crossover probability is $c_p = 0.4$ without the MAC/WAC crossover operators. However, there is no substantial increase in fitness at 100% of the evaluation budget. In Figure 4.2d the convergence curve reaches a better fitness for particularly $N = 10$ compared with using only four heuristic seeds, in Figure 4.1d.

When varying the population size, figures 4.2c and 4.2d show that small populations achieve better fitness under fixed evaluation budget. Population sizes in the range of $5 \leq N \leq 25$ produce the best fitness, in the case of deactivated MAC/WAC crossover operators, $N = 10$ has a clear advantage both in convergence speed and final fitness as indicated by Figure 4.2d. Even with a uniform seeding it appears that the GA performs best with low crossover to solve the problem.

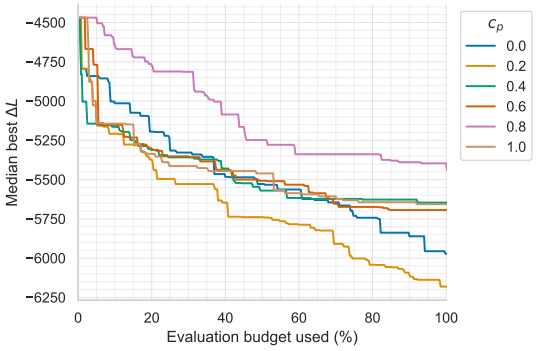
For tournament size, three clear candidates can be selected for best performance, $s = \{2, 4, 6\}$, as shown in Figure 4.2f.

Interaction Analysis

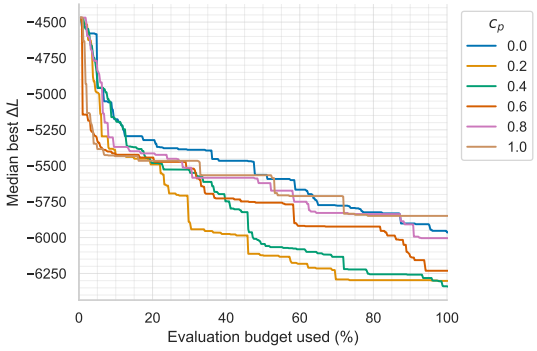
There is no clear evidence of improved final fitness except a slight advantage for seeded population size with $N = 10$, which converges the fastest, as seen in Figure 4.2d. Looking at 40% of the evaluation budget, the configuration achieves approximately 150 better final objective value compared to the previous seeding strategy. In the above parameter searches, the convergence curves were plotted keeping the remaining parameters constant across runs. The convergence curves indicate that $0.2 \leq c_p \leq 0.4$, $5 \leq N \leq 50$ gives the best convergence. An interaction analysis between these parameters is therefore conducted for both seeding methods.

Figure 4.3 shows the boxplots for the "one seed per heuristic" method initialization. In Figure 4.3a it is shown that $N = 10$ and $c_p = 0.2$ gives the best fitness with the lowest spread. The population and tournament size interaction in Figure 4.3b shows that the best fitness is reached by $N = 5$ and $s = 6$; however, this is at the cost of greater spread in the fitness. Although the fitness for $N = 10$ and $c_p = 0.2$ in Figure 4.4a achieved the best median objective value, the larger spread indicates lower robustness. Therefore, $s = 4$ is selected as the best candidate due to more stable convergence behavior. This choice is supported by Figure 4.3c, where interaction between crossover probability and tournament size is shown, $s = 4$ and $c_p = 0.2$ gives the best average fitness if we avoid $s = 8$ and $c_p = 0.1$ with wide spread with potential for worse solutions.

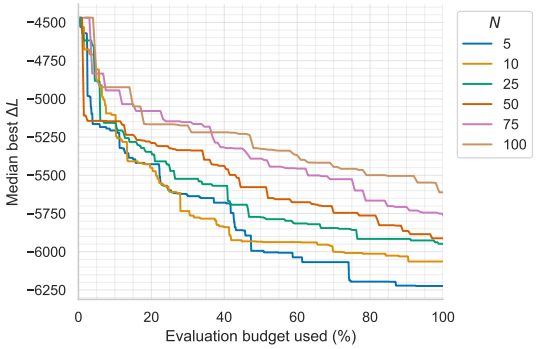
Figure 4.4 shows the boxplots for the uniformly seeded initialization. In Figure 4.4a



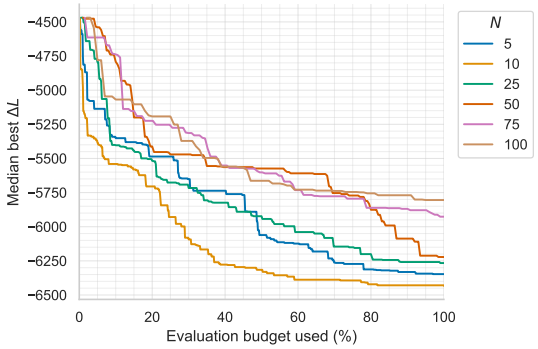
(a) Crossover search with MAC and WAC crossover operators uniformly seeded population.



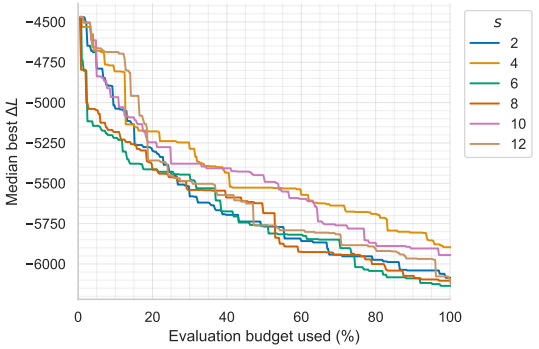
(b) Crossover probability search without MAC and WAC crossover operators and uniformly seeded population.



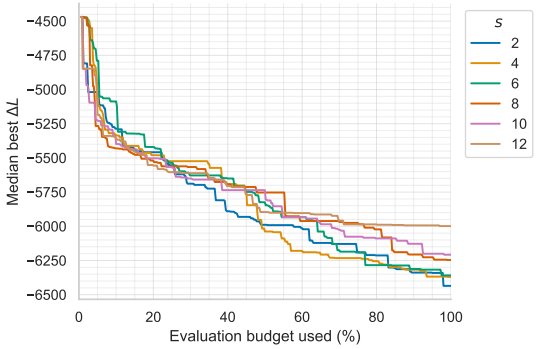
(c) Population size search with MAC and WAC crossover operators and uniformly seeded population.



(d) Population size search without MAC and WAC crossover operators and uniformly seeded population.



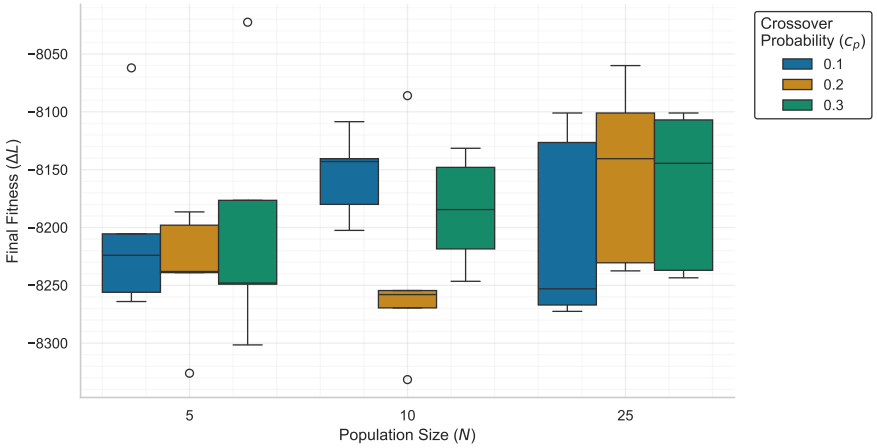
(e) Tournament size search with MAC and WAC crossover operators and uniformly seeded population.



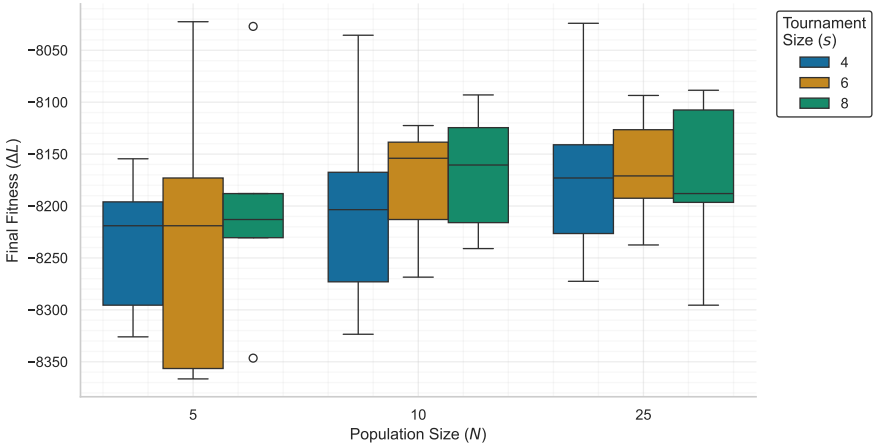
(f) Tournament size search without MAC and WAC crossover operators and uniformly seeded population.

Figure 4.2: Parameter search of population and crossover for a uniformly seeded population without MAC/WAC crossover operators.

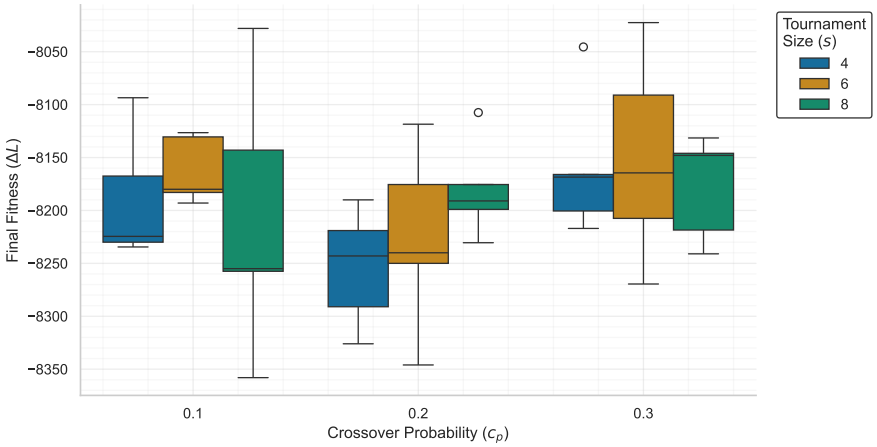
it is shown that $N = 5$ and $c_p = 0.2$ gives the best fitness with reasonable spread. The population and tournament size interaction in Figure 4.4b shows that reached by $N = 10$ and $s = 4$ results in the best median value of -8183. This choice is supported by Figure 4.4c, where interaction between crossover probability and tournament size is shown, $s = 4$ and $c_p = 0.2$ gives the best median fitness and the



(a) Boxplot of interaction between population and crossover probability.



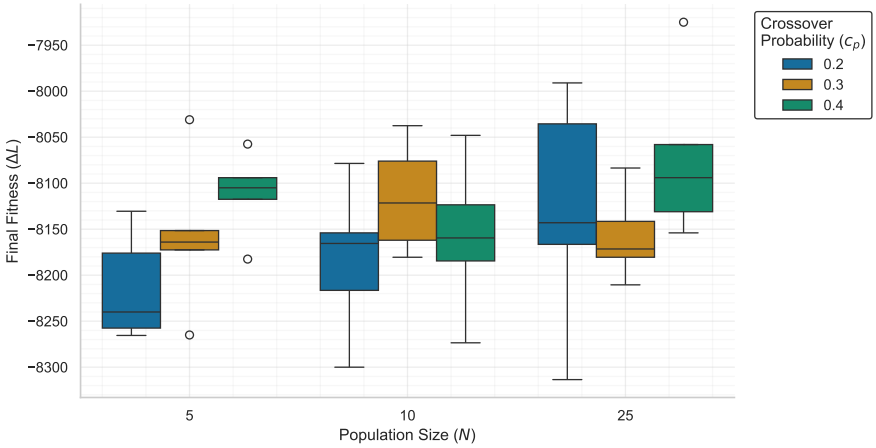
(b) Boxplot of interaction between population and tournament size.



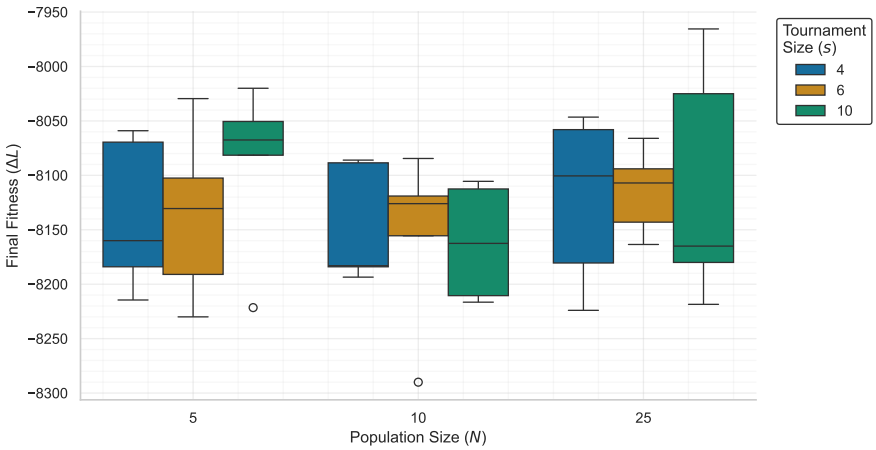
(c) Boxplot of interaction between crossover probability and tournament size.

Figure 4.3: Boxplots of final values of seeded population with one seed per heuristic seed, showing effects on final median lateness difference.

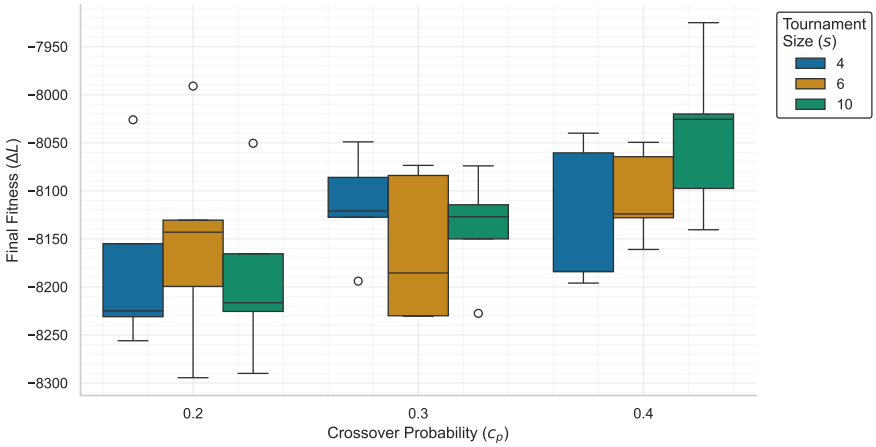
distribution is skewed to better objective values, giving better scheduling fitness.



(a) Boxplot of interaction between population and crossover probability.



(b) Boxplot of interaction between population and tournament size.

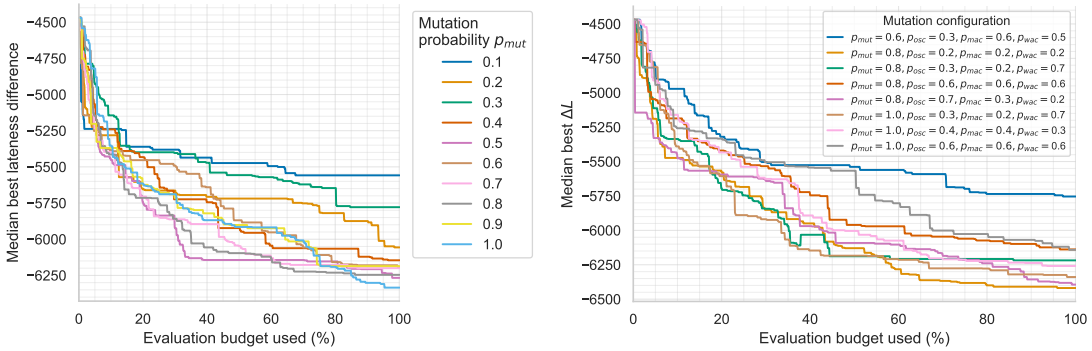


(c) Boxplot of interaction between crossover probability and tournament size.

Figure 4.4: Boxplots of final values of uniformly seeded population, showing effects on final median lateness difference.

Mutation Configuration

The parameter search for the mutation probabilities is run for the seeding strategy with one of heuristic schedule in the initialization and without MAC- and WAC-



(a) Parameter search for p_{mut}, keeping p_{osc}, p_{mac}, and p_{wac} constant.

(b) Refined mutation parameter search.

Figure 4.5: Mutation probability search for p_{mut}, p_{osc}, p_{mac}, and p_{wac}.

crossover, and is shown in Figure 4.5. The best mutation probabilities in Figure 4.5 are $p_{mut} = \{0.5, 0.8, 1\}$. With $p_{mut} = 1$ producing the best fitness at 100% of the evaluation budget. However, $p_{mut} = 0.5$ gives fastest convergence considering the value at 40% of the evaluation budget. From Figure 4.5b a refined search for p_{mut}, p_{osc}, p_{mac}, and p_{wac} gives the best configuration final fitness $p_{mut} = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.2$.

Final Parameter Selection

In order to determine whether one seed per heuristic or proportional to the population seeding strategy should be used, a comparison of the average fitness of the chosen parameters in each approach is considered. With $N = 10$, $c_p = 0.2$ and $s = 4$ for the GA with one seed per heuristic scheduling method the average fitness is in the range of $[-8270, -8220]$ and the spread reaches $[-8340, -8050]$. Likewise, for the uniformly seeded initialization with $N = 5$, $c_p = 0.2$ and $s = 4$, the average fitness is in the range of $[-8240, -8160]$ and the spread reaches $[-8260, -8060]$. The final fitness values differ only marginally between the two seeding strategies. However, the initialization method using only one seed from each heuristic schedule is selected due to slightly better objective values and stronger robustness. In particular, the selected configuration still contains a large proportion of randomly initialized individuals, suggesting that moderate random diversity contributes positively to convergence. For $N = 10$, $c_p = 0.2$, and $s = 4$, the proportion becomes 10% of each heuristic schedule and a total of 60% randomly initialized schedules, showing that introducing random diversity improves convergence. The mutation probabilities selected are chosen based on the final fitness, $p_{mut} = 0.8$, $p_{osc} = p_{mac} = p_{wac} = 0.2$.

ANOVA on Parameters

To investigate the relative importance of the GA's parameters an ANOVA (Analysis of Variance) was performed on the objective value at 40% of the evaluation budget. This value was chosen to evaluate how the selection of parameters affects the speed of convergence. The final median objective in the convergence curves showed little

variance. The parameters sum of squares, contribution to the variance, and p-value are displayed in Table 4.1. Most of the variance is explained by the instances (99.68%) and is statistically significant with a p-value = 0.00 (< 0.05). Further, the population size and MAC/WAC crossover were statistically significant ($p < 0.05$). Although crossover probability was statistically significant, its contribution to the explained variance is negligible.

	SS	Variance (%)	PR(>F)
Instance	1.21×10^{11}	99.68	**0.00**
Residual	3.30×10^{8}	0.27	
Population size (N)	3.05×10^{7}	0.03	**0.00**
MAC/WAC	2.41×10^{7}	0.02	**0.00**
Crossover probability (c_p)	8.70×10^{5}	0.00	**0.04**
MAC/WAC×c_p	5.49×10^{5}	0.00	0.15
Tournament size (s)	2.23×10^{5}	0.00	0.54
Seeding strategy	3.79×10^{4}	0.00	0.54

Table 4.1: ANOVA of population size, crossover probability, tournament size, instances, MAC/WAC crossover and seeding strategy, showing the sum of squares (SS), explained variance, and p-value.

Another ANOVA is performed for each instance, displayed in Table 4.2 and Table 4.3, showing variance importance and p-values, respectively. The residual explains most of the variance, meaning that a substantial proportion of the variance is not contributed to by the investigated parameters. Population size and MAC/WAC crossover were statistically significant across all instances explaining an average of 10.57% and 7.72% of the variance, respectively. Furthermore, crossover probability is statistically significant only for instance C and tournament size only for instance E. The seeding strategy was not statistically significant for any instance.

	Mean	A	B	C	D	E	F
Residual	79.65	83.63	74.35	92.26	85.47	52.99	89.21
Population size (N)	10.57	11.99	15.32	1.71	6.37	24.07	3.94
MAC/WAC	7.72	2.78	8.14	3.66	5.88	21.07	4.80
Crossover probability (c_p)	0.69	1.11	0.38	1.52	0.39	0.21	0.55
Tournament size (s)	0.71	0.30	0.33	0.48	0.58	1.39	1.16
MAC/WAC × c_p	0.59	0.12	1.27	0.35	1.26	0.28	0.26
Seeding strategy	0.07	0.07	0.22	0.03	0.05	0.00	0.08

Table 4.2: ANOVA per instance showing the variance importance of each parameter for each instance.

	A	B	C	D	E	F
Population size (N)	**0.00**	**0.00**	**0.01**	**0.00**	**0.00**	**0.00**
MAC/WAC crossover	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Crossover probability (c_p)	0.07	0.45	**0.04**	0.50	0.56	0.35
Tournament size (s)	0.60	0.51	0.44	0.31	**0.00**	0.08
MAC/WAC $\times$ c_p	0.86	**0.03**	0.58	0.05	0.43	0.67
Seeding strategy	0.52	0.22	0.66	0.59	0.97	0.50

Table 4.3: ANOVA per instance showing the p-values of each parameter for each instance, with statistically significant values marked in bold.

Another AVOVA was performed at 100% of the evaluation budget which showed that most of the effect of the parameters disappeared, with residual variance accounting for 97.32% of the total variance. This suggests that the investigated parameters primarily affects convergence and not final fitness.

4.2 Experiment I - Benchmarks

This section presents the results from 100 synthetic benchmark instances. All benchmark instances were run using Z3-4.16, Gurobi 13.0.1, and OR-Tools 9.15. To provide a sense of typical performance Table 4.4 shows the average makespan ($\overline{MS}$) and its standard deviation from each for each group of five instances as well as the number of instances solved to optimality. Table 4.5 shows the average solutions time each instances spent per instance configuration.

		Heuristic ()		OR-Tools			Gurobi			Z3			GA		
	Instance	$\overline{MS}$	σ_{MS}	*Opt*	$\overline{MS}$	σ_{MS}	*Opt*	$\overline{MS}$	σ_{MS}	*Opt*	$\overline{MS}$	σ_{MS}	*Sol*	$\overline{MS}$	σ_{MS}
A	3×3×3	203.4	45.1	5/5	155.0	11.8	5/5	155.0	11.8	5/5	155.0	11.8	5/5	159.0	12.2
A	5×5×5	419.8	44.2	5/5	225.8	22.7	5/5	225.8	22.7	5/5	225.8	22.7	5/5	233.8	15.8
A	7×4×3	863.0	125.7	5/5	385.0	51.0	2/5	406.5	81.3	2/5	406.5	81.3	5/5	464.9	52.4
A	10×5×5	1091.4	93.3	5/5	373.8	41.2	1/5	332.0	0.0	3/5	348.0	16.0	5/5	521.6	40.9
A	12×8×6	906.4	115.3	*	-	-	*	-	-	*	-	-	5/5	522.2	10.4
A	13×8×6	945.8	91.1	3/5	335.0	26.5	*	-	-	*	-	-	5/5	506.6	42.3
A	10×10×5	666.0	42.0	5/5	346.6	27.0	1/5	335.0	0.0	3/5	327.3	7.1	5/5	474.4	32.7
A	15×10×8	1258.0	77.2	*	-	-	*	-	-	*	-	-	5/5	707.9	39.5
A	16×10×8	1406.6	87.7	*	-	-	*	-	-	*	-	-	5/5	761.8	19.6
A	20×15×12	1216.4	137.3	2/5	320.0	11.3	*	-	-	*	-	-	5/5	712.4	29.2
B	3×3×3	220.0	66.8	5/5	146.4	22.9	5/5	146.4	22.9	5/5	146.4	22.9	5/5	150.4	16.1
B	5×5×5	396.4	45.5	5/5	223.2	36.2	5/5	223.2	36.2	5/5	223.2	36.2	5/5	235.8	38.4
B	7×4×3	902.8	104.4	5/5	382.6	26.2	5/5	382.6	26.2	5/5	382.6	26.2	5/5	467.8	31.5
B	10×5×5	1000.6	54.4	4/5	349.0	24.3	*	-	-	1/5	314.0	0.0	5/5	488.2	35.3
B	12×8×6	841.6	71.9	2/5	320.0	21.2	1/5	335.0	0.0	1/5	335.0	0.0	5/5	511.0	41.2
B	13×8×6	970.0	47.1	0/5	-	-	*	-	-	*	-	-	5/5	593.1	32.8
B	10×10×5	782.4	109.8	4/5	386.2	37.1	*	-	-	3/5	377.3	39.9	5/5	538.6	62.1
B	15×10×8	1336.0	104.9	*	-	-	*	-	-	*	-	-	5/5	695.2	37.2
B	16×10×8	1375.4	108.7	*	-	-	*	-	-	*	-	-	5/5	765.0	56.5
B	20×15×12	1185.0	72.2	2/5	331.5	36.1	*	-	-	*	-	-	5/5	725.1	44.0

Table 4.4: Comparison of heuristic approach, Genetic algorithm, OR-Tools, Gurobi, and Z3. For each method, the mean makespan ($\overline{MS}$) and standard deviation of the makespan (σ_{MS}) is displayed. For the exact methods, the number of instances solved to optimality is also displayed. The "*" symbol means that all instances reached the one hour time limit without proving optimality for any instance.

		Heuristic		OR-Tools			Gurobi			Z3			GA	
	Instance	$\bar{t}$	σ_t	*Opt*	$\bar{t}$	σ_t	*Opt*	$\bar{t}$	σ_t	*Opt*	$\bar{t}$	σ_t	*Sol*	$\bar{t}$	σ_t
A	3×3×3	0.00020	0.00003	5/5	0.01	0.00	5/5	0.05	0.04	5/5	0.02	0.00	5/5	2.89	0.38
A	5×5×5	0.00033	0.00006	5/5	0.04	0.01	5/5	0.75	1.04	5/5	0.18	0.06	5/5	3.65	0.37
A	7×4×3	0.00042	0.00002	5/5	229.92	267.83	2/5	1088.64	400.64	2/5	2718.05	69.94	5/5	9.40	1.77
A	10×5×5	0.00049	0.00001	5/5	283.66	393.05	1/5	1332.66	0.00	3/5	1156.93	1352.43	5/5	12.13	2.26
A	12×8×6	0.00060	0.00012	*	-	-	*	-	-	*	-	-	5/5	16.85	2.95
A	13×8×6	0.00069	0.00005	3/5	749.19	1007.62	*	-	-	*	-	-	5/5	17.67	4.24
A	10×10×5	0.00052	0.00002	5/5	135.85	173.96	1/5	128.22	0.00	3/5	437.58	360.63	5/5	12.53	2.39
A	15×10×8	0.00092	0.00009	*	-	-	*	-	-	*	-	-	5/5	33.13	6.51
A	16×10×8	0.00090	0.00006	*	-	-	*	-	-	*	-	-	5/5	36.04	6.93
A	20×15×12	0.00108	0.00002	2/5	1255.81	514.12	*	-	-	*	-	-	5/5	45.54	12.70
B	3×3×3	0.00019	0.00002	5/5	0.00	0.00	5/5	0.09	0.13	5/5	0.02	0.00	5/5	2.81	0.27
B	5×5×5	0.00030	0.00002	5/5	0.04	0.01	5/5	0.30	0.41	5/5	0.19	0.09	5/5	4.29	0.73
B	7×4×3	0.00039	0.00001	5/5	12.11	13.83	5/5	487.95	889.91	5/5	375.85	389.41	5/5	10.13	2.21
B	10×5×5	0.00048	0.00006	4/5	312.16	341.64	*	-	-	1/5	1385.84	0.00	5/5	13.95	4.40
B	12×8×6	0.00059	0.00005	2/5	814.44	1124.24	1/5	668.54	0.00	1/5	3377.32	0.00	5/5	14.33	3.44
B	13×8×6	0.00066	0.00006	0/5	-	-	*	-	-	*	-	-	5/5	16.86	3.18
B	10×10×5	0.00053	0.00005	4/5	89.93	74.85	*	-	-	3/5	1500.55	957.06	5/5	11.03	1.88
B	15×10×8	0.00086	0.00007	*	-	-	*	-	-	*	-	-	5/5	31.72	6.65
B	16×10×8	0.00091	0.00005	*	-	-	*	-	-	*	-	-	5/5	34.41	3.82
B	20×15×12	0.00117	0.00008	2/5	1568.91	854.39	*	-	-	*	-	-	5/5	45.68	3.48

Table 4.5: Comparison of heuristic approach, Genetic algorithm, OR-Tools, Gurobi, and Z3. For each method, the mean time ($\bar{t}$) and standard deviation of the time (σ_t)is displayed. For the exact methods, the number of instances solved to optimality is also displayed. The "*" symbol means that all instances reached the one hour time limit without proving optimality for any instance.

Figure 4.6 shows the number of instances that each exact method could solve optimally against solution time. For the GA is shown the mean makespan of five runs per instance versus time. It can be seen in Figure 4.6 that Gurobi proved optimality in only 30 of the 100 instances, while Z3 did so in 37 instances, and OR-Tools in 57 while the GA solves all 100 instances suboptimally.

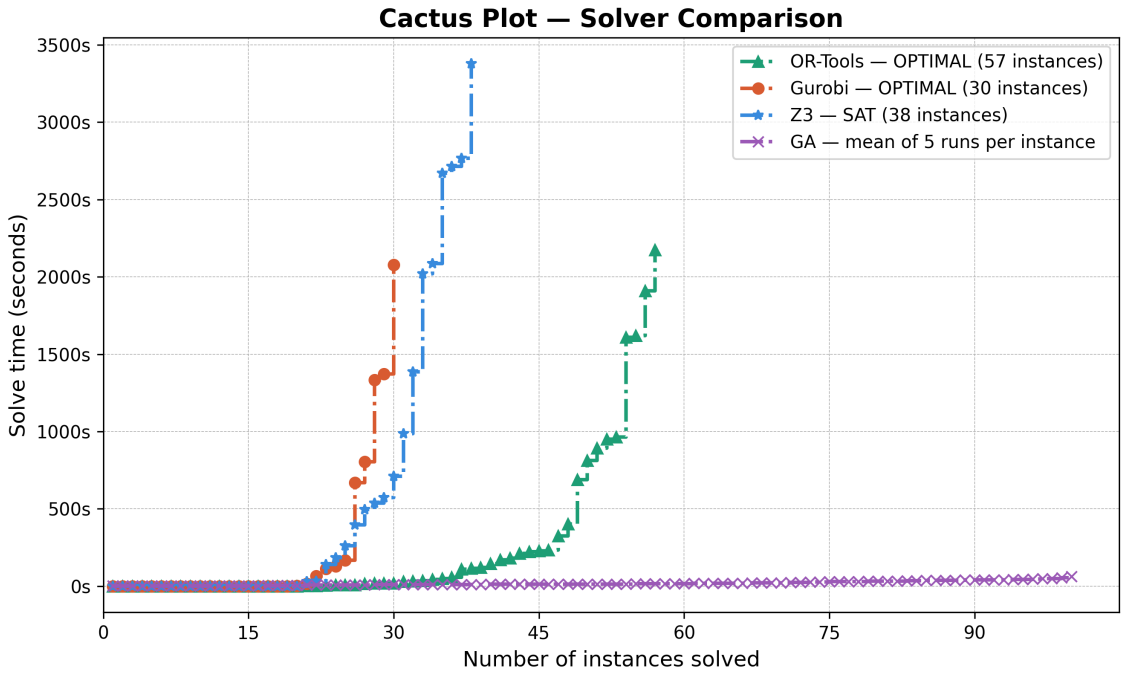
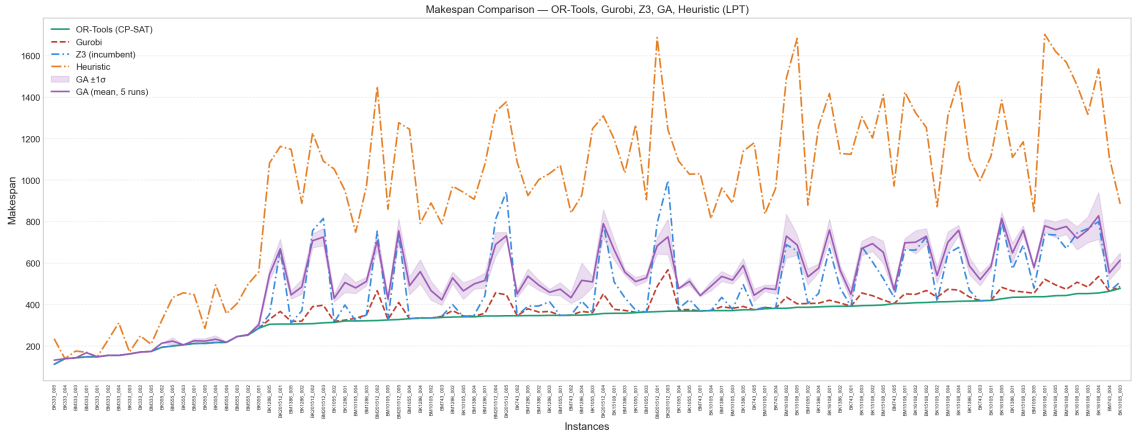
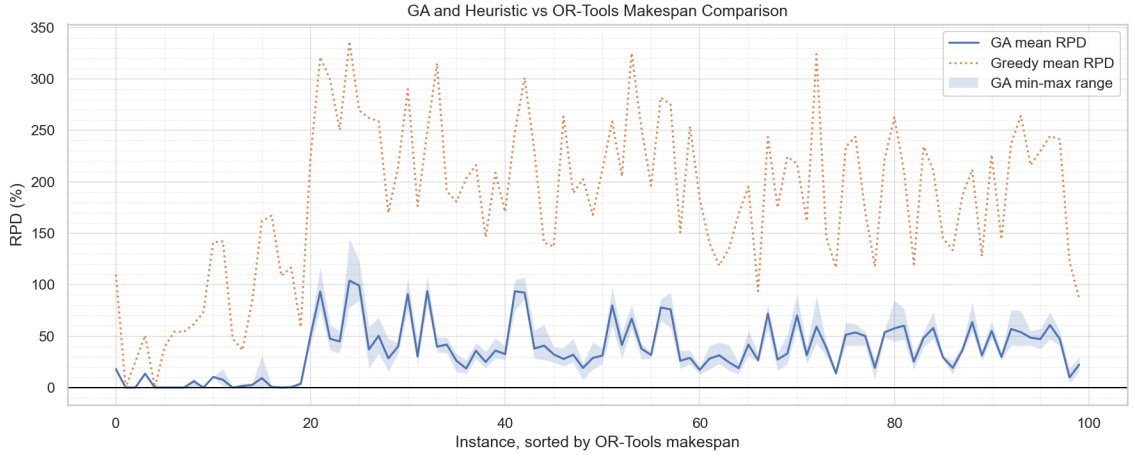


Figure 4.6: Comparison of solved instances for exact solvers and genetic algorithm. For exact solvers, only instances with proved optimality are shown. For the GA, the mean of five runs per instance are shown.

A comparison in makespan (minutes) between the GA, OR-Tools, Z3, Gurobi, and the heuristic approach mean can is shown in Figure 4.7a. The best incumbent solutions are considered for OR-Tools. Figure 4.7 shows the relative percentage difference for the GA and the greedy approach when using OR-Tools as a baseline. One can observe that greedy approach performs by far the worst, generally achieving results of 100% or more. The GA performs better than the greedy approach with many results below 50%.



(a) Cost function plot of heuristic algorithm and GA vs OR-TOOLS, with instances sorted in increasing makespan.



(b) *Relative percentage difference* of the greedy approach and the GA using OR-Tools as baseline.

Figure 4.7: Comparison of solution quality of OR-Tools and the GA.

Figures 4.8 and 4.9 show the solution quality per unit time in logarithmic scale for each set of instances with the same configurations in set A and B respectively.

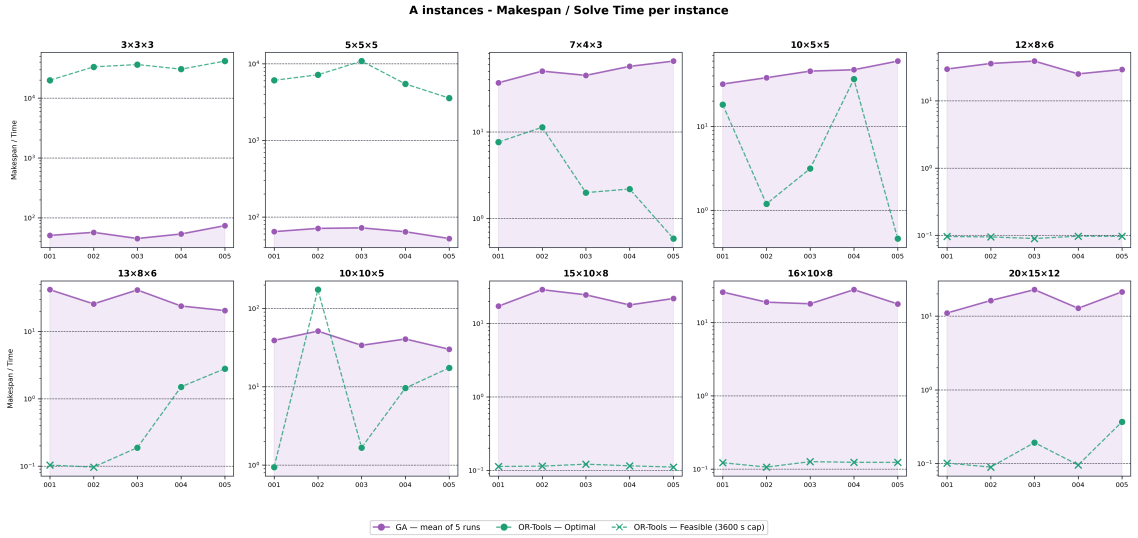


Figure 4.8: Solution quality per unit time per instance in logarithmic scale for each set of instances that share configuration in set A.

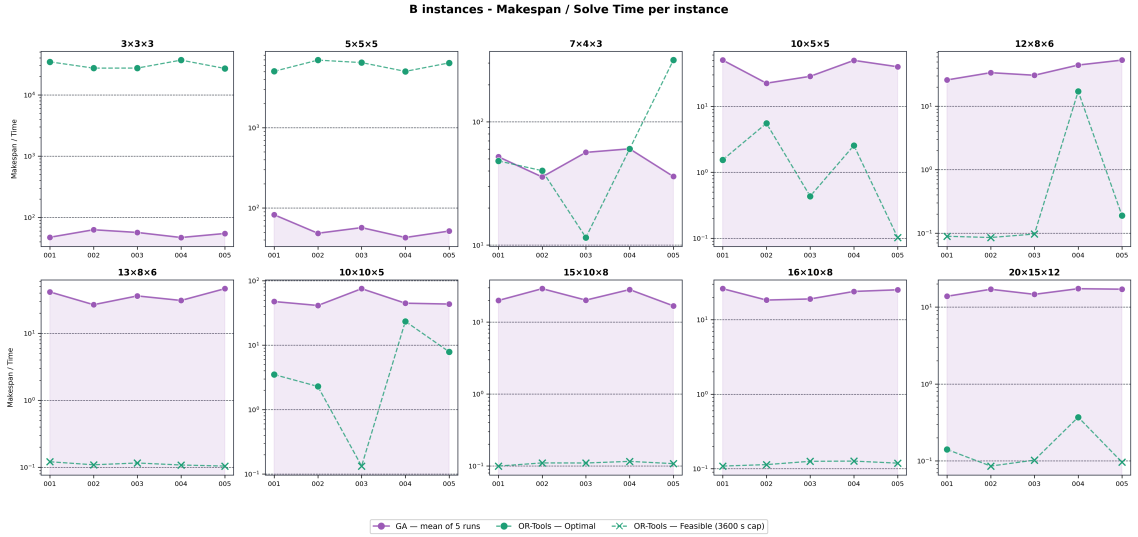


Figure 4.9: Solution quality per unit time per instance in logarithmic scale for each set of instances that share configuration in set B.

4.3 Experiment II - Production instances

The results from the 21 production instances with end-of-day due dates are presented in Table 4.6 and Figure 4.10. Table 4.6 reports the instance sizes, average runtime, average total tardiness, standard deviation of total tardiness, and average number of tardy jobs obtained by the GA. For the CP-SAT implementation, runtime, total tardiness, and number of tardy jobs are reported. Figure 4.10 compares the total tardiness achieved by both methods across all production instances. The shaded region represents one standard deviation around the GA's average total tardiness.

As shown in Table 4.6 and Figure 4.10, the GA and heuristic perform similarly for instances containing up to 42 jobs. For larger instances, the heuristic falls behind the GA. Furthermore, OR-Tools proves optimality in only 4 of the 21 instances and produces a higher total tardiness than the heuristic in the majority of cases, and than the GA in every case. Where after that the heuristic falls behind. OR-Tools only manages to prove optimality in 4 of the 21 instances and produces a worse total tardiness than the heuristic in a majority of instances and worse than the GA in all instances.

Table 4.6: Comparison of GA, Heuristic, and OR-Tools on production instances. Solve times in seconds; $*$ indicates timeout ($\geq 3600\,\text{s}$). GA results are averaged over 5 independent runs.

Size (j, m, w)	GA				Greedy			OR-Tools			
	Avg. Time (s)	Avg. Tard.	Std. Tard.	Avg. Tardy	Time (s)	Tard.	Tardy	Time (s)	Tard.	Tardy	Opt?
(7, 15, 16)	5.2919	0.00	0.00	0.00	0.0013	0	0	0.2184	0	0	✓
(10, 15, 16)	8.4052	95.00	0.00	1.00	0.0012	95	1	0.5223	95	1	✓
(17, 19, 16)	9.7043	0.00	0.00	0.00	0.0015	0	0	2.1003	0	0	✓
(20, 12, 16)	11.0671	0.00	0.00	0.00	0.0018	0	0	$*$	46	3	–
(21, 20, 16)	11.2235	0.00	0.00	0.00	0.0021	0	0	7.6400	0	0	✓
(24, 19, 16)	13.2678	0.00	0.00	0.00	0.0067	0	0	$*$	4	2	–
(26, 16, 16)	14.7356	0.00	0.00	0.00	0.0033	0	0	$*$	83	5	–
(28, 20, 16)	15.7032	0.00	0.00	0.00	0.0033	0	0	$*$	169	7	–
(32, 20, 16)	18.1702	0.00	0.00	0.00	0.0052	0	0	$*$	243	8	–
(33, 20, 16)	17.8586	0.00	0.00	0.00	0.0029	0	0	$*$	267	10	–
(35, 19, 16)	18.7765	0.00	0.00	0.00	0.0040	0	0	$*$	536	12	–
(36, 20, 16)	19.1764	95.00	0.00	1.00	0.0055	95	1	$*$	568	11	–
(37, 20, 16)	20.1297	95.00	0.00	1.00	0.0038	95	1	$*$	539	13	–
(41, 20, 16)	34.4707	95.00	0.00	1.00	0.0038	147	3	$*$	809	14	–
(42, 20, 16)	28.9024	95.40	0.89	1.20	0.0101	147	3	$*$	826	16	–
(45, 21, 16)	77.2822	174.60	131.87	1.80	0.0036	4235	15	$*$	739	13	–
(46, 21, 16)	38.9170	121.20	15.01	2.20	0.0047	167	3	$*$	746	15	–
(52, 24, 16)	32.0382	126.20	10.62	2.20	0.0043	167	3	$*$	789	15	–
(60, 25, 17)	110.2816	265.20	144.55	3.00	0.0043	5438	18	$*$	983	18	–
(64, 24, 16)	137.5385	441.40	159.84	5.40	0.0064	1733	13	$*$	1136	18	–
(68, 25, 17)	143.9477	616.20	244.36	6.00	0.0095	1528	13	$*$	1136	19	–

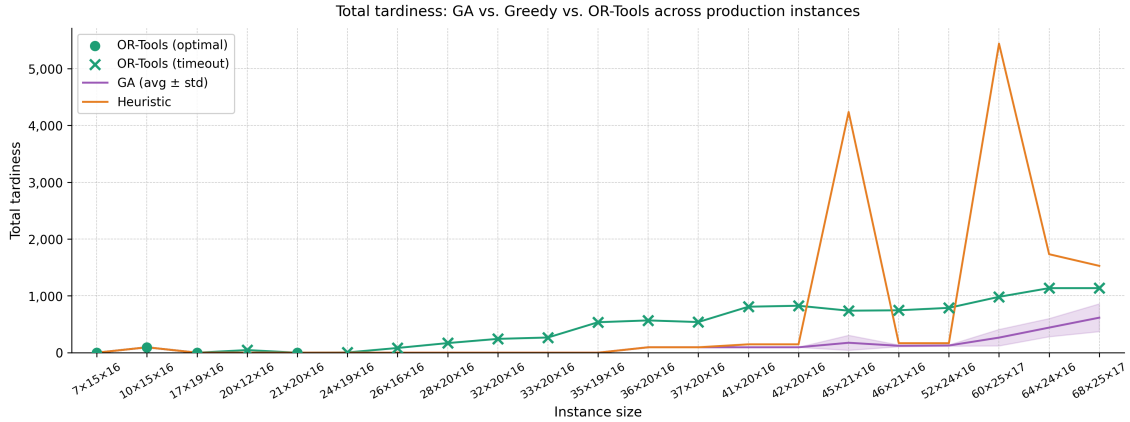


Figure 4.10: GA, OR-Tools, and heuristic across production instances using ERP due dates. Shaded area is ± the standard deviation of the GA for that instance. Green cross indicates solver timeout (≥ 3600).

4.4 Experiment III - Production instances with intra-day due dates

The results from experiment III can be viewed in Table 4.7 and in Figure 4.11. It can be observed that when using intra-day due dates, OR-Tools manages to prove optimality in 19 out of 21 instances. The GA performs similarly to OR-Tools for instances up to 42 jobs while the heuristic performs worse overall and only matching OR-Tools and the GA in objective value in three instances.

Table 4.7: Comparison of GA, Greedy, and OR-Tools on production instances. Solve times in seconds; $*$ indicates timeout (≥ 3600 s). GA results are averaged over 5 independent runs. Due dates drawn uniformly from the interval [60,960]

Size (j, m, k)	GA				Greedy			OR-Tools			
	Avg. Time (s)	Avg. Tard.	Std. Tard.	Avg. Tardy	Time (s)	Tard.	Tardy	Time (s)	Tard.	Tardy	Opt?
(7, 15, 16)	7.1341	65.00	0.00	2.00	0.0013	65	2	0.2425	65	2	✓
(10, 15, 16)	9.1882	328.00	0.00	2.00	0.0025	328	2	0.5044	328	2	✓
(17, 19, 16)	19.4070	14.00	0.00	2.00	0.0020	673	3	1.8292	2	1	✓
(20, 12, 16)	19.5570	3.60	3.58	1.00	0.0017	895	5	2.9597	0	0	✓
(21, 20, 16)	21.1956	186.20	20.57	2.00	0.0024	410	2	4.8766	162	2	✓
(24, 19, 16)	29.1038	24.00	0.00	1.00	0.0026	818	5	6.4762	15	1	✓
(26, 16, 16)	38.4272	458.60	62.19	3.80	0.0021	1488	6	13.7552	94	4	✓
(28, 20, 16)	27.6246	47.00	0.00	2.00	0.0027	890	4	11.4291	23	2	✓
(32, 20, 16)	38.7257	2059.40	184.43	12.40	0.0028	2759	14	817.6678	760	12	✓
(33, 20, 16)	63.0144	373.40	107.10	7.60	0.0032	2090	9	22.3584	53	3	✓
(35, 19, 16)	52.3749	327.40	109.11	3.20	0.0056	2062	8	22.8005	0	0	✓
(36, 20, 16)	86.6287	472.00	240.50	4.40	0.0043	3737	14	24.2646	104	4	✓
(37, 20, 16)	95.4718	598.80	103.37	5.20	0.0034	2468	10	27.5622	192	2	✓
(41, 20, 16)	77.8039	676.20	108.37	3.80	0.0037	1809	10	26.6482	0	0	✓
(42, 20, 16)	136.1232	1625.60	348.50	14.20	0.0054	5199	14	79.2070	426	13	✓
(45, 21, 16)	171.0831	2111.20	300.73	15.00	0.0047	13746	21	43.6717	200	5	✓
(46, 21, 16)	209.9240	1976.00	601.58	12.00	0.0071	5461	15	37.2400	62	2	✓
(52, 24, 16)	155.9857	4419.40	347.01	16.60	0.0044	6682	16	79.4203	616	12	✓
(60, 25, 17)	237.1405	6302.60	813.41	20.20	0.0046	19743	29	$*$	1835	25	–
(64, 24, 16)	362.7314	9065.00	430.49	29.60	0.0055	17088	37	$*$	3883	28	–
(68, 25, 17)	606.9474	9305.00	372.47	28.00	0.0055	19084	38	3314.3721	2718	30	✓

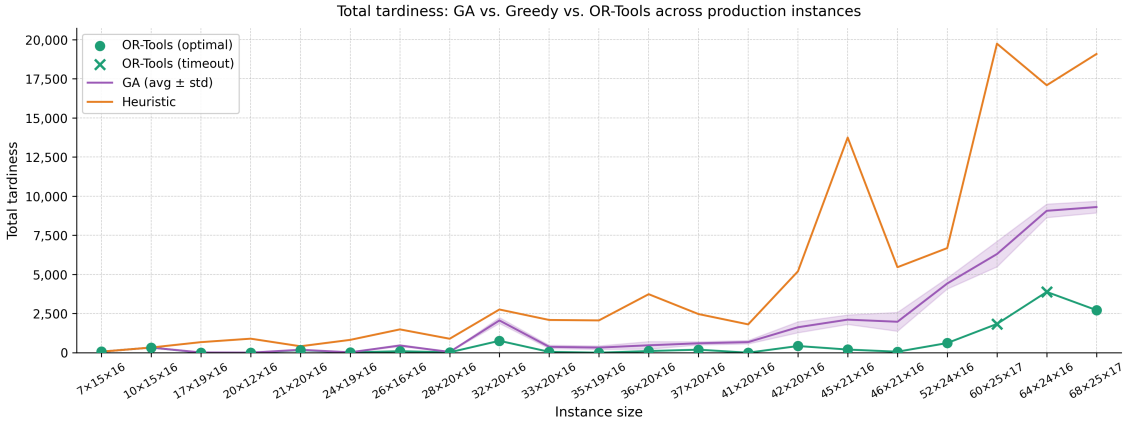


Figure 4.11: GA, OR-Tools, and heuristic across production instances using uniformly drawn due dates. Shaded area is ± the standard deviation of the GA for that instance. Green cross indicates solver timeout (≥ 3600).

5

Discussion

This chapter discusses the benchmark and production instance results in terms of solution quality, scalability, and practical applicability to the Thorlabs scheduling problem. It also addresses the modeling choices made in Chapter 3, the design decisions underlying each solution approach, and the relevance of scheduling optimization in the context of the company's lean manufacturing environment.

5.1 Experiment I - Benchmarks

This section discusses the benchmark results and compares the different solution approaches against one another, attempting to answer the research question 1.

The benchmark sizes were selected to span the transition from instances that exact solvers can solve to optimality within the one-hour time limit to instances that become intractable. This makes it possible to evaluate how the different methods scale as problem size increases. The instances were generated randomly to allow for controlled scaling across configurations. However, randomly generated instances tend not to produce bottlenecks that can occur in real production environments, potentially underestimating the practical difficulty of the DR-FJSP. Consequently, the benchmark results should be interpreted as measuring performance under controlled synthetic conditions, rather than as a direct representation of industrial scheduling performance.

It is important to note that the stopping conditions of the methods are not equivalent. The GA terminates when its convergence criterion is met, whereas the exact solvers either prove optimality or exhaust the time limit. As a result, the reported runtimes represent time-to-convergence for the GA and time-to-optimality (or timeout) for the exact methods. Although these quantities are fundamentally different, they reflect how the methods would typically be deployed in practice. Therefore, the runtime comparisons should be interpreted with this distinction in mind.

Tables 4.4 and 4.5 show the results from the 100 benchmark instances. For the small instances (up to $5 \times 5 \times 5$), all exact solvers reach the optimal solution with the GA close behind with near-optimal solutions. The heuristic produces makespans

roughly 1.3-1.9 times larger than the optimum. This gap only widens substantially as the problem size grows. This confirms that the heuristic approach, while fast, makes increasing sacrifices in solution quality as problem complexity increases.

Among the exact solvers, OR-Tools CP-SAT are the strongest performers, proving optimality on 57 of 100 instances. Z3 proves 38 and Gurobi 30, with both degrading sharply beyond sizes of $7 \times 4 \times 3$. Notably, OR-Tools manages to solve two instances at the largest scale ($20 \times 15 \times 12$), while Gurobi and Z3 exhaust the time limit on all five instances in that configuration. This advantage is attributable in large part to CP-SAT's global NoOverlap constraint, which applies dedicated propagation algorithms to prune infeasible regions early. MILP's standard big-M reformulation of the no-overlap constraint gives weak LP relaxations that give little guidance to the branch and bound algorithm. Z3's OMT approach similarly lacks the propagation machinery that makes CP-SAT effective on disjunctive scheduling constraints.

An interesting observation can be made between Z3 and Gurobi. Although Z3 proves optimality on more instances than Gurobi (38 compared to 30), Gurobi consistently produces equal or better solutions on instances where neither solver reaches optimality within the time limit as can be seen in 4.7a. This demonstrates that finding a high-quality feasible solutions and proving that it is optimal are distinct aspects of solver performance. Gurobi appears more effective at identifying good schedules quickly, whereas Z3 is more successful at reducing the search space sufficiently to prove optimality.

The GA produces feasible solutions on all 100 instances, but its mean makespan is higher than the exact solvers where comparison is possible. This can be seen in Figure 4.7a and Figure 4.7b, showing that OR-Tools produces a lower makespan for almost all instances and that the RPD between the GA and OR-Tools typically lies between 20-80%. One reason for this was mentioned earlier. The OR-Tools CP-SAT NoOverlap constraint is a structural advantage since the DR-FJSP involves a lot of disjunctive scheduling constraints.

Another reason for this could be the heuristic seeding that the GA uses. Seeding the initial population with SPT, LPT, and EDD solutions provides the GA with a structured starting population, whereas exact solvers do not receive comparable initial solutions. While this improves convergence and helps the GA quickly identify feasible schedules, it may also bias the search toward regions of the solution space associated with these dispatching rules. Since SPT produces makespans roughly 1.3–1.9 times the optimum (see Table 4.4), the GA may converge prematurely to locally good solutions rather than exploring regions closer to the global optimum. No such bias exists for OR-Tools as it explores from constraint propagation, which is structure-driven rather than solution driven. Furthermore, it can also be observed in Figure 4.7a that the GA's standard deviation across five runs is rather narrow. This consistency is a meaningful property for deployment. A stochastic scheduler that produces similar results across repeated runs is more trustworthy for a production manager than one with high variance.

Another strength of the GA is shown in figures 4.8 and 4.9. These plots show makespan divided by solving time on a logarithmic scale. The GA is more efficient than OR-Tools for almost all instances with more than 7 jobs for both sets. It can be seen that the GA is in a stable and bounded range across all instances. The GA stays within an order of magnitude regardless of problem configuration; this reflects the convergence-based stopping criterion working consistently. For instances up to roughly 5×5×5, OR-Tools is much more efficient, and there is no reason to use the GA. Beyond approximately 10×5×5 to 12×8×6 the GA becomes the more reliable choice in terms of efficiency. This raises the general question of the trade-off between solution quality and computation time, which is not a straightforward question to answer. Which method to use would heavily depend on the problem at hand and the given assumptions. The answer will lie somewhere between the statements: If a high quality or even optimal schedule is needed and the computations are allowed to take a lot of time – then an exact method such as CP-SAT is the best. But if a suboptimal schedule is accepted and the time it takes to compute it is relatively low, then the GA would be the best choice.

For synthetic benchmark instances considered in this study, OR-Tools CP-SAT emerges as the strongest overall method, achieving the best combination of solution quality and scalability among the evaluated methods. However, this conclusion is conditioned on the benchmark design and does not necessarily generalize to a production environment.

5.2 Experiment II - Production Instances

Table 4.6 and Figure 4.10 present results across the 21 production instances. All three methods perform comparably up to 24 jobs, after which OR-Tools' performance declines noticeably, with proven optimality achieved in only 4 of 21 instances. More surprisingly, the heuristic (denoted greedy in the table) matches outperforms OR-Tools in 17 of 21 cases. The GA delivers the best overall performance, consistently achieving the lowest (average) total tardiness and producing zero-tardiness solutions on several instances.

These results are largely explained by the structure of the Thorlabs' production data. Each operation is assigned a uniform processing time across all eligible machines and workbenches. Each operation is also assigned a uniform worker processing time across all eligible workers. This effectively trivializes both machine and worker assignment decisions. Combined with due dates that fall at end-of-day and take only one of two possible values that causes many jobs to have identical deadlines, the problem reduces largely to sequencing, with a large portion of feasible solutions carrying identical or near-identical cost.

Under this structure, a greedy heuristic that prioritizes earliest due dates and assigns workers by flexibility performs well precisely because most assignment choices are interchangeable. However, two instances show sharp spikes in total tardiness, that

being instances (45,21,16) and (60,25,17) suggesting that specific job configurations expose the greedy rule's inability to resolve conflicts among jobs sharing a due date. OR-Tools faces the opposite problem: when many feasible solutions have near-identical cost, the constraint propagation has little to exploit, and the solver exhausts its time limit in an informationally sparse search space.

The GA performs the best across all instance sizes. Since machine and worker assignment are effectively inconsequential, the problem reduces largely to sequencing, a setting where GAs are particularly well suited. This explains its ability to produce low- or zero-tardiness solutions even as instance size grows.

5.3 Experiment III - Production instances with intra-day due dates.

Table 4.7 and Figure 4.11 presents results for the same 21 production instances under randomly drawn due dates (uniform over $[60, 960]$). The differences from Experiment II are clear; replacing the end-of-day due dates with uniformly distributed deadlines improves performance of OR-Tools significantly. Going from proving only 4 of 21 instances optimally in Experiment II, to now achieving optimal solutions in 19 of 21 instances with solving times below 15 minutes for all but the three largest instances. The explanation is the same in the argument for Experiment II. Uniformly distributed due dates provide enough guidance for OR-Tools to effectively prune the search space. Notably, the 68-job instance, which also timed out in Experiment II, is solved to optimality here in approximately 3314 seconds, further reinforcing that problem structure rather than size alone is the reason for OR-Tools timeouts in Experiment II.

The heuristic (denoted "Greedy" in the table) on the other hand declines in performance. In Experiment II it matches the GA on most small-to-medium instances precisely because identical due dates trivialized the selection of workers and machines. When most jobs share a deadline, an EDD priority rule has little to distinguish between them, and assignments are mostly interchangeable. Under random due dates, this disappears. One can see the heuristic consistently produces high tardiness across the entire instance range with reaching above 19000 s in tardiness on the largest instances, more than double the GA and an order of magnitude above OR-Tools.

The GA sees a decrease in performance. Tardiness is generally higher in Experiment III compared to Experiment II. Despite this, the GA consistently outperforms the heuristic, and its gap from OR-Tools remains moderate at small-to-medium scale before widening on the five largest instances, where it reaches 9305 and 9065 against OR-Tools' 2718 and 3883 on the instances $(68, 25, 17)$ and $(64, 24, 16)$, respectively. Run times also increase. The 68-job instances requires 606 seconds on average compared to 144 seconds in Experiment II. This confirms that the uniform due dates makes the search harder, even for population-based methods.

Experiments II and III together suggest that the end-of-day due dates structure in Experiment II suited the heuristic and GA very well. Randomizing due dates in Experiment III largely restored the expected behaviour. OR-Tools solves nearly all instances to optimality, the heuristic declines in performance substantially, and the GA sit reliably between the two.

5.4 Modeling and Improvements

Modeling Choice and Extensions

The scheduling model was designed to capture the most essential properties of the production environment. The model can account for machine dependent processing-times and changeover times that are dependent on job, machine, or production pool. Further, the model has dual resource constraints dependent on operation; an operation can be assigned both a machine and a worker simultaneously or only one of each resource depending on the task. Routes are flexible in the sense that multiple machines can be selected for an operation.

In order to refine the model, queue times need to be considered, for which both machine and worker become available for a new operation assignment, but the operation is not finished until the queue time has completed. An example is for different epoxy curing times that can proceed in a buffer zone rather than fixed in the machine or test station. Queue times are of particular significance since they delay the completion of an operation, and would be the next variable to add to the models.

Another constraint of interest is considering workers having different skill-levels for certain tasks. Worker proficiency differences cause processing times to be worker dependent, and incorporating worker skill levels into scheduling models has been shown to significantly improve production performance. Studies such as [2] demonstrate that considering worker proficiency in DR-FJSP models can reduce total production completion time and lead to more realistic scheduling solutions. By introducing worker proficiency, the combinatorial landscape becomes more complex and may require further development of the GA to improve performance. The data for worker proficiency can be accessed through the ERP, since start time and end times are monitored in real time. A full MES integration tracking every operation's real start and finish for every worker, would enable complete modeling of worker proficiency. For realistic modeling, this data needs to be analyzed in regard to reliability and variation in order to set appropriate worker proficiency level dependent on job to ensure the logged times reflect reality.

The assumption that all workers are available at time zero does not hold since the workers have flexible working hours. An approach in modeling workers available at their regular arrival time is suggested, giving the model a calendar function. With digitalized solutions, it would be possible for workers to report their expected arrival time the preceding day and integrating this information with the scheduling module would provide more realistic solutions.

Another addition is to add setup times, accounting for mounting and dismounting equipment in e.g. test stations. These setup times should also be modeled as sequence dependent, since identical jobs require the same equipment, but another job requires another testing equipment. These times are small compared to total processing time and the omission from the presented model is in favor of changeover times for cleaning times, queue time is considered to be of more importance for expanding the model.

Further, variations in worker attendance due to, e.g. sick leave and vacation, have not been investigated. It is hypothesised that an optimized worker assignment is more relevant when the workforce is reduced compared with a heuristic schedule. Only the qualifications of the worker were considered, since the main goal is to utilize the cross-trained workforce to prioritize the most urgent jobs during a work day. By reducing worker attendance, the worker-machine ratio is decreased, and the worker assignment may have more significant effect on schedule performance compared to a heuristic solution.

The model does not consider stochastic variations, and processing times can be dependent on material quality and results of tests, making it a relevant consideration. It is a part of the lean process to reduce defects in production as a part of the zero waste goal, which can increase the reliability of deterministic processing times. However, defects can get past quality control or be caused to sensitive materials and parts during transportation and handling. Failed testing of products can cause adjustments and reassembly, having the job recirculate to previous machines to readjust assembly or change a defective part. Also, modeling of worker proficiency may introduce stochastic variations. Workers have an average processing time estimating their proficiency, but it may vary across jobs and workday depending on their concentration, stress, and rest levels. Monitoring of real processing times also enables analysis of stochastic variations. However, the cause of the variation needs to be determined, it can be both worker dependent and defect dependent.

Orders are received continuously; the goal of JIT is to respond immediately. Integration with the ERP would enable the scheduler to generate an updated schedule if urgent orders arrive. Furthermore, production disruption caused by defects, machine, and tool breakdowns also calls for the need of rescheduling. Having real time information of the current state of the production environment would enable rescheduling to dynamic changes. However, this demands that schedules are robust since it may be impractical to make major changes in a daily schedule during a day.

Adding these additional variables and constraints would add realism to the model, providing more reliable schedules, insights, and predictions. However, expanding the model with additional constraints introduces more complexity, which may require higher efficiency of the algorithms. The scheduling algorithms need to be tested against refined models and possibly redesigned and tuned to solve these efficiently. A possibility is that the need for meta-heuristic approaches is required at even smaller instances.

The DR-FJSP formulation captures all possible routes read from the data. In addition, the scheduler is robust to changes in production setups and the introduction of new routes for new products. Production environments in the real world are often unpredictable and dynamic, which demands robust algorithms [53]. The DR-FJSP model is a general model that also captures the structure of local hybrid flow shops and their couplings. It allows for selection of multiple feasible machines per operation and can also account for different processing speeds on machines, which is relevant for production cells other than the one modeled in this project. The flexibility in the current formulation relates to the fact that multiple machines can be selected. However, the main flexibility that determines assignment has been the dual resource constraints, where workers have varying skills and should be assigned jobs where they are needed the most to meet the current demand with varying quantities of jobs that have to be completed in a day. The modeling allows for global optimization of the cross-trained worker assignment, which would not be possible by modeling production pools as independent flow shops and hybrid flow shops, neither would shared resources across the pools have been taken into account.

It has been shown in Experiment I (Section 4.2) that using a general model such as the DR-FJSP, although theoretically complex, instances can be solved within reasonable computation time by OR-Tools up to 20 jobs and the GA finds feasible robust solutions for all instances that have been tested. Real production environments do not usually exhibit worst case behaviour which allows the solvers to handle larger instances, as illustrated by Experiment III. The flexibility in FJSP, that conventionally focuses on machine selection, is shifted to flexibility of the dual resource. Taking into account worker flexibility and modeling the entire production cell as a DR-FJSP allows for optimization of both job sequencing and worker assignment, while machine capacity constraints are maintained.

The Thorlabs' model is generic enough that it may be extended to other production cells with more complex behavior and could potentially capture interactions between production cells. However, this has not been tested in this work. At Thorlabs, other production cells focus on engraving of material before it's released for assembly, testing and assembly of different products, quality control, packaging, and shipping. The different cells could be modeled separately, but since workers may have experience across cells, it is of interest to see if scheduling can be done including more cells. Scheduling all cells with global worker assignment would lead to larger instances with more jobs, machines and workers which increases the solution space.

The scheduler does not integrate directly with the ERP, which would be beneficial to get real time update of incoming orders. Further, product routes would be able to be updated immediately if changed or new products are launched into production. These are functionalities that lay the foundation of Industry 4.0 and smart manufacturing concepts, indicating that the information needed for a complete scheduling model can technically be established.

Dataset Structure

The FJSP problem is hard to analyze already for the case that does not have different processing times for machines. The model is formulated to deal with different machine speeds and have been tested in the benchmark dataset.

The complexity of the problem is dependent on the combination of jobs, machines, and workers in the problem instance, this is supported by the ANOVA of the parameters of the GA. The instances explained 99.68% of the variance of the convergence curves, indicating that the instances differ structurally, which can be explained by the orders being released on demand. Production data may contain both high variety of jobs and a few of the same job based on the current demand of the customer, the exploitation of bottlenecks may then vary depending on incoming orders.

The dataset of the problem does not necessarily capture the full complexity of the modeled DR-FJSP problem, since a structured real production environment does not provide the same combinatorial space as the theoretical possibility. Thus, when running on real world data the distribution of job variety and quantity is dependent on customer demand. This means that some pools may have higher workload than others and that bottlenecks are not exploited to the full extent.

To test the algorithms under less structured conditions, a benchmark data set was generated. This was done to test the generality of the algorithms, ensuring that they did not just solve the problem because of a structured production environment. The benchmark data was customized to include variations in the variables of the model, with different pools and processing times for machine and worker. The benchmark data was randomly generated and not designed with intentional bottlenecks. The algorithms should be tested on more challenging instances, introducing more bottlenecks and different processing times for machines, in order to determine their performance to more general problem instances designed to challenge the solvers. The standard FJSP benchmark datasets can be modified to contain workers and pools. Using the standard benchmarks as the baseline would introduce different levels of complexities to be solved, since algorithms in studies often are compared against these. Further, a comparison with the company's ERP's computed metrics and real world KPI would be interesting to evaluate the performance of the algorithms, which would indicate how much an optimized schedule actually can contribute to improved operation efficiency.

5.5 Algorithm Design and Solver Implementation

Heuristics

The heuristics used in this project were based on the simplest methods, SPT, LPT, and EDD. More elaborate heuristic algorithms can be constructed and compared against. We have therefore compared with a simple Heuristic benchmark, EDD first, motivated by the lean methodology of JIT. The precise dispatching rules of

the company's ERP are not known, since it is proprietary. An interesting analysis would be to compare with makespan, flow time and tardiness computed by the ERP.

Genetic Algorithm

The GA design and its parameter choice have been thoroughly investigated. The crossover designs reveal effects of repairing infeasible assignments, and alternative seeding strategies show effects in parameter choice. Choosing GA design and parameter configuration is based on best fitness reached under a fixed evaluation budget and the robustness of the solutions.

The scalability of the proposed GA remains a question for future investigation. Further experiments are required to determine how performance changes with instance size. If similar improvements can be maintained for larger planning time scale, the GA would become a useful decision support tool not only for operational scheduling but also for longer term capacity planning and workforce cross-training decisions.

Effect of MAC/WAC Repair

The parameter search showed that the configurations with MAC/WAC crossover consistently achieved poorer convergence than the configurations where these operators were disabled. This behavior was observed for both seeding strategies, and is supported by the ANOVA that showed that the MAC/WAC operators affected the fitness of all instances. In the seeding strategy with one seed per heuristic schedule, the best performance was obtained for $c_p = 0$, while deactivating the MAC/WAC crossover operators shifted the best crossover probability to $c_p = 0.2$. A similar pattern was observed for the uniformly seeded population where the best performance was obtained at $c_p = 0.4$ without MAC/WAC crossover and $c_p = 0.2$ when operators were active. These result indicate that crossover becomes more effective once the disruptive effects caused by the MAC and WAC crossover operators are removed.

The diagnostic logs provide an explanation for this behavior. A large number of repairs were triggered after both MAC and WAC crossover operations, indicating that the offspring frequently violated assignment feasibility constraints. Since infeasible assignments were repaired through resampling, inherited machine and worker assignments were often replaced after crossover. This suggests that the repair mechanisms disrupt inherited structural information and reduce the effectiveness of crossover recombination. The results therefore suggest that the repair process itself, rather than crossover in general, is responsible for the observed degradation in convergence.

Although repair operations reduced convergence speed, their impact on final fitness were comparatively small. At the end of the evaluation budget, the best solutions obtained with and without MAC/WAC crossover differed only marginally. This indicates that the GA is capable of recovering from disruptive crossover operations through repeated mutation and selection. However, the faster convergence observed without MAC/WAC crossover suggests that a substantial part of the search effort

is spent repairing infeasible offspring rather than exploring promising regions of the search space.

The parameter search also revealed that the GA remained effective even with very low crossover probabilities. Together with the strong performance of small population sizes, particularly $N = 5$, this suggests that the search process is largely driven by mutation based local improvements around heuristic solutions. The convergence curves rarely improve in the later stages of the evaluation budget, suggesting that the search becomes trapped in local optima, since improvements become rare in later stages of the evaluation budget, and the global optimum is not reached for larger instances. The results from experiments II and III support that the GA gets trapped in local minima, because OR-Tools proves optimality, where the GA terminates after 500 generations without improvements.

A more effective crossover operator could potentially improve exploration and allow the GA to escape these regions of the search space. Designing such crossover is challenging because feasibility must be preserved simultaneously for OSC, MAC and WAC. Potential alternatives to OSC crossover include job-based crossover operators and pool-based crossover operators. To keep feasibility for MAC, crossover could be performed on groups of machines with the same set of feasible jobs. Similarly for WAC crossover, where crossover could be performed on groups of workers that can perform the same set of jobs. These constraints of crossover can be precomputed from the data and then the entire chromosome undergoes crossover by recombining the feasible groups in sequence. The results obtained in this study suggest that feasibility-preserving crossover operators are likely to be more effective than repair-based approaches for this scheduling problem.

Seeding strategy

The results show that the choice of seeding strategy influences both convergence behavior and parameter sensitivity. Analyzing the convergence curves, the uniform seeding strategy achieved the best performing crossover probabilities and population sizes compared with the initialization strategy using one seed from each heuristic. In particular, the best population size increased to $N = 10$ and higher crossover probabilities became more favorable when MAC/WAC crossover operators were disabled. These observations suggest that increased population diversity improves the usefulness of crossover, providing a larger variety of solutions for recombination. However, the ANOVA showed that the seeding strategy was not statistically significant for any instance.

Despite these differences in convergence behavior, neither seeding strategy produced substantial improvement in final fitness, which is supported by the ANOVA results at 100% of the evaluation budget, showing that the residual explains 97.32% of the final fitness. The best solutions obtained from the convergence curves were similar; together with the ANOVA result this indicates that initialization diversity primarily affects the search dynamics rather than the quality of the final solution.

One possible explanation is that the heuristic schedules used for initialization are of similar quality and, therefore, provide limited additional information in the search process. An approach for future work would be to improve the effect of seeding by investigating stronger heuristic algorithms that generate higher quality initial solutions. One suggestion of heuristics would be to distribute worker across pools in proportion to their workload and then assign jobs according to EDD to each pool. The idea of seeding the initial population is that crossover can combine building blocks from different heuristic schedules; improving the quality and diversity of the seeded solutions may increase effectiveness of recombination and ultimately improve final solution quality.

Interaction Analysis

The interaction analysis reveals that some configuration causes a wider spread in fitness, increasing the potential to get worse schedules. The parameter choice is determined by trade-off in final fitness value under an evaluation budget and the spread across different seeds of the random number generator. The parameters chosen are based on the convergence curves and boxplots, displaying the median fitness between instances. The effect of population size, crossover probability and MAC/WAC crossover is evident, while tournament size and seeding strategy effects were subtle. This is explained by the ANOVA that showed statistical significance for population size and MAC/WAC crossover across all instances, while crossover probability and tournament size were only statistically significant for one instance.

Comparison with Previous Work

The parameter analysis revealed that the designed GA differs substantially from the GA proposed by [51]. Although [51] reported good performance using a population size of 160 with multiple heuristic initialization strategies and higher crossover probability, the GA developed in this project achieved its best performance with small population sizes and small crossover probability. In particular, the best performing configurations were obtained for $0 \leq c_p \leq 0.4$, while [51] reported crossover probabilities of 0.9 and 0.5, for the GA and GA-VNS, respectively. Improvements become rare in later stages of the evaluation budget, and the global optimum is not reached for larger instances.

A difference in problem instances is that the probability that a worker is qualified for a job in [51] is 80%, a property not investigated in this thesis. A lower probability of a worker being able to perform a job is a likely source of many repairs; however the repair frequency in [51] is not reported so further comparison cannot be made. In [51], different stochastic optimization methods are compared, the GA and GA-VNS parameter analysis reveals that algorithms differ in parameter configurations, further supporting the GA sensitivity to parameter selection and problem structure discussed in the literature.

These differences highlight that genetic algorithm performance is highly dependent on problem structure, operator design, and parameter selection. The assignment

constraints in this thesis caused infeasible machine and worker assignment due to the design of the MAC/WAC operators, reducing the effectiveness of recombination during crossover, increasing the importance of mutation-based improvements. Consequently, the search process appears to rely more heavily on local refinement around seeded solutions than on large scale recombination of individuals.

5.6 Industrial Significance

Solving scheduling problems extends beyond the generation of daily production schedules. In a Lean production environment, scheduling can serve as a decision support tool by providing insights into bottlenecks, workforce utilization, and the impact of operational changes. By modeling machines, workers, and sequence-dependent changeovers, the proposed scheduling framework provides a more realistic representation of the production environment than the model in the ERP using traditional dispatching rules.

Workforce Flexibility and cross-training

Worker assignment remains a challenging scheduling problem due to variation in customer demand, worker availability, vacation periods and sick leave. Since workforce flexibility is central in Lean manufacturing, scheduling models that consider worker qualifications and a realistic representation of the production system can provide support for cross-training decisions.

The scheduling model allows the utilization of cross-trained workers where their skill is most required based on current demand, which introduces flexibility in work shifts. The results indicate that dynamic assignment of cross-trained workers improves the production system's ability to meet due dates. This illustrates the benefits of a cross-trained workforce that the lean methodology promotes, but also shows the benefit of being able to rotate the workforce dynamically to respond to varying demand of a variety of products. The results show that the GA can improve the EDD dispatching rule for larger instances, suggesting that more effective allocation of workers contributes to improved schedule quality. Furthermore, the scheduling model makes it possible to investigate hypothetical scenarios, such as increased worker flexibility through cross-training. By comparing schedules under different qualification structures, management can estimate the potential impact of cross-training on throughput, tardiness and resource utilization. For instance, it is possible to simulate the effect 100% cross-training would have on throughput and meeting due dates. Further, worker idle time identified in the schedule, can be used for cross-training opportunity, by identifying which workers become idle on lower demand days and which skills they can acquire.

The model can also help identify competence vulnerabilities. If a schedule depends on a small number of highly specialized workers, this may indicate the need for additional training to improve workforce resilience.

With scheduling, more precise due dates can be set in order to distribute finished jobs evenly and with the right priority. This is beneficial since the packaging station then receives a more even distribution of products to prepare for shipping. By having more due dates, multiple shipping points in a day can be modeled, having products leave the factory earlier than by the end of day, which may speed up warehouse replenishment and customer delivery time. This would require further investigation of how feasible due dates should be set.

Stability

The stability of the schedule is of importance; a mathematically optimal schedule is not always intuitive or understandable, making it less operationally accepted. Earliest due date first is an intuitive algorithm to meet due dates, by different prioritization may cause scepticism although simulations indicate that goals are reached more efficiently. The schedule needs to be presented in a reliable way that shows that orders meet due dates and in a way that is easy to implement by managers, production planners, and operators. There may be many solutions obtaining the same objective, by having different schedules for similar work days, trust in the scheduling module could be compromised. However, by learning to adapt to different worker configurations, the workforce can become trained to adapt to many different circumstances and maintain skill levels over time. It is important to consider how shifting between different product groups affects worker performance, since a common worker assignment is to have a worker operate one specific pool per day. Therefore, implementation also requires behavioral change in habits, and workers must be trained to shift between pools under the duration of a day.

Scheduling as a Lean Decision Support Tool

Lean is evidently popular and has made significant contributions to operations management, and is traditionally implemented without information technology. Modern production environments are dynamic and complex; therefore, integrating lean with the information tools of Industry 4.0 has been suggested to solve dispatching and scheduling in today's production environments. Traditional lean dispatching using Kanban works well in high volume and low mix environments. However, uncertainty in demand, unpredictable material flow, high variety of products moving through different routes with different priorities and sequences of workstations, processing times and due dates make these lean tools insufficient. For instance, Kanban is unable to predict dispatching at the right time with regard to changes in production constraints, customer importance, due dates, resource availability, and current workload. Dispatching rules and optimization then become important to decrease variability, reduce waiting times, increase utilization of resources and improve production smoothness. Production scheduling is traditionally generated within a certain timeframe making it insensitive to unpredictable events that occur in real time, such as machine failure, defects, order cancellation, and raw material shortages; this creates a gap between the plan and real time events that deteriorates the leanness level. It may be hard for production managers and supervisors to respond to all these urgent problems at the right time, which emphasizes the need of scheduling

models for decision support.

Ideally, Lean reduces shared bottlenecks, reduces variability, setup and changeover. However, real factories often contain shared machines and workers, bottlenecks and flexible routes that cannot be reduced. Operational scheduling is recognized as one of ERP's main weaknesses; therefore, scheduling optimization and dual resource job shop scheduling complement Lean by taking the constraints by the Lean production into account.

The Lean methodology reduces the structural complexity of the production system. Scheduling can be used to optimize the remaining complexity that Lean cannot remove, such as worker assignment and job sequencing for shared resources. Multi-skilled workers may not optimally be assigned to one production cell or product group over a work day, but a worker's skills might be better utilized in more places. The functionality of modern technology in Industry 4.0 enables scheduling models to run on real time data, making it possible to get decision support in real time. Lean manufacturing emphasizes continuous improvement and the elimination of waste. Scheduling models can contribute to these goals by identifying bottlenecks and quantifying the effects of resource constraints. Repeated bottlenecks in schedules may indicate where investments in new equipment, additional staffing or process improvements would have the greatest impact.

The scheduling model can also provide information on worker utilization. Periods of low utilization can reveal opportunities for cross-training, maintenance, 5S activities, or other continuous improvement activities. In this way, scheduling becomes not only an operational planning tool but also a mechanism for supporting Kaizen activities and long term process development.

ERP Integration and Customized Scheduling

A common critique of ERP scheduling models is that they often generate schedules based on simplified assumptions that do not fully reflect the constraints of the production system. The main critique with ERPs is that the scheduling functions often provide unrealistic operational schedules. Consequently, planned schedules can be difficult to execute in practice, reducing confidence in planning decisions.

A customized scheduling model provides the possibility to reflect the specific constraints of the company's production environment that may not be possible in an ERP. By complementing existing ERP functionality with a more realistic optimization model, schedules can become more aligned with the actual production conditions. This may improve the reliability of the predictions and provides management with a stronger basis for production planning and decision making.

6

Conclusions

This thesis formulated the production environment at Thorlabs Sweden AB as a DR-FJSP and developed five solution approaches. MILP (Gurobi), SMT (Z3), CP (OR-Tools CP-SAT), a genetic algorithm (GA), and a greedy priority-rule heuristic. Each method was evaluated across synthetic benchmarks and real production data.

On synthetic benchmarks, OR-Tools CP-SAT proved the strongest exact solver, achieving optimality in 57 of 100 instances against 37 for Z3 and 30 for Gurobi, with its global NoOverlap constraint providing a structural propagation advantage that neither MILP's big-M formulations or Z3's OMT approach can replicate. The GA consistently solved all 100 instances suboptimally within 20-80% of the OR-Tools baseline, while remaining tractable at all scales.

On production data, performance was strongly shaped by the problem structure. With end-of-day due dates, the GA outperformed all other methods including OR-Tools, which struggled to prune a search space with near-identical solution cost. Under uniformly distributed intra-day due dates, OR-Tools recovered to prove optimality on 19 of 21 instances, while the heuristic declined substantially and the GA sat reliably between the two.

The parameter search revealed that the GA's MAC- and WAC-crossover operations degraded convergence by triggering frequent feasibility repairs, effectively destroying inherited structural information. The search was found to be driven primarily by mutation-based refinement around heuristic seeds rather than recombination, with small population sizes and low crossover probabilities performing best. Neither seeding strategy produced a significant difference in final fitness, suggesting that initialization affects search dynamics more than solution quality.

Further development is needed before the scheduler is ready for production deployment, particularly the addition of queue times, worker proficiency differences, and availability calendars.

The results in this report show that a customized DR-FJSP model can improve upon simple dispatching rules in a multi-skilled manufacturing environment, and that the choice of method depends critically on the problem structure.

The design of the GA has been analysed through the parameter search and revealed effects of repairs of infeasible crossover operator. Furthermore, two seeding strategies were investigated, showing that greater diversity is not necessarily a contributor to faster convergence or better fitness.

It has been shown that OR-Tools performed best solving benchmark instances, both in terms of runtime and instance size. The GA are able to find optimal solutions for small instances, and for larger instances it finds solutions within a range of 20-80% from the optimal solution.

This thesis concludes that since variety in demand, product customization, and limitation of equipment often are a reality in production environment, the benefits of scheduling optimization should not be ignored, since a customized scheduling model taking the significant constraints into account would be a helpful tool to utilize cross-trained workers under unpredictable demand from customers. In light of digital transformation in the context of Industry 4.0 and Smart manufacturing, the data required to make a realistic model can be made available, and support the lean methodology.

Bibliography

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Cham: Springer International Publishing, 2022.

[2] C. Wang, D. Fan, Y. Liu, S. Ren, and J. Wang, "Dual-resource flexible job shop scheduling considering worker proficiency differences," *Computers & Operations Research, vol. 184, p. 107216*, Dec. 2025.

[3] J. A. Gromicho, J. J. van Hoorn, F. Saldanha-da Gama, and G. T. Timmer, "Solving the job-shop scheduling problem optimally by dynamic programming," *Computers & Operations Research*, vol. 39, no. 12, pp. 2968–2977, 2012.

[4] L. A. Wolsey, *Mixed Integer Programming*. 2008.

[5] T. Perroux, T. Arbaoui, and L. Merghem-Boulahia, "The flexible job shop scheduling problem with setups and operator skills: An application in the textile industry," in *Proceedings of the 18th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2024)*, (Vienna, Austria), pp. 1090–1095, 2024. HAL Id: hal-04811448.

[6] D. Monniaux, "A survey of satisfiability modulo theory," *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–36, 2016.

[7] S. F. Roselli, K. Bengtsson, and K. Åkesson, "SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pp. 547–552, 2018.

[8] F. Rossi, P. van Beek, and T. Walsh, eds., *Handbook of Constraint Programming*, vol. 2 of *Foundations of Artificial Intelligence*. Amsterdam and Heidelberg: Elsevier, 2007. Transferred to digital print.

[9] K. Czerniachowska, R. Wichniarek, and K. Żywicki, "Constraint programming for flexible flow shop scheduling problem with repeated jobs and repeated operations," *Advances in Science and Technology Research Journal*, vol. 17, pp. 280–

293, June 2023.

[10] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[11] G. Teng, "An improved genetic algorithm for dual-resource constrained flexible job shop scheduling problem with tool-switching dependent setup time," *Expert Systems with Applications*, vol. 281, p. 127496, 2025.

[12] K. Rihane, A. Dabah, and A. AitZai, "Adapted q-learning for the blocking job shop scheduling problem," in *Optimization and Learning* (B. Dorronsoro, M. Zagar, and E.-G. Talbi, eds.), vol. 2311 of *Communications in Computer and Information Science*, pp. 51–66, Cham: Springer Nature Switzerland, 2025.

[13] L. Alcamo, G. Bruno, and N. Giovenali, "A reinforcement learning algorithm for dynamic job shop scheduling," in *Communications in Computer and Information Science*, vol. 2372, (Cham), pp. 350–366, Springer Nature Switzerland, 2025.

[14] E. Allender and M. Loui, "Complexity theory," *Computing Handbook, Third Edition: Computer Science and Software Engineering*, 06 2004.

[15] J. Erickson, *Algorithms*. n.p.: Self-published, 1 ed., 2019.

[16] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and G. Schaefer, "Scheduling to minimize flow time metrics," *International Parallel and Distributed Processing Symposium (IPDPS-03), IEEE, 223b-CDROM (2003)*, 01 2003.

[17] P. Brucker, *Scheduling Algorithms*. Berlin Heidelberg: Springer, 5 ed., 2007.

[18] A. Afriansyah and A. Mohruni, "Production planning and control system with just in time and lean production: A review," *Journal of Mechanical Science and Engineering*, vol. 6, pp. 019–027, 01 2021.

[19] W. J. Hopp and M. L. Spearman, *Factory physics*. Maidenhead, England: Irwin Professional Publishing, 2 ed., Apr. 2000.

[20] S. Jituri, B. Fleck, and D. R. Ahmad, "Lean OR ERP – a decision support system to satisfy business objectives," *Procedia CIRP*, vol. 70, pp. 422–427, 01 2018.

[21] G. Culot, G. Nassimbeni, G. Orzes, and M. Sartor, "Behind the definition of Industry 4.0: Analysis and open questions," *International Journal of Production Economics*, vol. 226, p. 107617, 2020.

[22] G. Beier, A. Ullrich, S. Niehoff, M. Reißig, and M. Habich, "Industry 4.0: How

it is defined from a sociotechnical perspective and how much sustainability it includes – a literature review," *Journal of Cleaner Production*, vol. 259, p. 120856, 2020.

[23] M. Al-Amin, M. T. Hossain, M. J. Islam, and S. Kumar Biwas, "History, features, challenges, and critical success factors of enterprise resource planning (ERP) in the era of Industry 4.0," *European Scientific Journal, ESJ*, vol. 19, p. 31, Feb. 2023.

[24] A. Kusiak, "Smart manufacturing," *International Journal of Production Research*, vol. 56, no. 1-2, pp. 508–517, 2018.

[25] J. Davis, T. Edgar, J. Porter, J. Bernaden, and M. Sarli, "Smart manufacturing, manufacturing intelligence and demand-dynamic performance," *Computers & Chemical Engineering*, vol. 47, pp. 145–156, 2012. FOCAPO 2012.

[26] J. Bicheno and M. Holweg, *The lean toolbox*. Buckingham, England: PICSIE Books, 4 ed., Dec. 2008.

[27] M. Ramadan, B. Salah, M. Othman, and A. Arsath Abbasali, "Industry 4.0-based real-time scheduling and dispatching in lean manufacturing systems," *Sustainability*, vol. 12, p. 2272, 03 2020.

[28] R. Ruiz, *Scheduling Heuristics*, pp. 1–24. 01 2015.

[29] J. Kleinberg and E. Tardos, *Algorithm Design: Pearson New International Edition*. London, England: Pearson Education, June 2013.

[30] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, version 29 ed., 2023.

[31] R. O. Edokpia and K. O. Ohikhuare, "Transportation cost minimization of a manufacturing firm using linear programming technique," *Advanced Materials Research*, vol. 367, pp. 685–695, oct 2011.

[32] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, vol. 285 of *International Series in Operations Research & Management Science*. Springer, 5 ed., 2020.

[33] H. P. Williams, *Model Building in Mathematical Programming*. John Wiley & Sons, 1 ed., 2013.

[34] K. G. Murty, *Linear Programming*. Wiley, revised ed., 1983.

[35] A. K. Bhunia, L. Sahoo, and A. A. Shaikh, *Advanced Optimization and Operations Research*, vol. 153 of *Springer Optimization and Its Applications*. Singa-

pore: Springer, 2019.

[36] H. Karloff, *Linear Programming*. Modern Birkhäuser Classics, Birkhäuser, 2009. Reprint of the 1991 edition.

[37] L. G. Khachiyan, "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 20, no. 1, pp. 53–72, 1980.

[38] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, Dec. 1984.

[39] J. MacGregor Smith, "Integer programming $\sum c_j \delta_j, \delta_j \in \{0,1\} \forall j$," in *Springer Optimization and Its Applications*, vol. 175, pp. 133–178, 2021. Scopus Author ID: 6602494601.

[40] L. A. Wolsey, *Integer Programming*. John Wiley & Sons, 1998.

[41] A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds., *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*. Amsterdam: IOS Press, 2009.

[42] S. Nejati and V. Ganesh, "CDCL(Crypto) SAT Solvers for Cryptanalysis," *arXiv preprint arXiv:2005.13415*, 2020.

[43] R. Sebastiani and P. Trentin, "On optimization modulo theories, MaxSMT and sorting networks," *arXiv preprint arXiv:1702.02385*, 2017.

[44] P. Trentin and R. Sebastiani, "Optimization modulo the theory of floating-point numbers," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11716 LNAI, p. 550 – 567, 2019. Cited by: 4.

[45] J.-C. Régin, "A filtering algorithm for constraints of difference," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, (Seattle, WA, USA), pp. 362–367, AAAI Press, 1994.

[46] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, 1980.

[47] M. Wahde, *Biologically inspired optimization methods: an introduction*. Southampton, UK: WIT Press, 2008.

[48] M. Mitchell, *An Introduction to Genetic Algorithms*. The MIT Press, 02 1996.

[49] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*.

Artificial Intelligence, Berlin Heidelberg: Springer-Verlag, 1992. Softcover reprint of the 1st edition.

[50] J. Bergstra, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, 03 2012.

[51] R. Wu, Y. Li, S. Guo, and X. Wenxiang, "Solving the dual-resource constrained flexible job shop scheduling problem with learning effect by a hybrid genetic algorithm," *Advances in Mechanical Engineering*, vol. 10, 10 2018.

[52] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.

[53] R. Ruiz and J. A. Vázquez Rodríguez, "The hybrid flow shop scheduling problem," *European Journal of Operational Research*, vol. 205, pp. 1–18, 08 2010.

A

Appendix 1

A.1 MILP formulation

Sets and parameters
J - set of jobs
O_j - Ordered set of operations for job j
$O = \{(j,i)|j \in J, i \in O_j\}$
M - set of machines
W - Set of workers
$M_{ji} \subseteq M$ - Eligible machines for operation (j,i)
$W_j \subseteq W$ - Eligible workers for job j
$\tau(j,i) \in \{M, W, BW\}$
p^m_{jim} - Machine processing time of operation (j,i) on machine m
p^h_{jiw} - Worker processing time of operation (j,i) using worker w
s^w - Worker setup time between different jobs
c^w - worker cleaning time between job pools
s^m - Machine changeover time between different machines
G - Sufficiently large upper bound on schedule
$pool(j)$ - Job pool label of job j

Decision variables
$t_{ji} \in$ - Start time of operation (j,i)
T_{ji} - Completion time of operation (j,i)
$y_{jim} \in \{0,1\}$ - 1 if operation (j,i) is assigned to machine m
$z_{jiw} \in \{0,1\}$ - 1 if operation (j,i) is assigned to worker w
$\delta^m_{j_1,i_1,j_2,i_2,m} \in \{0,1\}$ - 1 if (j_1,i_1) precedes (j_2,i_2) on machine m
$\delta^w_{j_1,i_1,j_2,i_2,w}$ - 1 if (j_1,i_1) precedes (j_2,i_2) on worker w
C_{max} - makespan

1. Assignment constraint

$$\sum_{m \in M_{ji}} y_{jim} = 0 \quad \sum_{w \in W_j} z_{jiw} = 1 \quad \tau(j,i) = W, \quad \forall (j,i) \in O \tag{A.1}$$

$$\sum_{m \in M_{ji}} y_{jim} = 1 \quad \sum_{w \in W_j} z_{jiw} = 0 \quad \tau(j,i) = M, \quad \forall (j,i) \in O \tag{A.2}$$

$$\sum_{m \in M_{ji}} y_{jim} = 1 \quad \sum_{w \in W_j} z_{jiw} = 1 \quad \tau(j,i) = BW, \quad \forall (j,i) \in O \tag{A.3}$$

2. Completion time

$$T_{ji} = t_{ji} + \sum_{m \in M_{ji}} p^m_{jim} y_{jim} \quad \forall (j,i) \in O, \ \tau(j,i) = M \tag{A.4}$$

$$T_{ji} = t_{ji} + \sum_{w \in W_j} p^w_{jiw} z_{jiw} \quad \forall (j,i) \in O, \ \tau(j,i) = W \tag{A.5}$$

$$T_{ji} = t_{ji} + \sum_{m \in M_{ji}} \max(p^m_{jim}, p^w_{jim}) y_{jim} \quad \forall (j,i) \in O, \ \tau(j,i) = BW \tag{A.6}$$

3. Precedence

$$T_{ji} \leq t_{ji+1} \quad j \in J, \ i = 1, ..., |O_j| - 1 \tag{A.7}$$

4. machine no-overlap. For every distinct pair of (j_1, i_1),(j_2, i_2) and where neither is a W-type, and both are eligible for machine m:

$$t_{j_2 i_2} \geq T_{j_1 i_1} - G(3 - y_{j_1 i_1 m} - y_{j_2 i_2 m} - \delta^m_{j_1, i_1, j_2, i_2, m}) \tag{A.8}$$

$$t_{j_1 i_1} \geq T_{j_2 i_2} - G(3 - y_{j_1 i_1 m} - y_{j_2 i_2 m} - (1 - \delta^m_{j_1, i_1, j_2, i_2, m}))$$

5. Worker no-overlap For every distinct pair (j_1, i_1),(j_2, i_2) that share an eligible worker w:

$$t_{j_2 i_2} \geq T_{j_1 i_1} + \theta - G(3 - z_{j_1 i_1 w} - z_{j_2 i_2 w} - \delta^w_{j_1, i_1, j_2, i_2, w}) \tag{A.9}$$

$$t_{j_1 i_1} \geq T_{j_2 i_2} + \theta - G(3 - z_{j_1 i_1 w} - z_{j_2 i_2 w} - (1 - \delta^w_{j_1, i_1, j_2, i_2, w}))$$

A.2 OR-Tools

OR-Tools

The difference in the CP-sat formulation is introduction of the duration variable d_{ji} and the use of interval variables.

Sets and parameters
J - Set of jobs
M - Set of machines
W - Set of workers
O_j - Ordered set of operations of job j
$O = \{(j,i)|j \in J, i \in O_j\}$ - Set of all operations
M_{ji} - Set of eligible machines for operation (j,i)
W_j - Set of eligible workers for job j
$pool(j)$ - pool of job j
$\tau(j,i) \in \{M, W, BW\}$
p^M_{jim} - Machine processing time of operation (j,i) on machines m
p^W_{ji} - Worker processing time of operation (j,i)
s^w - Worker setup time when switching jobs
s^m - Machine setup when switching machines within a job
c - Cleaning time when a worker switches job pools
G - Sufficiently large upper bound on makespan

Decision variables
t_{ji} - Start time of operation (j,i)
T_{ji} - Completion time of operation (j,i)
d_{ji} - Duration of operation (j,i)
$y_{jim} \in \{0,1\}$ - 1 if operation (j,i) is assigned to machine m
$z_{jiw} \in \{0,1\}$ - 1 if operation (j,i) is assigned to worker w
$I^m_{jim} = (t_{ji}, d_{ji}, T_{ji}, y_{jim})$ - Interval of operation (j,i) using machine m
$I^w_{jiw} = (t_{ji}, d_{ji}, T_{ji}, z_{jiw})$ - Interval of operation (j,i) using worker w
C_{max} - makespan

Objective $\min C_{max}$

Constraints

1. Domain constraints

$$t_{ji} \in [0,G], \quad T_{ji} \in [0,G], \quad d_{ji} \in [0,G] \quad \forall (j,i) \in O \tag{A.10}$$

2. Assignment constraints

$$\sum_{m \in M_{ji}} y_{jim} = 1, \quad \sum_{w \in W_j} z_{jiw} = 0 \quad \forall (j,i) \in O : \tau(j,i) = M \tag{A.11}$$

$$\sum_{m \in M_{ji}} y_{jim} = 0, \quad \sum_{w \in W_j} z_{jiw} = 1 \quad \forall (j,i) \in O : \tau(j,i) = W \tag{A.12}$$

$$\sum_{m \in M_{ji}} y_{jim} = 1, \quad \sum_{w \in W_j} z_{jiw} = 1 \quad \forall (j,i) \in O : \tau(j,i) = BW \tag{A.13}$$

3. Duration constraints

$$y_{jim} = 1 \Rightarrow d_{ji} = p^M_{jim} \quad \forall (j,i) \in O : \tau(j,i) = M \tag{A.14}$$

$$z_{jiw} = 1 \Rightarrow d_{ji} = p^W_{ji} \quad \forall (j,i) \in O : \tau(j,i) = W \tag{A.15}$$

$$(y_{jim} = 1 \wedge z_{jiw} = 1) \Rightarrow d_{ji} = \max(p^W_{ji},\, p^M_{jim}) \quad \forall (j,i) \in O : \tau(j,i) = BW \tag{A.16}$$

4. Completion time is enforced via the interval variables

$$I^M_{jim} = (t_{ji}, d_{ji}, T_{ji}, y_{jim}) \tag{A.17}$$

$$I^W_{jiw} = (t_{ji}, d_{ji}, T_{ji}, y_{jim}) \tag{A.18}$$

5. Precedence constraint

$$T_{j,i} \leq t_{j,i+1} \quad \forall j \in J,\ i = 1, \ldots, |O_j| - 1 \tag{A.19}$$

6. Machine setup within a job. For consecutive operations (j,i) and $(j,i+1)$ assigned to machines m_1 and m_2 respectively:

$$y_{j,i,m_1} = 1 \wedge y_{j,i+1,m_2} = 1 \wedge m_1 \neq m_2 \Rightarrow t_{j,i+1} \geq T_{j,i} + s^m \quad \forall j \in J,\ i = 1, \ldots, |O_j| - 1 \tag{A.20}$$

7. Machine no-overlap. For all operations of type M or BW:

$$\text{NoOverlap}\Big(\Big\{ I^M_{jim} \;\Big|\; (j,i) \in O,\ \tau(j,i) \neq W,\ m \in M_{ji} \Big\}\Big) \quad \forall m \in M \tag{A.21}$$

8. Worker no-overlap with setup. For every pair of distinct operations $(j_1, i_1) \neq (j_2, i_2)$ sharing an eligible worker w, let

$$\delta = \begin{cases} s^w & j_1 \neq j_2 \\ 0 & \text{otherwise} \end{cases} + \begin{cases} c & pool(j_1) \neq pool(j_2) \\ 0 & \text{otherwise} \end{cases}$$

Then:

$$z_{j_1 i_1 w} = 1 \wedge z_{j_2 i_2 w} = 1 \Rightarrow (t_{j_2 i_2} \geq T_{j_1 i_1} + \delta) \ \vee \ (t_{j_1 i_1} \geq T_{j_2 i_2} + \delta) \quad \forall w \in W \tag{A.22}$$

9. Makespan definition

$$C_{max} \geq T_{ji} \quad \forall (j,i) \in O \tag{A.23}$$

A.3 Heuristics

Heuristic dispatching rule

Algorithm 1: Heuristic algorithm

Input:
Jobs $J = 1 \dots n$, operations O_j, machines M, workers W and job pool g_j
Feasibility sets: $M_j \subseteq M$, $W_j \subseteq W$
Processing times $P_{j,i}^{m,w}$
Output: Schedule S, performance metrics
Initialize availability times:
$A_j \leftarrow 0 \; \forall j \in J$ ▷ Job availability
$A_m \leftarrow 0 \; \forall m \in M$ ▷ Machine availability
$A_w \leftarrow 0 \; \forall w \in W$ ▷ Worker availability
Sort jobs in order of priority rule
Sort workers in increasing skill flexibility
for each job $j \in J$ **do**
    Get job's pool
    Determine feasible workers W_j
    Select worker:
    $w^* \leftarrow \arg\min_{w \in W_j} A_w$
    if w^* has multiple minimizers **then** Select w^* with lowest flexibility
    end if
    for operation $i \in O_j$ **do**
        Determine feasible machines M_j
        Select machine:
        $m^* \leftarrow \arg\min_{m \in M_j} A_m$
        if m^* has multiple minimizers **then** Select m^* with lowest flexibility
        end if
        Compute start time:
        $t_{start} \leftarrow \max(A_j, A_m^*, A_w^*)$
        Compute completion time:
        $t_{end} \leftarrow t_{start} + p_{j,i}^{m^*,w^*}$
        Get processing times for j, i at m, w.
        Compute earliest possible start time
        Update:
        $A_j \leftarrow t_{end}$
        $A_{m^*} \leftarrow t_{end}$
        $A_{w^*} \leftarrow t_{end}$
        Store $j, i, m^*, w^*, t_{start}, t_{end}$ in S
    end for
end for
return S, metrics

A.4 Genetic Algorithm

Decoder

Algorithm 2: Decoding algorithm for GA

Input:
OSC/MAC/WAC chromosome
Processing times $P_{j,i}^{m,w}$
s^w, c^w, s^m ▷ Worker, pool and machine setup times.
Output: Schedule S, performance metrics
Initialize lookup tables:
$A_j \leftarrow 0 \; \forall j \in J$ ▷ Job availability time
$A_m \leftarrow 0 \; \forall m \in M$ ▷ Machine availability time
$A_w \leftarrow 0 \; \forall w \in W$ ▷ Worker availability time
$O_j^{\text{next}} \leftarrow 0 \; \forall j \in J$ ▷ How to interpret job number in chromosome
$F_w^j \leftarrow -1 \; \forall w \in W$ ▷ Former job j handled by worker w
$F_w^p \leftarrow -1 \; \forall w \in W$ ▷ Former pool visited by worker
$F_j^m \leftarrow -1 \; \forall j \in J$ ▷ Former machine used in job j
$F_j^o \leftarrow -1 \; \forall j \in J$ ▷ Former operation of job j
$F_j^a \leftarrow 0 \; \forall j \in J$ ▷ Former machine active or not, 0=False
$O_j^{\text{last}} \leftarrow o_{jk} \; \forall j, \text{where} \quad k \quad \text{is the last operation in} \quad j$
Initialize metrics:
makespan $\leftarrow 0$
$C_{\text{sum}} \leftarrow 0$
tardiness $\leftarrow 0$
nr_tardy_jobs $\leftarrow 0$
lateness $\leftarrow 0$
for each position i,with job $j \in OSC$ **do**
    Get assignments:
    $j^* \leftarrow j$ ▷ Current job
    $o^* \leftarrow O_{j^*}^{\text{next}}$ ▷ Current job's operation
    $m^* \leftarrow \text{MAC}_i$ ▷ Assigned machine
    $w^* \leftarrow \text{WAC}_i$ ▷ Assigned worker
    $p^* \leftarrow \text{pool}(j^*, o^*)$ ▷ Operation's pool
    Get and compute processing times:
    $(p^w, p^m, p^s, p^t, p^q) \leftarrow P_{j^*,o^*}^{m^*,w^*}$ ▷ Extract all processing times from data
    worker_active $\leftarrow$ False, machine_active $\leftarrow$ False ▷ Track if operation requires one or two resources
    if $p^w > 0$ **then**
        worker_active $\leftarrow$ True
    end if
    if $p^m > 0$ **then**
        machine_active $\leftarrow$ True
    end if

Compute changeover:
machine_change_setup $\leftarrow 0$
prev_op $\leftarrow F^o_{j^*}$
prev_machine $\leftarrow F^m_{j^*}$
prev_machine_active $\leftarrow F^a_{j^*}$
$\text{cond}_1 \leftarrow$ machine_active $\wedge$ prev_machine_active
$\text{cond}_2 \leftarrow$ prev_machine $\neq m^*$
if $\text{cond}_1 \wedge \text{cond}_2$ **then**
    machine_change_setup $\leftarrow$ machine_change_time
end if
worker_change_setup $\leftarrow 0$
if worker_active **then**
    prev_job $\leftarrow F^j_{w^*}$
    prev_pool $\leftarrow F^p_{w^*}$
    if prev_job $\neq j^* \wedge$ prev_job $\neq -1$ **then**
        worker_change_setup $\leftarrow$ worker_change_setup $+ s^w$
    end if
    if prev_pool $\neq p^* \wedge$ prev_pool $\neq -1$ **then**
        worker_change_setup $\leftarrow$ worker_change_setup $+ c^w$
    end if
end if
Compute start time:
if worker_active **then**
    worker_free_time $\leftarrow A_{w^*}$ + worker_change_setup
else
    worker_free_time $\leftarrow 0$
end if
if machine_active **then**
    machine_free_time $\leftarrow A_{m^*}$
else
    machine_free_time $\leftarrow 0$
end if
job_free_time $\leftarrow A_{j^*}$ + machine_change_setup
start_time $\leftarrow$ max(job_free_time, machine_free_time, worker_free_time)
Compute finish times:
if worker_active $\wedge$ machine_active **then**
    duration $\leftarrow \max(p^w, p^m)$
    latest_finish_time $\leftarrow$ start_time + duration
else if machine_active **then**
    duration $\leftarrow p^m$
    latest_finish_time $\leftarrow$ start_time + duration
else if worker_active **then**
    duration $\leftarrow p^w$

latest_finish_time ← start_time + duration
end if
Compute metrics:
makespan ← max(makespan, latest_finish_time)
if $O_{j^*}^{\text{last}} = o^*$ **then**
tard ← get_tardiness(latest_finish_time, j^*)
late ← get_lateness(latest_finish_time, j^*)
$C_{\text{sum}} \leftarrow C_{\text{sum}}$ + latest_finish_time
tardiness ← tardiness + tard
lateness ← lateness + late
if tard>0 **then**
nr_tardy_jobs ← nr_tardy_jobs + 1
end if
end if
Store assignment:
schedule ← j^*, o^*, m^*, w^*, latest_finish_time
Update lookup tables:
if worker_active **then**
$A_{w^*} \leftarrow$ latest_finish_time
$F_{w^*}^{j} \leftarrow j^*$
$F_{w^*}^{p} \leftarrow p^*$
end if
if machine_active **then**
$A_{m^*} \leftarrow$ latest_finish_time
end if
$O_{j^*}^{\text{next}} \leftarrow O_{j^*}^{\text{next}} + 1$
$A_{j^*} \leftarrow$ latest_finish_time
$F_{j^*}^{m} \leftarrow m^*$
$F_{j^*}^{o} \leftarrow o^*$
$F_{j^*}^{a} \leftarrow$ machine_active
end for
metrics ← makespan, tardiness, lateness, nr_tardy_jobs
return schedule, metrics

Department of Electrical Engineering
CHALMERS University of Technology
Gothenburg, Sweden
www.chalmers.se

CHALMERS