# Optimizing Conflict-Free-Transportation-Constrained Flexible Manufacturing Systems

Embedded Systems Master Thesis

| Full Name | Student ID |
|---|---|
| Roel van Os | 1638521 |

Graduation Supervisor: Prof.dr.ir. Twan Basten

Eindhoven, August 29, 2024

# Abstract

With the latest advancements in manufacturing technologies and customization of systems and products, there is a rise in required flexibility for manufacturing processes. Automated material handling and transportation of semi-finished products between different parts of the manufacturing systems brings such added flexibility but carries along various new challenges. The resulting optimization problems required to find the shortest makespan schedules are complex and often hard to solve as automated vehicles should avoid colliding with one another in the flexible manufacturing system while finishing all required jobs in the shortest possible time. One relevant optimization problem is the Conflict-Free-Transporation-constrained Flexible Job-Shop Scheduling Problem (CFTFJSSP). Only a limited amount of research deals with solving the CFTFJSSP. The available literature proposed heuristic approaches to solve this optimization problem, but there is no exact solution approach yet for finding optimal makespan schedules. This thesis addresses the challenge of finding this exact solution approach to the CFTFJSSP.

A Logic-Based Benders Decomposition (LBBD) approach is developed that decomposes the problem into the Transportation-constrained Flexible Job-Shop Scheduling Problem (TFJSSP) as master problem and Conflict-Free Routing Problem (CFRP) as subproblem. General-purpose mathematical definitions are provided for the specification and optimization of the CFTFJSSP and the related decomposed optimization problems. These mathematical definitions serve as solution-independent references for defining exact and heuristic solution approaches for the CFTFJSSP and related decomposed optimization problems. Using these mathematical references, this thesis develops exact Constraint Programming (CP) and Mathematical Programming (MP) models for the master and subproblem, respectively.

The LBBD-based approach is proven to outperform existing approaches to solving the CFTFJSSP on the currently available benchmark instances in the literature in terms of solution quality through various experiments. It consistently finds the smallest makespan solutions compared to other proposed heuristic algorithms. For most benchmark instances, the LBBD-based approach was able to find optimal makespan values. Because of time-boxing, in a few cases, optimality of the found solution cannot be guaranteed. One application of the approach is to determine the optimal number of vehicles for given combinations of system layouts and job sets. Our experiments show that the heuristic approach proposed in the literature for determining the number of needed vehicles provides suboptimal results in all but the smallest benchmark instances.

To encourage future development and research, all the results and software developed for this thesis are publicly available as reference material.

# Preface

During the academic year of 2023-2024, I had the opportunity to work on this thesis regarding Optimizing Conflict-Free-Transportation-Constrained Flexible Manufacturing Systems as part of the Embedded Systems Master at Eindhoven University of Technology.

First, I would especially like to thank my supervisor Twan Basten. When I started this thesis months ago, I had no idea what my exact research topic was going to be. Twan allowed me to independently traverse this complex research landscape and find a research topic that was, while difficult, a very inspiring challenge. Also, as my supervisor, he taught me a lot about formalizing proper mathematical problem descriptions and academic writing through his attention to detail and clear feedback. However, foremost, I want to thank him for sparking my enthusiasm for academic research through our many interesting discussions about this project over a good cup of coffee.

I also want to thank my fellow students who worked on their graduation project parallel to mine. Together, we've spent many hours on the fourth floor of Flux, motivating each other and pointing out the occasional break when needed.

Lastly, I like to thank my partner Femke for her patience and support when writing this thesis. When things got stressful, she was always there to give me the needed moral support.

Eindhoven, August 29, 2024

# Contents

# Acronyms

**AGVs** Automated Guided Vehicles. 1

**CFR** Conflict-Free Routing. 1

**CFRP** Conflict-Free Routing Problem. i, 1, 6, 35, 37, 39

**CFTFJS** Conflict-Free-Transporation-constrained Flexible Job Shop. 54

**CFTFJSSP** Conflict-Free-Transporation-constrained Flexible Job-Shop Scheduling Problem. i, 2, 7, 57

**CP** Constraint Programming. i, 2, 19, 31

**DOcplex** IBM Decision Optimization CPLEX Modeling. 12

**FMS** Flexible Manufacturing System. 1

**ILP** Integer Linear Programming. 11

**IPODS** Integrated Production and Outbound Distribution Scheduling. 5

**LBBD** Logic-Based Benders Decomposition. i, 13, 61

**LP** Linear Programming. 11

**MILP** Mixed-Integer Linear Programming. 11

**MP** Mathematical Programming. i, 1, 11

**SchedPT** Scheduling Problems in Manufacturing with Transportation. 5

**TFJS** Transportation-constrained Flexible Job Shop. 28

**TFJSSP** Transportation-constrained Flexible Job-Shop Scheduling Problem. i, 2, 6, 25, 31

**TJSP** Transportation-constrained Job-Scheduling Problem. 2, 5, 25

**TJSSP** Transportation-constrained Job-Shop Scheduling Problem. 5

**VRP** Vehicle Routing Problem. 35

# Chapter 1

# Introduction

Since the introduction of the *flow shop* by Johnson in 1954 [23], scheduling has been regarded as an important and independent research area [38]. From this point in time, numerous variations of the flow shop, including the *job shop* and *open shop*, emerged, giving rise to optimal job scheduling as a class of optimization problems in the scheduling domain. Simultaneously, throughout the late 20th century, manufacturing became more automated, eventually leading to the introduction of the Flexible Manufacturing System (FMS). An FMS can be described as a manufacturing line that easily adapts to produce customer-specific products, using robots and automated processing stations [37]. With added flexibility, the efficient scheduling of jobs within an FMS becomes increasingly complex. Optimal job scheduling has been an important part of optimizing the performance of these FMSs as the production of goods can be formalized by a set of jobs each consisting of a set of operations. These job optimization problems come with various objective values to minimize, such as completion times (makespan), lateness, and tardiness of jobs, which are common measures of efficiency within FMS.

With the modern advancements in FMSs, manufacturing processes require ever more flexibility. Traditional optimal job scheduling focuses on the sequencing of operations. Most of these problems, assume fixed machine allocations and transportation of materials and semi-finished products are left out of scope. However, material handling and transportation of semi-finished products are an integral part of the manufacturing system and its automation is highly anticipated [26]. Moreover, in modern FMSs, internal transportation is increasingly becoming a bottleneck that requires optimization [22]. New affordable and flexible autonomous transportation technologies, such as Automated Guided Vehicles (AGVs), have great prospects to optimize the manufacturing procedure. With improved internal transportation, decoupling of machines and operations, which can be allocated to different machines, can even further improve internal transport and enhance the flexibility of FMSs. By incorporating flexible job scheduling, each operation can be assigned to a machine that best fits the production and vehicle routing scheme. Job shops with flexible operation allocation are referred to as *flexible job shops*. Combining these flexible job shops with internal transportation considerations is a relevant new research trend as these characterize the most flexible manufacturing processes.

New challenges arise when combining the integration of internal transportation with optimal job scheduling. Moving autonomous vehicles must avoid possible bottlenecks such as collisions, deadlocks, and livelocks when moving semi-finished jobs or materials on the manufacturing floor. The optimization problem related to these three bottlenecks is often named the Conflict-Free Routing Problem (CFRP). The CFRP is a frequently researched topic. However, the combination of Conflict-Free Routing (CFR) with flexible job-shop scheduling is scarcely studied, despite its practical relevance. The combination of eliminating conflicts and maximizing production efficiency is an interesting challenge in the integral scheduling of flexible jobs and conflict-free internal transportation.

The next step is to devise methods that can effectively deal with the identified challenges. Exact solutions for optimal job scheduling problems are mostly obtained employing Mathematical Programming

(MP) and Constraint Programming (CP). Both techniques usually specify an objective function and a set of constraints to specify the problem at hand. CP has been shown to outperform MP for optimal job scheduling in computation time and ability to deal with relatively larger problem sizes [13, 27, 16, 17]. However, exact solutions are a computation-intensive task, and both solution techniques scale poorly to large problem sizes. Often, (meta)heuristics are computationally more efficient and can deal with larger problem sizes. Heuristic approaches, however, do not guarantee finding globally optimal solutions but instead provide near-optimal solutions. A specific class of problems can use decomposition-based optimization. This technique involves breaking down a larger optimization problem into smaller subproblems that can be solved by either exact techniques as CP or heuristic approaches. Flexible job-shop scheduling problems with internal transportation offer a natural decomposition as we deal with both an optimal job scheduling problem and the CFRP. Compared to the overall problem, these decomposed problems are often smaller and simpler, which can be more easily solved using exact methods such as CP and MP.

While this new line of research shows promise, the current literature is widely scattered. The available studies regarding optimal job scheduling with internal transportation mainly focuses on the flow-shop- or job-shop scheduling problems. Flexible job shops are often not considered, meaning machine allocation is typically fixed. These works usually ignore the CFRP, and transportation times are considered fixed intervals.

The combination of optimal job scheduling with CFR has been researched very limitedly and presents numerous obstacles to their integration. Optimal job scheduling with internal transportation and the CFRP have been studied exclusively as separate topics for a long time. Eventually, these two fields merged. While these two lines of research integrate well, there are often variations in the presented concepts. As a result, the conflict-free internal transportation with optimal job scheduling research lacks consistency in problem definitions, benchmarking, and solutions, making it difficult to expand this line of work. Additionally, most literature solely presents their findings and theoretical models. This makes replicating and comparing new solutions difficult due to the lack of model implementation or software tools. Furthermore, the current state of the art considers primarily (meta)heuristic solving approaches. Most of the available benchmark results can, therefore, not be compared with an exact solution approach, leaving the quality of the solving technique unclear.

In this research, we aim to integrate conflict-free internal transporter routing into the flexible job-shop optimization problem. We refer to this integrated optimization problem as the Conflict-Free-Transporation-constrained Flexible Job-Shop Scheduling Problem (CFTFJSSP). The main objective is to find exact shortest makespan solutions for the CFTFJSSP, in an attempt to outperform the currently available metaheuristic approaches available in literature on existing CFTFJSSPs benchmarks. To solve the CFTFJSSP, a decomposition-based approach is proposed, as the overall problem naturally decomposes into separate optimization problems. The CFTFJSSP is decomposed into a Transportation-constrained Flexible Job-Shop Scheduling Problem (TFJSSP) master problem and a CFRP subproblem. The overall CFTFJSSP is then solved by iterating over master and subproblem instances, where infeasible subproblem instances provide information to constrain the master problem instances, via so-called *cuts*, in following iterations. A second objective of this work is to provide clear, uniform problem definitions, model implementations, and software for future work to expand on the foundations of this research.

The CFTFJSSP is an example of what we refer to as a Transportation-constrained Job-Scheduling Problem (TJSP) (i.e., job scheduling problems with internal transportation). This thesis makes the following contributions to optimization research in TJSPs:

- General-purpose mathematical definitions for the CFTFJSSP and for the TFJSSP and CFRP used in its decomposition. These formalizations may serve as solution-independent reference definitions.
- The first exact approach to solving the CTFJSSP, based on the Logic-Based Benders Decomposition [20, 21]. The decomposition uses an innovative conflict-driven cut that uses information from routing conflicts from CFR subproblems to speed up convergence of the iterative solution process.
- CP and MP implementations of the TFJSSP and CFRP for construction of the decomposition-based CFTJSSP. Various alternative CP and MP solutions for the CFRP are evaluated on computational efficiency, to select the best implementation to be used in the Benders decomposition.
- An evaluation of the proposed approach on existing benchmarks from literature, showing that the

approach outperforms existing heuristic approaches in solution quality on all available benchmarks.

- Open-source software tools allowing further development on presented TJSP implementations and performance comparison on benchmarks.

This report is organized in multiple chapters. Chapter 2 discusses the work related to the main topics of this research project. Chapter 3 discusses mathematical notations, solving techniques, and decomposition-based optimization that is relevant to the exact solution of the CFTFJSSP. The mathematical problem statement and constraints related to the flexible job-shop scheduling problem with internal transportation is given in Chapter 4 along with a constraint programming model that implements the problem. CP is chosen as implementation technique because it has been shown to outperform other approaches. Chapter 5 discusses the definition of the CFRP. Furthermore, multiple CP and MP models are implemented to solve this problem. Their comparative performance is evaluated to select the best option for a decomposition-based approach to solving the CFTFJSSP. Chapter 6 discusses the mathematical definition and constraints of the CFTFJSSP. In Chapter 7, an exact decomposition-based solution to the CFTFJSSP is described. The performance of this new solution and tools to reproduce the results are discussed in Chapter 8. Lastly, Chapter 9 summarizes the main conclusions of this work and it discusses potential future work.

## Chapter 2

# Related Work

Although the combination of optimal job scheduling and transportation is not very extensively researched, some relevant work has been done. This section gives an overview of the related studies that preceded our research. We start by giving an overview of the foundation and classification of optimal job scheduling with transportation considerations. Then, we summarize the foregoing research for optimal job scheduling with fixed internal transportation times. Hereafter, we introduce previous works related to AGV control, specifically conflict-free routing. Lastly, we look at the combined research of optimal job scheduling with conflict-free routing.

**Optimal Job Scheduling with Transportation Classification** According to Lee and Chen [28], the earliest optimal job scheduling publication with transportation considerations is the paper from Maggu and Das [32]. They consider a flow shop makespan problem where transporters move semi-finished jobs from one machine to the next. Simultaneously, research by Potts [39] showed a job-scheduling problem where the completion time of jobs is defined as the time when a job arrives at a customer. The publications by Maggu and Das [32] and Potts [39] defined two branches of optimal job scheduling with transportation considerations. One focuses on the intermediate transport of jobs from one processing unit to another, while the other focuses on the distribution of finished jobs to customers. These two different types of transportation are classified by Lee and Chen [28] as *type 1* and *type 2* transportation, respectively. Later, scheduling problems with type 2 transportation considerations would be commonly known as the Integrated Production and Outbound Distribution Scheduling (IPODS) problem, as introduced by Chen [8]. The research in this report focuses on job scheduling problems with type 1 transportation considerations. Hosseini *et al* [22] names the problem setting for the integrated planning of manufacturing and internal logistics (type 1 transportation) as Scheduling Problems in Manufacturing with Transportation (SchedPT). However, we prefer a naming connecting to the standard naming in the domain: Transportation-constrained Job-Scheduling Problems (TJSPs), where *Transportation-constrained* refers to *internal* transportation constraints. A rather small share of articles on TJSPs considers complex manufacturing environments with multiple transporters and additional scheduling flexibility [22]. Furthermore, in most TJSP research, transportation times are constant between machines, and detailed routing is often left out of scope.

**Transportation-constrained Job-Scheduling Problems:** One specific instance of the class of TJSP, Transportation-constrained Job-Shop Scheduling Problem (TJSSP), has been studied, as reviewed by Xie and Allen [49] and Nouri *et al.* [36]. This problem considers a traditional job-shop environment, forming the basis of the flexible job-shop problem we explore in this work. Most of the current state-of-the-art TFJSSP solutions are inspired by the foundations laid by Bilge and Ulusoy [4]. They consider a traditional job shop with a makespan minimization objective. The transportation is performed by AGVs that start at a load/unload station. AGVs do not return to the load/unload station after transportation and transfer directly to the next job, resulting in sequence-dependent travel times. They suggest a decomposition-based time-window heuristic in which a schedule with a transportation time window is provided and subsequently, a feasible transportation solution for that time window is searched. If no such solution is found, the process starts over with a revised schedule and a new time window. The algorithm was tested by introducing a benchmark set of problem instances. Later, Ulusoy *et al.* [47] proposed a genetic algorithm for solving the same problem, outperforming the solution of [4] on the same benchmark. This genetic algorithm approach was eventually improved by Abdelmaguid *et al.* [1] who introduced a hybrid scheme with a genetic algorithm to solve the scheduling of machines and a heuristic for scheduling the AGVs. The introduction of the heuristic increases the efficiency of the genetic algorithm search, resulting in even better benchmark results. Several improved heuristics for the benchmark of Bilge and Ulusoy [4] followed, such as a tabu search algorithm by Zheng *et al.* [51] and a colored Petri net-based heuristic by Baruwa and Piera [2]. Also, several problem extensions were researched, such as the multi-objective TJSSP by Reddy and Rhao [41].

Deroussi and Norre [12] adapted the benchmark of Bilge and Ulusoy to deal with a Transportation-constrained Flexible Job-Shop Scheduling Problem (TFJSSP). That is, the allocation of operations to machines was made flexible. Zhang *et al.* [50] provided the best heuristic method for solving the benchmark instances of Deroussi and Norre. They suggested a genetic algorithm for solving the machine allocation problem for operations and a tabu search procedure for sequencing the operations on each machine. Several other heuristic approaches solved the benchmark instances provided by Deroussi and Norre. Deroussi [12] suggests a hybrid particle swarm optimization algorithm with a local search. Nouri *et al.* [35] suggested an alternative approach of using a genetic algorithm with a tabu search, unfortunately not outperforming the earlier proposed method by Zhang *et al.*. While these studies showed great results regarding the TJSSP and TFJSSP, all their implementations are heuristic-based and do not provide exact solutions to the benchmark problems.

El Khayat *et al.* [14] took another research direction and provided exact mathematical and constraint programming models for the TJSSP. Both models were implemented and evaluated on their computation time to find an optimal solution. Later, Ham [17] proposed another improved constraint programming model outperforming all other Bilge's benchmark approaches in the literature at that point. Ham *et al.* [16] also suggested the only constraint programming approach for the TFJSSP, providing the exact smallest makespan solution for all the benchmark instances of Deroussi and Norre for the first time. Ham indicated the successful results were obtained by using IBM's CP Optimizer solver, which uses machine-learning techniques to find the best-combined method of large neighborhood and completion strategies. These exact solutions form an important base for researching job scheduling with integrated transportation. However, all reviewed studies so far always assume fixed transportation times, ignoring conflict avoidance.

**AGV Control & Conflict-Free Routing:** Routing and scheduling of AGVs is widely researched as a separate topic. Qui *et al.* [40] discuss two main aspects for the control of AGV systems: scheduling and routing. Scheduling deals with the dispatching of AGVs to a pickup or drop-off task (transfer). In the case of a TJSP, this usually is part of the optimal job scheduling problem as seen before. Routing, on the other hand, deals with the determination of a suitable route to perform the scheduled task. Important contributions in routing are the introduction of the Conflict-Free Routing Problem (CFRP) by Broadbent *et al.* [7] and its solution proposed by Kim and Tanchoco [24]. They proposed a Dijkstra-based algorithm to determine the shortest conflict-free route for an AGV in a bi-directional flow path network. Where Kim and Tanchoco focused only on conflict-free pathfinding for vehicles in a grid, Krishnamurthy [25] generalized the problem to a scheduling context. They proposed an optimization approach to reduce the overall makespan by implementing a column generation-based approach. Their method assumes the allocation of tasks to vehicles is already available. Much research followed on the study of AGV control and the CFRP. However, most research does not consider simultaneous job and AGV scheduling and assumes optimal job schedules to start with.

**Job Scheduling with Conflict-Free Routing:** The first important combination of scheduling and routing was investigated by Corréa *et al.* [9]. They proposed a hybrid method for dispatching and conflict-free routing of AGVs in FMSs by decomposing routing and scheduling of vehicles using the Logic-Based Benders Decomposition [20, 21]. The research, however, does not consider the scheduling of machines, but only the scheduling of the vehicles. More decomposition-based approaches followed. Nishi *et al.* [34]

came up with another decomposition-based approach, now considering simultaneous machine scheduling and the CFRP in flexible *flow* shops (referred to as hybrid flow shops in [34]) to minimize the total weighted tardiness. Riazi *et al.* [43] and Riazi and Lennartson [42] expanded on the work of Corréa *et al.* [9]. They improved their decomposition method to deal with bigger routing graphs and solve the subproblems using constraint programming and a simple heuristic that can provide good solutions quickly. Despite these contributions, decomposition-based optimization for *flexible job-shop* scheduling and CFR of AGVs remains unexplored.

Besides decomposition-based approaches, several pure (meta)heurstic approaches exist for integrated job scheduling with conflict-free routing problems. Saidi-Mehrabad *et al.* [44] proposed an approach for solving a TJSSP and CFRP without the use of a problem decomposition but applying a two-stage Ant Colony Algorithm to a mixed-integer linear program. The introduced problem however takes vehicles that escort a job until completion, i.e., all transports for a single job are done by the same vehicle and vehicles only become available again after a job is complete. Simultaneously, Umar *et al.* [48] suggested a hybrid genetic algorithm to integrate conflict-free AGV control activities in the TJSSP. Their iterative approach first considers machine scheduling, AGV scheduling, and path planning, then detecting and avoiding route conflicts. Another TJSSP approach considering CFR was introduced by Zhou and Lei [52], optimizing both makespan and energy consumption using a special grey wolf optimization algorithm. Lyu *et al.* [31] were the first to consider the Conflict-Free-Transporation-constrained Flexible Job-Shop Scheduling Problem (CFTFJSSP) that considers both the TFJSSP and CFRP. The solution applies a genetic algorithm for solving the job/machine scheduling and AGV scheduling, and they combined it with a time-window-based Dijkstra algorithm for finding the shortest conflict-free path. Recently, this work was expanded on by Liu *et al.* [29], who replaced the genetic algorithm with a self-learning genetic algorithm and showed that this successfully enhances the effectiveness of the algorithm. None of these implementations consider exact solutions for comparison of algorithms and they use slight variations in their problem descriptions, making it hard to compare approaches.

In contrast to the surveyed work, this thesis proposes an exact decomposition-based approach to the CFTFJSSP, using the Logic-Based Benders Decomposition. This includes solution-independent mathematical definitions and both CP and MP models for the optimization problems part of the decomposition. The approach is shown to outperform the state-of-the-art heuristic approaches in solution quality, finding optimal solutions for most of the benchmarks available in the literature. The tools developed in this work and the benchmarks are made available to the community at large to facilitate experimental comparison in future research. The tools and models are set up to facilitate experimentation with different problem variants and decomposition approaches.

A summary of the most important articles relevant to the work in this thesis can be seen in Table 1.

| **Article** | **Job scheduling** | **Flexible job-shop scheduling** | **Conflict-Free Routing** | **Exact Solution** |
|---|---|---|---|---|
| Corréa *et al.* [9] | | | ✓ | ✓ |
| Riazi and Lennartson [42] | | | ✓ | ✓ |
| Ham *et al.* [16] | ✓ | ✓ | | ✓ |
| Nishi *et al.* [34] | ✓ | | ✓ | ✓ |
| Saidi-Mehrabad *et al.* [44] | ✓ | | ✓ | |
| Umar *et al.* [48] | ✓ | | ✓ | ✓ |
| Lyu *et al.* [31] | ✓ | ✓ | ✓ | |
| Liu *et al.* [29] | ✓ | ✓ | ✓ | |
| **This research** | ✓ | ✓ | ✓ | ✓ |

**Table 1:** Summary of important related literature.

# Chapter 3

# Background

Throughout this report, different notations and techniques are used. This section provides an overview and introduction to both. We discuss specific notations used to define constrained mathematical sets. To become familiar with scheduling problems, an exemplary simple scheduling problem is specified. Additionally, background information is provided for several concepts and techniques related to the scheduling problems studied in this research.

## 3.1 | Notations

Throughout this report, various mathematical notations are used. The sets of natural, integer, rational, and real numbers are denoted by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$, respectively. $\mathbb{N}_0$ denotes the natural numbers including 0. Superscripts can specify a range within a set of numbers; $^+$ specifies all positive numbers and inequality symbols such as $<, >, \leq, \geq$ can be used to specify specific bounds.

## 3.2 | A Simple Scheduling Problem

This research studies complex scheduling problems. To become familiar with scheduling problems and the techniques related to solving them, we specify a simple scheduling problem that is used as a leading example for upcoming sections.

Consider a simple scheduling setup where five operations need to be distributed over two different machines. The duration of operations may differ from one another and for each machine. The goal is to map all operations onto machines to minimize the total schedule duration. The set of operations is denoted as $O = \{o_0, \ldots, o_4\}$ where $o_i$ represents any operation $i$. Furthermore, the set of machines is denoted as $M = \{m_0, m_1\}$ where $m_0$ and $m_1$ represent machine 0 and machine 1, respectively. Lastly, the function $\mathcal{P} : O \times M \rightarrow \mathbb{N}_0^+$, specified in Table 2, gives the duration of an operation on the provided machine.

| Operation | Duration $m_0$ (s) | Duration $m_1$ (s) |
|---|---|---|
| $o_0$ | 40 | 50 |
| $o_1$ | 20 | 30 |
| $o_2$ | 30 | 40 |
| $o_3$ | 20 | 70 |
| $o_4$ | 10 | 70 |

**Table 2:** Operation durations on each machine.

A schedule $\mathcal{S}, \alpha$ defined by complete functions $\mathcal{S} : O \rightarrow \mathbb{R}_0^+$ and $\alpha : O \rightarrow M$ describes the start times and machine allocation for all operations respectively. The function $\mathcal{C} : O \rightarrow \mathbb{R}_0^+$ denotes the completion times of all operations and is derived directly from decision variable $\mathcal{S}$. A *feasible* schedule is defined by:

**C1.** Every machine can execute at most one operation at a time:

$$\forall o_x, o_y \in O : (o_x = o_y) \vee \alpha(o_x) = \alpha(o_y) \Rightarrow (\mathcal{C}(o_x) \leq \mathcal{S}(o_y) \vee \mathcal{C}(o_y) \leq \mathcal{S}(o_x)).$$

**C2.** The completion time of an operation is its start time plus its duration.

$$\forall o \in O : \mathcal{C}(o) = \mathcal{S}(o) + \mathcal{P}(o, \alpha(o))$$

The makespan of a schedule $\mathcal{S}, \alpha$ is the latest completion time of any operation:

$$C_{max}(\mathcal{S}, \alpha) = \max_{o \in O}(\mathcal{C}(o))$$

An optimal solution for this simple scheduling problem is a feasible schedule $\mathcal{S}, \alpha$ with the smallest makespan.

## 3.3 | Optimal Job Scheduling Problems

Optimal job scheduling is a category of optimization problems—often related to FMSs—in the scheduling domain. These problems usually require a list of jobs or products to be produced by a defined number of operations on a set of machines. In the literature, a couple of basic job scheduling problems are studied. However, in practice, their exact interpretations may vary. This section discusses the definitions used in this research.

**Definition 3.3.1** (Job)**.** A job $j$ is a partially ordered set $O_j = \{o_{j,1}, \ldots, o_{j,r_j}\}$ of operations, where $r_j$ indicates the job size, with partial order $\preceq_j \; \subseteq O_j^2$ defining precedence relations among the operations.

**Definition 3.3.2** (Shop)**.** A shop $M$ describes the set of available machines, each of which can work on one operation at a time from any job.

**Definition 3.3.3** (Machine Allocation)**.** A machine allocation for job $j$ is a complete function $\alpha_j : O_j \rightarrow M$ that allocates on which machine an operation is scheduled.

Optimal job scheduling problems may vary as different constraints may apply. The literature identifies the following basic job scheduling instances:

- **Open Shop:** Every job $j$ contains the same number of operations $r$, $M$ is known, for all $j \in J$, $\alpha_j$ is specified, and only one machine can work on a given job at the same time. Operations do not have precedence relations, so $\preceq_j = \{(o, o) \mid o \in O_j\}$.
- **Flow Shop:** Every job $j$ contains the same number of operations $r$, $M$ is known, and for all $j \in J$, $\alpha_j$ is specified. Furthermore, for all jobs $j_1$ and $j_2$ and all operations $k \in \{1, \ldots, r\}$, $\alpha_{j_1}(o_{j_1,k}) = \alpha_{j_2}(o_{j_2,k})$. Moreover, the operations are totally ordered, i.e., $\preceq_j$ is a total order. Without loss of generality, we may assume that $o_{j,k_1} \preceq o_{j,k_2}$ if and only if $k_1 \leq k_2$.

- **Job Shop:** Every job $j$ contains $r_j$ operations, $M$ is known, and for all $j \in J$, $\alpha_j$ is specified. As for the flow shop, the operations are totally ordered.
- **Flexible Shop:** Any of the above shop types can be generalized by allowing each operation to be performed on a set of machines. Hence, the $\alpha_j$ need to be determined as part of the solution. In some studies, the term *Hybrid* is used instead of *Flexible*.

These job scheduling problems often come with different additional characteristics. For example, one may consider job/operation deadlines, release times, re-entrance, transportation times, and/or buffer capacity in their problem definition.

The simple scheduling problem from Section 3.2 is a *Flexible Job Shop* with five jobs with a single operation each. All operations can be executed on both machines with varying durations.

## 3.4 | Mathematical Programming

The exact solving of optimization problems such as the job optimization problems shown earlier requires an effective modeling and solving technique. Mathematical Programming (MP) allows us to capture such a problem and its constraints. An MP solver can then be used to find the best solution from a set of feasible alternatives, given constraints and an objective function. MP is commonly used across various domains like production planning, event scheduling, and vehicle routing. MP is often referred to as a collection of similar optimization programming techniques such as the following.

**Linear Programming:** Linear Programming (LP) involves optimizing a linear objective function subject to linear constraints. A linear program comes in the form:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} \quad & \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}_0^n \end{aligned} \tag{3.1}$$

where $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}_0^n$ are vectors and $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a matrix specifying the linear constraints. The shown linear program describes a minimization objective function with decision variable $\boldsymbol{x}$, subject to linear and domain constraints.

**Integer Linear Programming** Integer Linear Programming (ILP) is a form of linear programming that restricts all decision variables to integer values, making it suitable for discrete choices in scheduling, routing, and assignment problems. An integer linear program comes in the form:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} \quad & \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{Z}_0^n \end{aligned} \tag{3.2}$$

where $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{Z}_0^n$ are vectors and $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a matrix specifying the linear constraints. A special kind of ILP uses, instead of integer decision variable domains, boolean decision variable domains as $\mathbb{B} = \mathbb{N}_0^{\leq 1}$.

**Mixed-Integer Linear Programming** Mixed-Integer Linear Programming (MILP) combines integer and continuous variables, resulting in an optimization problem of the form:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d}^T\boldsymbol{y} \\ \text{subject to} \quad & \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{y} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{Z}_0^n, \boldsymbol{y} \in \mathbb{R}_0^{n'} \end{aligned} \tag{3.3}$$

where $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n, \boldsymbol{d} \in \mathbb{R}^{n'}, \boldsymbol{x} \in \mathbb{Z}_0^n$ and $\boldsymbol{y} \in \mathbb{R}_0^{n'}$ are vectors and $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{m \times n'}$ are matrices specifying the linear constraints.

There are solvers available that allow us to find the optimal objective values for any MP program. This study uses the IBM ILOG CPLEX Optimizer solver together with the IBM Decision Optimization CPLEX Modeling (DOcplex) Python API for the development of mathematical programs. For any practical mathematical programming solver, we need to restrict the input variables for the optimization problem to the rational number domain $\mathbb{Q}$.

**Example 3.4.1.**

Consider the simple scheduling problem introduced in Section 3.2. We can capture this problem as an MP problem with decision variables:

| | |
|---|---|
| *Variables* | |
| $K$ | A sufficiently large number. |
| *Decision Variables* | |
| $C_{max} \in \mathbb{R}_0^+$ | The makespan. |
| $s_o \in \mathbb{R}_0^+$ | The starting times of each operation $o \in O$. |

$$x_{o,m} = \begin{cases} 1 & \text{if } o \in O \text{ is allocated on machine } m \in M \\ 0 & \text{otherwise} \end{cases}$$

$$z_{o_1,o_2,m} = \begin{cases} 1 & \text{if } o_1 \in O \text{ is scheduled before } o_2 \in O \text{ on machine } m \in M \\ 0 & \text{otherwise} \end{cases}$$

Observe that the problem at hand is an MILP problem as we both consider continuous decision variables $C_{max}, s_o$ and boolean decision variables $x_{o,m}, z_{o_1,o_2,m}$. Let $\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{z}$ denote the vector representations of $s_o, x_{o,m}, z_{o_1,o_2,m}$. Then, we can formulate the MILP model as:

$$\min_{C_{max}, \boldsymbol{s}, \boldsymbol{x}, \boldsymbol{z}} \quad C_{max} \tag{3.4}$$

$$\text{subject to} \sum_{m \in M} x_{o,m} = 1, \qquad \forall o \in O \tag{3.5}$$

$$s_{o_1} + \mathcal{P}(o_1, m) \leq s_{o_2} + K(1 - z_{o_1,o_2,m}), \qquad \forall m \in M, \forall o_1, o_2 \in O, o_1 \neq o_2 \tag{3.6}$$

$$z_{o_1,o_2,m} + z_{o_2,o_1,m} = 1, \qquad \forall m \in M, \forall o_1, o_2 \in O, o_1 \neq o_2 \tag{3.7}$$

$$s_o + \sum_{m \in M} x_{o,m} \cdot \mathcal{P}(o, m) \leq C_{max}, \qquad \forall o \in O \tag{3.8}$$

Equation 3.4 is the objective function to minimize, the makespan. Equation 3.5 ensures that all operations are allocated to a machine. Equation 3.6 enforces that every machine can only execute at most one operation at a time. Equation 3.7 ensures that operations are sequenced correctly. Finally, Equation 3.8 specifies the makespan $C_{max}$ as the latest completion time of any operation.

Using IBM's CPLEX Optimizer solver, we find the schedule seen in Figure 1 with a makespan of $C_{max} = 70$.

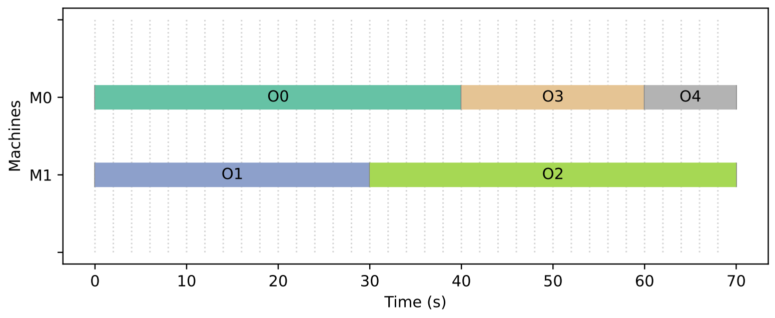


**Figure 1:** Optimal schedule for the simple scheduling example.

## 3.5 | Logic-Based Benders Decomposition

As some optimization problems are very complex in terms of the number of decision variables and constraints, they become very hard to solve. In these cases, another strategy is needed to find solutions to these problems efficiently. One of the best-known and successful strategies for finding solutions to these hard optimization problems is the *Benders Decomposition* [3].

The general idea of the Benders decomposition is to fixate a part of the decision variable value assignments *(trial values)* in such a way the remaining problem (subproblem) becomes easier to solve. Smart evaluation of this subproblem's solution allows us to select a new, better set of trial values for a new iteration of the procedure. After a couple of iterations, the procedure converges towards finding the optimal solution to the overall problem. All in all, the overall idea is that each iteration of the procedure learns from previously made mistakes.

The first Benders decomposition was introduced in 1962 by Benders [3]. This first variant—often referred to as the *classical Benders decomposition*—proved very successful in decomposing and solving hard linear programming problems. Later, Hooker and Ottosson [20] introduced the Logic-Based Benders Decomposition (LBBD), a generalized Benders Decomposition allowing any kind of optimization problem. In this research, we often refer to LBBD as simply *Benders decomposition*.

As a result, the LBBD allows for solving different decomposed problems using different methods. This can be a huge advantage for complex optimization problems, as we can tailor the solving techniques that best suit the decomposed problem's structure.

### Principles

As the Benders decomposition decomposes a hard optimization problem into two or multiple smaller optimization problems it is useful to introduce proper terminology. Within this research, the optimization problem that is desired to be solved using a Benders decomposition is referred to as the *overall problem*. In the Benders decomposition procedure, some of the decision variable assignments are fixated. However, these chosen trial values cannot be completely arbitrary as the overall problem specifies constraints related to these decision variables. Hence, the chosen trial values have to comply with their related constraints from the overall problem. Therefore, the optimization problem related to finding the trial values can be seen as the *master problem*. The remaining optimization problem that remains after the trial values are assigned is called the *subproblem*.

The power of the Benders decomposition is primarily related to the so-called *Benders cuts*. Benders cuts are constraints added in each iteration of the Benders decomposition to the master problem. In each iteration of the Benders decomposition, a subproblem solution is found for a given set of new trial values. However, there is no guarantee that the solutions found at that point will be part of the optimal solution to the overall problem. Here is where the Benders cut comes into play. The goal of these Benders cuts is to guide the decision of the next set of trial values towards the optimal solution of the overall problem by using information on the found subproblem solution. When the optimal master problem objective value and the best optimal subproblem objective value obtained at that point converge to the same value, the process stops, and the optimal solution of the overall problem can be concluded.

To highlight the differentiation between the master and subproblems and their interaction. Figure 2 illustrates the complete procedure using the introduced terminology.

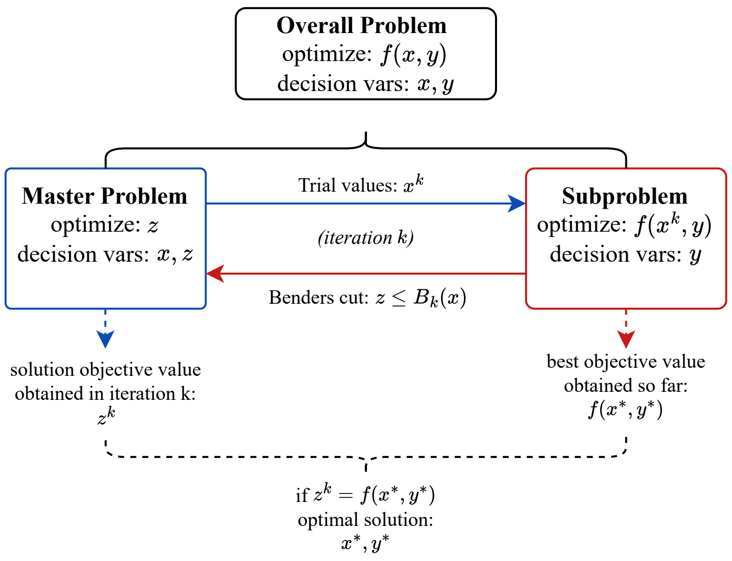


**Figure 2:** Principles of a Benders decomposition.

## 3.5.1 | The Benders Decomposition

Using the principles of the Benders decomposition and the definitions from [20], [42], and [21], let us define a generic Benders decomposition.

Consider an overall optimization problem where the decision variables are partitioned into two vectors of decision variables $\boldsymbol{x}, \boldsymbol{y}$ with domains $D_{\boldsymbol{x}}$ and $D_{\boldsymbol{y}}$ and objective function $f : D_{\boldsymbol{x}} \times D_{\boldsymbol{y}} \to \mathbb{R}_0^+$:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & f(\boldsymbol{x}, \boldsymbol{y}) \\ \text{s.t.} \quad & C(\boldsymbol{x}, \boldsymbol{y}) \\ & C(\boldsymbol{x}),\ C(\boldsymbol{y}) \\ & \boldsymbol{x} \in D_{\boldsymbol{x}},\ \boldsymbol{y} \in D_{\boldsymbol{y}} \end{aligned} \tag{3.9}$$

$C(\boldsymbol{x})$, $C(\boldsymbol{y})$, and $C(\boldsymbol{x}, \boldsymbol{y})$ denote the sets of constraints considering decision variables $\boldsymbol{x}$, $\boldsymbol{y}$, and both $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively.

Since the decision variables are partitioned into two vectors, we can easily apply a Benders decomposition and decompose the problem into a master and subproblem. Let us first fixate the values of $\boldsymbol{x}$ to obtain the subproblem:

$$v^k = \left\{ \begin{aligned} \min_{\boldsymbol{y}} \quad & f(\boldsymbol{x}^k, \boldsymbol{y}) \\ \text{s.t.} \quad & C(\boldsymbol{x}^k, \boldsymbol{y}) \\ & C(\boldsymbol{y}) \\ & \boldsymbol{y} \in D_{\boldsymbol{y}} \end{aligned} \right\} \tag{3.10}$$

Here, $\boldsymbol{x}^k$ denotes the trial values of $\boldsymbol{x}$, and $v^k$ denotes the optimal solution at iteration $k$ of the Benders decomposition. The subproblem only considers $\boldsymbol{y}$ as decision variables. Hence, constraint set $C(\boldsymbol{y})$ is part of the problem. Note that constraint set $C(\boldsymbol{x}, \boldsymbol{y})$ of Equation 3.9 is also part of the subproblem but uses the trial values $\boldsymbol{x}^k$.

The subproblem generates a Benders cut of the form $z \geq B^k(\boldsymbol{x})$ where $B_k(\boldsymbol{x}) : D_{\boldsymbol{x}} \to \mathbb{R}_0^+$ is a function that results from the subproblem solution at iteration $k$. The purpose of the constraint value $B_k(\boldsymbol{x})$ is to provide a lower bound on a variable $z \in \mathbb{R}_0^+$ that is minimized by the master problem:

$$z^k = \left\{ \begin{array}{lll} \min\limits_{\boldsymbol{x}, z} & z & \\ \text{s.t.} & z \geq B^i(\boldsymbol{x}), & \forall i \in \{1, \ldots, k-1\} \\ & C(\boldsymbol{x}) & \\ & \boldsymbol{x} \in D_{\boldsymbol{x}} & \end{array} \right\} \tag{3.11}$$

$z^k$ denotes the optimal solution to the master problem at iteration $k$ of the Benders decomposition. The constraints for the master problem are defined by the constraint set $C(\boldsymbol{x})$ and the Benders cuts obtained so far. In each iteration of the Benders decomposition, more cuts are added, thus increasing the constraints in the master problem.

For the Benders cuts to qualify as a valid cut two properties must hold:

**BCP1.** $B^k(\boldsymbol{x})$ must be a valid lower bound on $f(\boldsymbol{x}, \boldsymbol{y})$ of the overall problem. Hence, $B^k(\boldsymbol{x}) \leq f(\boldsymbol{x}, \boldsymbol{y})$ must hold for any feasible $(\boldsymbol{x}, \boldsymbol{y})$.

**BCP2.** By instantiating the cut $B^k(\boldsymbol{x})$ with trial values $\boldsymbol{x}^k$, the optimal $v^k$ will result. Hence, $B^k(\boldsymbol{x}^k) = v^k$.

The optimal solution to the overall problem is found when $z^k$ results in the same value as the best obtained $v^k$ at that point $v^*$. Algorithm 1 captures the entire Benders decomposition process.

**Algorithm 1:** Logic-Based Benders Decomposition Algorithm for Optimization Subproblems [21]

```
$k \leftarrow 0$;
$v^* \leftarrow \infty$;
repeat
    $k \leftarrow k + 1$;
    solve Equation 3.11 to obtain $z^k$;
    if $z^k < \infty$ then
        let $\boldsymbol{x}^k$ be trial values for decision variables $\boldsymbol{x}$;
        solve Equation 3.10 for $f(\boldsymbol{x}^k, \boldsymbol{y})$;
        if $v^k < v^*$ then
            $v^* \leftarrow v^k$;
            $x^* \leftarrow x^k$;
            let $y^*$ be the values for decision variables $\boldsymbol{y}$;
        end
        if $z^k < v^*$ then
            generate Benders cut $z \geq B^k(\boldsymbol{x})$ according to Properties BCP1. and BCP2.;
        end
    end
until $z^k = v^*$;
if $z^k < \infty$ then
    $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ gives the optimal solution to Equation 3.9
else
    Equation 3.9 is infeasible;
end
```

### 3.5.2 | Benders Cuts and Duality

As mentioned, the power of the Benders decomposition is controlled by the Benders cuts. Defining these cuts can be very challenging, as many different trial values may result in the same or a similar subproblem solution. However, the goal is to define cuts that exclude a *large* part of the possible trial values. Such cuts will lead to a faster convergence towards an optimal solution.

In the classical Benders decomposition, the Benders cuts are directly derived from the linear programming dual. This is a very useful property as it is a very straightforward method for finding good cuts. However, if the overall problem is not a linear programming problem, another method must be found.

Hooker and Ottonson [20] determined that the linear programming dual is a special case of the inference dual which can be logically deduced from its constraint set. From this observation, it becomes possible to formulate Benders cuts based on logical inference for all optimization problems. However, while formulating a classical Benders cut is fairly straightforward, a logic-based Benders cut can be quite complex as it requires some analysis and creativity for every type of problem.

### 3.5.3 | Simple Scheduling Example

Using the information obtained so far, let us take a look at the scheduling example from Section 3.2 (overall problem) and solve it using the Logic-Based Benders Decomposition.

**Example 3.5.1.**

The problem contains two sets of decision variables: $\mathcal{S}$ (scheduling of operations) and $\alpha$ (machine allocation of operations). Hence, we can decompose the problem into a master problem that performs the machine allocation and a subproblem dealing with the scheduling of the operations. This leads to the resulting master problem:

$$\begin{aligned} \min_{\alpha,z} \quad & z \\ \text{s.t.} \quad & z \geq B^i(\alpha), \quad \forall i \in \{1,\dots,k-1\} \\ & \alpha : O \to M \end{aligned} \tag{3.12}$$

Since our simple scheduling problem does not define constraints exclusively for $\alpha$. There does not exist a set of constraints $C(\alpha)$.

From here, the sub-problem is defined where $\alpha^k$ denotes the allocation at iteration $k$ of the Benders decomposition:

$$\begin{aligned} \min_{\mathcal{S}} \quad & C_{max}(\mathcal{S},\alpha^k) \\ \text{s.t.} \quad & \mathcal{S} : O \to \mathbb{R}_0^+ \\ & C(\mathcal{S},\alpha^k) := \{\forall o_x, o_y \in O : (o_x = o_y) \vee \alpha^k(o_x) = \alpha^k(o_y) \Rightarrow (\mathcal{C}(o_x) \leq \mathcal{S}(o_y) \vee \mathcal{C}(o_y) \leq \mathcal{S}(o_x)), \\ & \qquad \forall o \in O : \mathcal{C}(o) = \mathcal{S}(o) + \mathcal{P}(o,\alpha^k(o))\} \end{aligned} \tag{3.13}$$

A simple cut can be created that reports a bound on the makespan $C_{max}$ for each allocation the master problem finds, as illustrated next.

**Iteration 1:** Solving the master problem provides an allocation for all operations. Since there are no constraints on alpha and no cuts (yet), the solver has absolute freedom in assigning this first allocation $\alpha^1$. Let us assume the allocation to be: $\alpha^1(o_0) = 0$, $\alpha^1(o_1) = 1$, $\alpha^1(o_2) = 0$, $\alpha^1(o_3) = 0$, $\alpha^1(o_4) = 0$.

Solving the resulting master problem gives objective value $z^1 = 0$ as there are no constraints bounding the value of $z$. Solving the subproblem for $\alpha^1$ yields a makespan and subproblem objective value of $v^1 = 100$ as the subproblem has multiple constraints regarding $\mathcal{S}$ and $\alpha^1$. The resulting schedule can be seen in Figure 3.

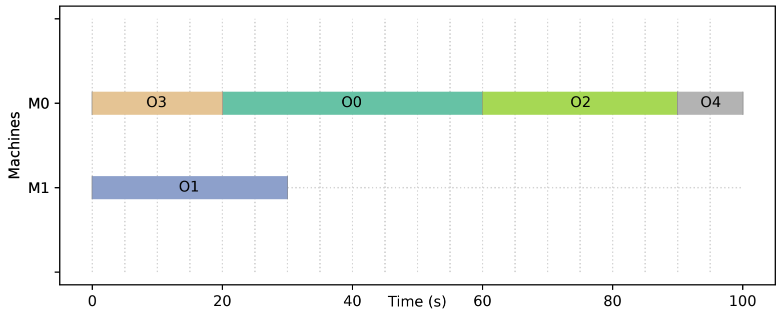


**Figure 3:** Subproblem solution in iteration 1.

These results show that as long as the master problem chooses $\alpha^1$ as trial values, the subproblem objective value will always result in makespan $v^k = 100$. Hence, we need the master problem to become aware that choosing $\alpha^1$ results in a makespan of $v^k = 100$. In other words, we must ensure that $\alpha^1$ results in $z \geq 100$ in future iterations. Hence, we add a simple cut $z \geq B^1(\alpha)$ that adheres to the properties **BCP1.** and **BCP2.** to make sure that a possible future assignment of $\alpha^1$ also leads to $z \geq 100$. Here, $B^1(\alpha)$ can be defined as:

$$B^1(\alpha) := \alpha(o_0) = 0 \wedge \alpha(o_1) = 1 \wedge \alpha(o_2) = 0 \wedge \alpha(o_3) = 0 \wedge \alpha(o_4) = 0 \implies 100$$

This cut is simple in the sense that it is only reporting a bound on $z$ for this and only this allocation. It is clear that it satisfies both **BCP1.** and **BCP2.**. Observe that the cut can be strengthened by removing $\alpha(o_1) = 1$ as it does not have an impact on the makespan as seen in Figure 3. For the strengthened cut $z \geq B^1(\alpha)$ we need to adjust $B^1(\alpha)$ accordingly:

$$B^1(\alpha) := \alpha(o_0) = 0 \wedge \alpha(o_2) = 0 \wedge \alpha(o_3) = 0 \wedge \alpha(o_4) = 0 \implies 100$$

The resulting cut leaves the allocation of $o_1$ free while still providing a valid lower bound on the makespan. As the new cut applies to more than one specific assignment of $\alpha$, it is considered stronger than the previous cut.

**Iteration k:** After the first iteration, the algorithm continues for a while. Let us assume it provides the results seen in Table 3. Observe that since we added the cut $z \geq B^1(\alpha)$, the allocation $\alpha^1$ is avoided, resulting in a new allocation that results in a master problem objective value $z^2 = 0$ and subproblem objective value $v^2 = 70$. Additionally, we can see that the best found objective value $v^*$ at this point now becomes $v^* = 70$. The solution $(\mathcal{S}^*, \alpha^*)$ that results in $v^*$ is defined by $\alpha^2$, and the resulting $\mathcal{S}$ from iteration 2.

While the algorithm continues, more Benders cuts $z \geq B^k(\alpha)$ are added, expanding the lower bound on $z$. After several iterations, the minimal $z$ that can be obtained becomes equal to the best-found subproblem solution $v^*$. In Table 3, we see this in iteration $k$. Here, the algorithm terminates and concludes that the optimal objective value to the overall problem of Section 3.2 is $C_{max}(\mathcal{S}^*, \alpha^*) = v^* = 70$, which is the same as we found with the MILP approach in Section 3.4. The allocation and start times that provide this result were found in iteration 2 and can be seen in Figure 4. Compared to the schedule of Figure 1, operations $o_0$ and $o_3$ swapped order. This illustrates that the scheduling problem has multiple optimal solutions.

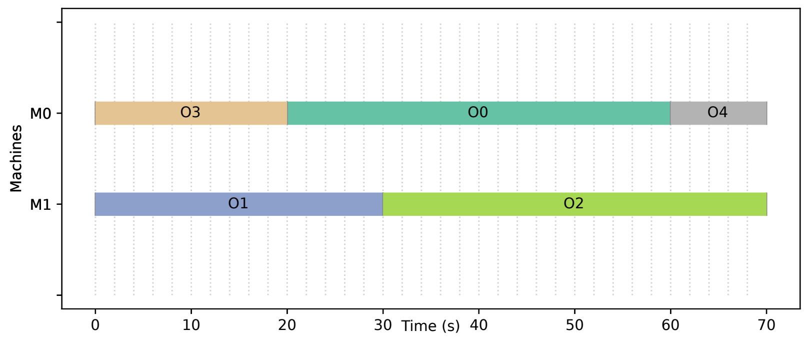


**Figure 4:** Solution to the overall problem and iteration 2 of the procedure.

| Iteration | $z^k$ (master problem objective value) | $\alpha^k$ $(\alpha(o_0), \alpha(o_1), \alpha(o_2), \alpha(o_3), \alpha(o_4))$ (trial values) | $v^k$ (subproblem objective value) | $v^*$ |
|---|---|---|---|---|
| 1 | 0 | $(0, 1, 0, 0, 0)$ | 100 | 100 |
| **2** | **0** | $\mathbf{(0, 1, 1, 0, 0)}$ | **70** | 70 |
| ... | ... | ... | ... | 70 |
| $k$ | 70 | $(1, 1, 0, 0, 0)$ | 80 | 70 |

**Table 3:** Example results from Benders decomposition algorithm.

## 3.5.4 | LBBD for feasibility subproblems

In some cases—such as the problems studied in this research—decomposing a complex optimization problem results in a feasibility subproblem. Such a subproblem does not have an objective function and merely checks whether a certain set of trial values provides a feasible solution to the overall problem. If the subproblem provides this feasible solution, the algorithm terminates and concludes that the last obtained master problem solution and subproblem solution form the optimal solution of the overall problem.

The Benders cuts that this feasibility subproblem should provide cannot give a valid lower bound on the master problem's objective function but instead try to exclude the generation of infeasible trial values. This results in a feasibility cut $F^k(\boldsymbol{x}) : D_{\boldsymbol{x}} \to \mathbb{B}$. The properties for these feasibility cuts differ from those presented earlier:

**FCP1.** Providing trial values $\boldsymbol{x}^k$ to the cut $F^k(\boldsymbol{x})$ results in infeasibility. Hence, $F^k(\boldsymbol{x}^k) \equiv \texttt{False}$.

Since the subproblem has no objective function, the overall problem's objective function only depends on the master problem's decision variables:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & C(\boldsymbol{x},\boldsymbol{y}) \\ & C(\boldsymbol{x}),\, C(\boldsymbol{y}) \\ & \boldsymbol{x} \in D_{\boldsymbol{x}},\, \boldsymbol{y} \in D_{\boldsymbol{y}} \end{aligned} \tag{3.14}$$

The resulting master problem and subproblem are then of the form:

$$z^k = \left\{ \begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & F_i(\boldsymbol{x}), \quad \forall i \in \{1, \ldots, k-1\} \\ & C(\boldsymbol{x}) \\ & \boldsymbol{x} \in D_{\boldsymbol{x}} \end{aligned} \right\} \tag{3.15}$$

$$v^k = \left\{ \begin{array}{ll} \min\limits_{\boldsymbol{y}} & 0 \\ \text{s.t.} & C(\boldsymbol{x}^k, \boldsymbol{y}) \\ & C(\boldsymbol{y}) \\ & \boldsymbol{y} \in D_{\boldsymbol{y}} \end{array} \right\} \tag{3.16}$$

The algorithm also changes as the procedure terminates once a feasible subproblem solution is found:

**Algorithm 2:** Logic-Based Benders Decomposition Algorithm for Feasibility Subproblems [21]

```
$k \leftarrow 0$;
repeat
    $k \leftarrow k + 1$;
    solve Equation 3.15 to obtain $z^k$;
    if $z^k < \infty$ then
        let $\boldsymbol{x}^k$ be trial values for decision variables $\boldsymbol{x}$;
        solve Equation 3.16 for $\boldsymbol{x}^k$ resulting in solution $\boldsymbol{y}^k$;
        if Equation 3.16 is infeasible then
            generate Benders cut $F^k(\boldsymbol{x})$ according to Property FCP1.;
        end
    end
until Equation 3.16 is feasible or Equation 3.15 is infeasible;
if Equation 3.16 is feasible then
    $(\boldsymbol{x^k}, \boldsymbol{y^k})$ gives the optimal solution to Equation 3.14
else
    Equation 3.14 is infeasible;
end
```

## 3.6 | Constraint Programming

While MP is a commonly used approach for the exact solving of various optimization problems, it may not always perform best for certain types of optimization problems. Constraint Programming (CP) is a modeling and analysis approach for finding feasible and optimal solutions to combinatorial problems and was shown to be a very successful mechanism when applied to scheduling problems [27, 16, 17, 13]. A constraint program specifies a problem definition along with solution constraints in a syntax close to natural language. Without specifying how to find a solution, a constraint solver searches for a solution that satisfies all the constraints. To explore the solution space, techniques such as constraint propagation, pruning, and backtracking are used.

Building a CP model takes several elements: decision variables, constraints, and optionally an objective function. Decision variables are all problem variables for which the solver needs to find a solution. In scheduling problems, these variables often are the start/end times of operations and the allocation of tasks to machines. Constraints, on the other hand, are (in)equalities between—possibly, multiple—decision variables and/or constants. Lastly, an objective function might be presented to solve a specific optimization problem.

There are various tools available to develop and run CP models. Their modeling practices may differ [18]. Often tools present new decision variable types to simplify the modeling effort, or introduce new shorthands for temporal constraints, specifying relationships between events that refer to time. Therefore, some concepts might not be universal to all CP implementations. This study uses the IBM CPLEX CP Optimizer (CP Optimizer) solver together with the DOcplex Python API for the development of constraint programs. Note that for any practical constraint programming implementation to work, we need to restrict the input variables for the optimization problem to the rational number domain $\mathbb{Q}$.

### 3.6.1 | Decision Variables

Besides common integer, boolean, and floating point variables, this research uses interval and sequence variables [27]. Interval variables refer to a time interval of interest in a schedule. These time intervals might have unknown start times, end times, and durations. Furthermore, interval variables may be optional; this means that one may deconsider them in the solution. Sequence variables capture the orderings of given sets of interval variables. In other words, the solver receives a set of interval variables and decides upon an ordering sequence for all these interval variables.

**Example 3.6.1.**

Consider the scheduling problem introduced in Section 3.2. The constraint program for this problem needs to make the following decisions:

- Operation intervals: start times of each operation.
- Operation allocation: mapping of an operation to a machine

The constraint problem then consists of several decision variables:

- Mandatory interval variable for each operation $o \in O$ with no assigned duration.

```
operations = {o: interval_var() for o in range(5)}
```

- For each operation $o \in O$, we need an optional interval variable for each possible assignment to a machine $m \in M$. The duration of each of these `machine_options` was specified in Section 3.2 and is represented in the constraint program as `DURATIONS[o][m]`. An optional interval variable may or may not exist and if it exists it must have size `DURATIONS[o][m]`.

```
machine_options = {
    (o, m): interval_var(optional=True, size=DURATIONS[o][m])
    for o in range(5) for m in range(2)
}
```

- List of sequence variables where each entry contains the sequence of all `machine_options` interval variables mapped on the particular machine.

```
machine_sequences = [
    sequence_var(
        [machine_options[(o, m)] for o in range(5)]
    ) for m in range(2)
]
```

### 3.6.2 | Temporal Constraints

Building CP models for scheduling problems incorporates defining temporal constraints. These constraints specify relations between interval and sequence variables, and their timing. To enhance simplicity, specifying these temporal constraints may use various predefined global functions [27]:

- `no_overlap`: Constrains that interval variables within a sequence variable cannot overlap in time.

- `start_of`/`end_of`/`size_of`: Allows the creation of constraints related to the start, end, and duration of an interval variable, respectively.
- `alternative`: States a list of optional interval variables with alternative properties and a mandatory interval variable without properties. `alternative` selects one and only one optional interval variable to be present. The mandatory interval variable must have the same start and end times as the selected optional interval variable. This can be useful for capturing an allocation when the duration of an interval variable depends on the allocated resource.
- `presence_of`: Allows the creation of constraints related to the presence of optional interval variables.
- `type_of_prev`/`type_of_next`: Situations occur where constraints depend on the ordering of interval variables. In this case, a non-negative integer *type* can be assigned to each interval variable in the sequence. The `type_of_prev` makes it possible to make a constraint based on the type of the preceding interval variable in a sequence. Additionally, the `type_of_next` makes it possible to create a constraint based on the succeeding interval variable in a sequence.

Constraints are constructed using a functional language. Logical connectives such as implications and conjunctions are represented as `if_then()` and `logical_and()` functions, respectively.

**Example Continued.**

Consider the scheduling problem and decision variables in Section 3.2. The constraints given earlier in Section 3.2 can be specified in the CP model as follows:

- Every machine can execute at most one operation at a time. Hence, no overlap may exist within the same sequence.

  For all machines $m$:

  ```
  no_overlap(machine_sequences[m])
  ```

- Every operation $o$ can be executed on either $m_0$ or $m_1$. Hence, for each operation, two alternative durations exist.

  For all operations $o$:

  ```
  alternative(
      operations[o],
      [machine_options[a] for a in machine_options if a[0] == o]
  )
  ```

To illustrate some of the other CP concepts, we introduce some additional constraints. For instance, the start time of a specific operation can be constrained:

- The start of operation $o_4$ is at $t = 0$.

  ```
  start_of(operations[4]) == 0
  ```

Additionally, we may assign a type to each interval variable `machine_options` corresponding to its operation number. To specify the successor of operation $o_4$ to be $o_1$, we can then introduce the following constraint:

- The operation $o_4$ should always be succeeded by $o_1$ if both are allocated on machine 0.

```
type_of_next(machine_sequences[0], machine_options[4,0]) = 1
```

### 3.6.3 | Transition Matrices

A powerful CP concept is the use of *transition matrices* to introduce transition times between interval variables. When two consecutive interval variables (interval 1, interval 2) cannot start right after one another, a transition time specifies the time it takes to transition from interval 1 to interval 2.

Consider, for example, a machine shop with a milling machine. Operation 1 enters the milling machine, the milling machine performs its task, and finally, operation 1 is completed. Next in line is operation 2. However, operation 2 needs a different milling cutter. In this case, the milling machine has to change the cutter, which takes up some additional time, a transition time.

The above-illustrated example shows a case of *sequence-dependent setup times*. Such sequence-dependent setup times are a common modeling problem in FMS scheduling. We can specify transition times in a *transition matrix*. This matrix holds all the transition times between two types of interval variables. By providing a transition matrix as a parameter to the `no_overlap` constraint, we can easily introduce these sequence-dependent setup times.

**Example Continued.**

Let us continue with our running example, Example 3.6.1, and illustrate the use of transition matrices. We want to ensure that there is at least a setup time of 5 time units after each operation. We can introduce a transition matrix $\mathbf{T}$, where each row represents an operation, and each column represents the next operation.

$$\mathbf{T} = \begin{bmatrix} 0 & 5 & 5 & 5 & 5 \\ 5 & 0 & 5 & 5 & 5 \\ 5 & 5 & 0 & 5 & 5 \\ 5 & 5 & 5 & 0 & 5 \\ 5 & 5 & 5 & 5 & 0 \end{bmatrix}$$

We need to change the `no_overlap` constraint presented in the previous section accordingly:

For all machines $m$:

```
no_overlap(machine_sequences[m], T)
```

### 3.6.4 | Objective function

In most scheduling applications, the goal is to find the shortest schedule. In this case, an optimization function can be included in the CP model.

**Example Continued.**

Consider the scheduling problem in Section 3.2. The shortest schedule can be obtained by an objective function that `minimizes` the latest end time of any operation (i.e., the makespan).

```
minimize(max(end_of(operations[o]) for o in operations))
```

Running the entire example CP model developed throughout this section provides a schedule seen in Figure 5.

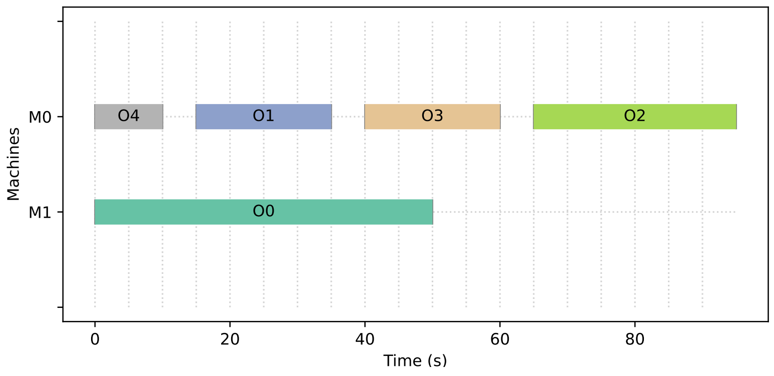


**Figure 5:** Optimal solution to the simple scheduling problem introduced in Section 3.2.

We can see from the figure that, indeed operation 4 starts at time $t = 0$ and is directly followed by operation 1 on machine 0. Furthermore, observe that the transition matrix ensured there is always a 5-second setup time. Verifying the optimality of this schedule is an interesting exercise for the reader.

# Chapter 4

# Transportation-Constrained Flexible Job-Shop Scheduling

To create a decomposition-based solution for the CFTFJSSP, we need a good understanding and definition of its foundational optimization problems. One of these optimization problems is the Transportation-constrained Flexible Job-Shop Scheduling Problem (TFJSSP), combining the flexible job-shop scheduling problem with internal transportation. In this chapter, we introduce the motivation for this specific Transportation-constrained Job-Scheduling Problem (TJSP), a definition for the TFJSSP, and a constraint programming model for the TFJSSP, which we use later as part of our decomposition solution for the CFTFJSSP.

## 4.1 | Introduction

The introduced optimal job-scheduling problems so far only focus on operations being allocated and scheduled on a set of machines. However, in reality, these machines are often not directly connected but are distributed at several places inside a factory or machine shop. If we consider a job with multiple operations, when an operation finishes a partial product, this semi-finished product needs to emphtransfer to the next machine for its succeeding operation. In many traditional optimal job-scheduling problems, these *transfer times* are constrained as sequence-dependent setup times or not considered at all. While this is sufficient for some applications, it forms a bottleneck when vehicles (AGVs) are introduced for transportation of materials and semi-finished products. When the number of vehicles is limited, we need to distribute the different transfer tasks among these vehicles. Hence, a vehicle might not be available at all times. This is where a TJSP comes in place. A TJSP considers a traditional optimal job-scheduling problem and extends its definition by incorporating a set of vehicles that need to make sure all results of operations are transferred to the correct location. This brings up some additional constraints that deal with vehicle allocation and guarantee correct transfer times.

## 4.2 | State of the Art

There are many perspectives when it comes to defining TJSPs. A major contribution came from Bilge and Ulusoy [4] in 1995, who described a generic TJSSP with a makespan objective and a benchmark consisting of four FMS layouts and 10 job sets to verify algorithmic performance in solving these problems. Many academics have proposed heuristic methods to solve these benchmark problems effectively [4, 47, 1, 14, 41, 51, 2, 17].

Bilge and Ulusoy had to introduce new terms to extend the traditional JSSP. Where traditional JSSPs only include machines, the TJSSP should cover the entire FMS layout. Firstly, vehicles—often AGVs—are introduced to transport (partial) products between machines. Furthermore, a general loading/unloading (L/U) station is defined as a location where all vehicles and starting materials are located at the start of the schedule. Secondly, the notion of a trip was defined. A trip is a movement task of a vehicle. If a vehicle needs to pick up a product, it has no payload. Bilge and Ulusoy define such trips as deadheading trips. Trips that include a payload are called loaded trips. The layout of the FMS is defined by a travel time matrix which denotes the relative positioning of all machines, the loading and unloading stations.

The exact terminology used by Bilge and Ulusoy is not consistently followed in the literature, as the practical background problems for each instance might differ. In this research, we name a vehicle movement task a transfer. Deadheading trips are specified as empty transfers and denote the movement of a vehicle without payload toward any pickup location. Loaded trips are named loaded transfers and denote the transfer to deliver a payload to a drop-off location.

TJSSP research is quite uniformly following and extending the work of Bilge and Ulusoy. Deroussi and Norre [12] adapted the benchmark of Bilge and Ulusoy to make it compatible with TFJSSP. Several solutions were proposed to solve these benchmark instances [50, 11, 35, 19]. Only recently, Ham [16] came up with an *exact* Constraint Programming solution to verify the exact makespan of the benchmark instances.

Besides all these contributions, TFJSSP research is still quite scarce and often fragmented into use-case-specific adaptations. For this research, we define a generic TFJSSP definition that extends the definitions seen for the TJSSP and TFJSSP so far and integrates well with existing CFTFJSSP research, as we see later.

## 4.3 | Illustrative Example of the Transportation-Constrained Flexible Job Shop

To understand what the TFJSSP actually looks like, let us introduce an illustrative example from the benchmark of Deroussi and Norre [10].

Consider an FMS with 9 notable locations: 8 machines $\{m_1, \ldots, m_8\}$ and 1 loading/unloading station $lus$. There are two AGVs $\{v_0, v_1\}$ responsible for transferring partially finished products from one location to another. The transfer times from one location to another are described in Table 4.

| | $lus$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ |
|---|---|---|---|---|---|---|---|---|---|
| $lus$ | 0 | 6 | 8 | 6 | 8 | 10 | 12 | 10 | 12 |
| $m_1$ | 8 | 0 | 2 | 8 | 2 | 4 | 6 | 4 | 6 |
| $m_2$ | 6 | 10 | 0 | 10 | 8 | 2 | 4 | 6 | 4 |
| $m_3$ | 12 | 4 | 6 | 0 | 6 | 8 | 10 | 8 | 10 |
| $m_4$ | 10 | 2 | 4 | 6 | 0 | 6 | 8 | 2 | 8 |
| $m_5$ | 8 | 8 | 2 | 8 | 6 | 0 | 6 | 4 | 2 |
| $m_6$ | 6 | 10 | 8 | 10 | 8 | 6 | 0 | 6 | 4 |
| $m_7$ | 12 | 4 | 6 | 4 | 2 | 8 | 10 | 0 | 10 |
| $m_8$ | 10 | 6 | 4 | 6 | 4 | 2 | 8 | 2 | 0 |

**Table 4:** Transfer times between each pair of locations.

Observe that the transfer times differ for the direction in which the vehicles drive. These are due to travel restrictions in the FMS, as illustrated in Figure 6. The figure shows the travel path for vehicles moving from the loading station to machine 1. This path differs from the path vehicles take when moving from machine 1 to the loading station.

This FMS needs to complete a set of 6 jobs $\{j_0, \ldots, j_5\}$ (job set 2 of [10]) each consisting of up to 5 operations $o_{j,k}$ where $j$ indicates the job number and $k$ indicates the operation number. The first

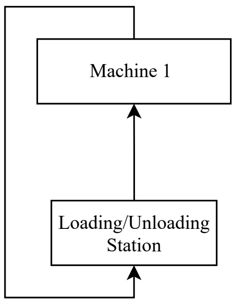


Figure 6: Travel paths between the loading station and machine 1.

operation $o_{j,0}$ of each job marks a loading operation in the loading/unloading station, where a vehicle picks up the starting materials for the job to start. The last operation marks the unloading of the final product in the unloading station. The remaining operations are machine operations to shape the materials and partially finished products into the final product. Note that some of these machine operations can be performed at different locations as we deal with a flexible job shop. An overview of the machine options and processing times of each operation can be seen in Table 5.

| Job | Operation | $lus$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ | $m_8$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $j_0$ | $o_{0,0}$ | 0 | - | - | - | - | - | - | - | - |
| | $o_{0,1}$ | - | 20 | 20 | - | - | - | - | - | - |
| | $o_{0,2}$ | - | - | - | - | - | - | - | 36 | 36 |
| | $o_{0,3}$ | 0 | - | - | - | - | - | - | - | - |
| $j_1$ | $o_{1,0}$ | 0 | - | - | - | - | - | - | - | - |
| | $o_{1,1}$ | - | - | - | 20 | 20 | - | - | - | - |
| | $o_{1,2}$ | - | - | - | - | - | - | - | 36 | 36 |
| | $o_{1,3}$ | 0 | - | - | - | - | - | - | - | - |
| $j_2$ | $o_{2,0}$ | 0 | - | - | - | - | - | - | - | - |
| | $o_{2,1}$ | - | 20 | 20 | - | - | - | - | - | - |
| | $o_{2,2}$ | - | - | - | - | - | 40 | 40 | - | - |
| | $o_{2,3}$ | 0 | - | - | - | - | - | - | - | - |
| $j_3$ | $o_{3,0}$ | 0 | - | - | - | - | - | - | - | - |
| | $o_{3,1}$ | - | - | - | 20 | 20 | - | - | - | - |
| | $o_{3,2}$ | - | - | - | - | - | 30 | 30 | - | - |
| | $o_{3,3}$ | - | - | - | - | - | - | - | 24 | 24 |
| | $o_{3,4}$ | 0 | - | - | - | - | - | - | - | - |
| $j_4$ | $o_{4,0}$ | 0 | - | - | - | - | - | - | - | - |
| | $o_{4,1}$ | - | 20 | 20 | - | - | - | - | - | - |
| | $o_{4,2}$ | - | - | - | 30 | 30 | - | - | - | - |
| | $o_{4,3}$ | - | - | - | - | - | - | - | 24 | 24 |
| | $o_{4,4}$ | 0 | - | - | - | - | - | - | - | - |
| $j_5$ | $o_{5,0}$ | 0 | - | - | - | - | - | - | - | - |
| | $o_{5,1}$ | - | 20 | 20 | - | - | - | - | - | - |
| | $o_{5,2}$ | - | - | - | 30 | 30 | - | - | - | - |
| | $o_{5,3}$ | - | - | - | - | - | 24 | 24 | - | - |
| | $o_{5,4}$ | 0 | - | - | - | - | - | - | - | - |

Table 5: Operation processing times on each machine.

Observe that the above processing times define where each operation can be performed. A - symbol indicates that there is no defined operation duration, and therefore, the operation cannot be performed on the respective machine. Note that only the loading and unloading operations take place in the loading station and are considered to take up no time.

Ideally, we want the FMS to complete all the jobs described above as soon as possible. This turns out to be rather complicated due to several constraints. Machines can only work on one operation at a time, and an operation $o$ can only start when its preceding operation has completed and the resulting partial product has been transferred to the machine where $o$ will be performed. The vehicles performing these transfers are also restricted. Vehicles can only serve one job at a time. With limited vehicles available and many transfer tasks, machines often need to wait for the vehicle to arrive before they can start processing the next operation.

To formalize this problem further, in the next section, we give a definition of the TFJSSP problem and its scheduling constraints.

## 4.4 | Problem Definition

Let us begin by defining the generic transportation-constrained flexible job shop, its schedule, and a TFJSSP with a makespan objective. This research divides the loading/unloading station into two separate stations, a common practice in CFTFJSSP literature. It is worth noting that for specific instances, such as the example in the previous section, the loading and unloading station can still be considered as a single location.

The definition of the transportation-constrained flexible job shop is given as follows:

**Definition 4.4.1.** A Transportation-constrained Flexible Job Shop (TFJS) is a tuple $(L, V, J, r_j\,(j \in J), \mathcal{P}, \mathcal{T})$, with the following elements:

*Variables*

| | |
|---|---|
| $L$ | The set of locations, including the loading station $ls$, unloading station $us$, and machine locations. |
| $V$ | The set of automated guided vehicles. |
| $J, r_j \in \mathbb{N}$ | The set of jobs, with every job $j \in J$ describing a sequence of $r_j + 2$ operations $\langle o_{j,0}, \ldots, o_{j,r_j+1} \rangle$ where operation $o_{j,k}$ denotes the $k$-th operation of the $j$-th job. Operation $o_{j,0}$ denotes the loading of materials at the loading station $ls$, and operation $o_{j,r_j+1}$ denotes the unloading of the finished product at the unloading station $us$. |

*Derived Variables*

| | |
|---|---|
| $O = \{o_{j,k} \mid j \in J, k \in \mathbb{N}_0^{\leq r_j+1}\}$ | The set of operations including the loading and unloading operations for each job. |

*Functions*

| | |
|---|---|
| $\mathcal{P} : O \times L \hookrightarrow \mathbb{R}_0^+$ | Partial function denoting the processing time of an operation $o \in O$ at a location $l \in L$. A respective operation $o$ cannot be performed at the corresponding location $l$ when $\mathcal{P}(o, l)$ is undefined. For the operations $o_{j,0}$ and $o_{j,r_j+1}$ of any job $j \in J$, $\mathcal{P}$ is only defined for the loading station $ls$ and unloading station $us$, respectively. Processing times for operations other than $o_{j,0}$ and $o_{j,r_j+1}$ are assumed to be non-zero. |
| $\mathcal{T} : L \times L \to \mathbb{R}_0^+$ | Transfer time between two locations. |

*Derived Variables (Continued)*

| | |
|---|---|
| $L_o = \{l \in L \mid \mathcal{P}(o,l) \text{ defined}\}$ | The set of locations that can perform operation $o$. |
| $LT = \{lt_{j,k} \in T \mid j \in J, k \in \mathbb{N}_0^{\leq r_j}\}$ | The set of loaded transfers where $lt_{j,k}$ is a shorthand notation for the loaded transfer between $o_{j,k}$ and $o_{j,k+1}$. |
| $ET = \{et_{j,k} \in T, \mid j \in J, k \in \mathbb{N}_0^{\leq r_j}\}$ | The set of empty transfers for each loaded transfer in $LT$, with $et_{j,k}$ forming the shorthand notation for the corresponding empty transfer for $lt_{j,k}$. The duration of this transfer can be 0 if a vehicle is already at the correct location. |
| $T = LT \cup ET$ | The complete set of all transfers. |

The above definitions use the following assumptions:

- Job ordering is arbitrary.
- Product materials are initially available at the loading station.
- The considered vehicles are a fleet of identical automated guided vehicles.
- All vehicles are available at the loading station at the start.
- Vehicles have unlimited range and do not need charging/maintenance.
- Vehicles must be present to support loading and unloading when the same machine processes any two consecutive operations of a job.
- Loading/unloading times are included in the transfer and/or processing times.
- Machines have sufficient buffer capacity.

This brings us to the definition of the schedule for the transportation-constrained flexible job shop:

**Definition 4.4.2.** A *transporation-constrained flexible job-shop schedule* $\mathcal{S}, \mathcal{C}_t, \alpha, \beta$ for a TFJS describes the starting times, completion times, machine allocation, and vehicle allocation for all operations and transfers respectively.

*Decision variables*

| | |
|---|---|
| $\mathcal{S} : O \cup T \to \mathbb{R}_0^+$ | Complete function giving the starting time of an operation or transfer. |
| $\mathcal{C}_t : T \to \mathbb{R}_0^+$ | Complete function giving the completion time of a transfer. |
| $\alpha : O \to L$ | Complete function giving the allocation of an operation to a location. |
| $\beta : T \to V$ | Complete function giving the allocation of a transfer to a vehicle. |

*Derived functions*

| | |
|---|---|
| $\mathcal{C}_o : O \to \mathbb{R}_0^+$ | Complete function giving the completion time of an operation, where for all $o \in O$, $\mathcal{C}_o(o) = \mathcal{S}(o) + \mathcal{P}(o, \alpha(o))$. |

The following constraints apply:

**JC1.** Every machine can execute at most one operation at a time:

$$\forall o_x, o_y \in O : (o_x = o_y) \vee \alpha(o_x) = \alpha(o_y) \Rightarrow (\mathcal{C}_o(o_x) \leq \mathcal{S}(o_y) \vee \mathcal{C}_o(o_y) \leq \mathcal{S}(o_x)).$$

**JC2.** Every vehicle can perform one transfer at a time:

$$\forall t_x, t_y \in T : (t_x = t_y) \vee \beta(t_x) = \beta(t_y) \Rightarrow (\mathcal{C}_t(t_x) \leq \mathcal{S}(t_y) \vee \mathcal{C}_t(t_y) \leq \mathcal{S}(t_x)).$$

**JC3.** Operations can only be executed on specified machines.

$$\forall o \in O : \alpha(o) \in L_o$$

**JC4.** An operation can only start when its preceding loaded transfer is completed.

$$\forall j \in J : \forall k \in \mathbb{N}_0^{\leq r_j} : \mathcal{C}_t(lt_{j,k}) \leq \mathcal{S}(o_{j,k+1})$$

**JC5.** A loaded transfer can only start when its preceding operation is completed.

$$\forall j \in J : \forall k \in \mathbb{N}_0^{\leq r_j} : \mathcal{C}_o(o_{j,k}) \leq \mathcal{S}(lt_{j,k})$$

**JC6.** Empty and loaded transfers for a particular operation are always performed by the same vehicle:

$$\forall j \in J : \forall k \in \mathbb{N}_0^{< r_j} : \beta(et_{j,k}) = \beta(lt_{j,k})$$

**JC7.** A loaded transfer is always preceded by an empty transfer

$$\forall j \in J : \forall k \in \mathbb{N}_0^{\leq r_j} : \mathcal{C}_t(et_{j,k}) = \mathcal{S}(lt_{j,k})$$

**JC8.** The completion time of a loaded transfer is bounded by the start of the transfer plus the transfer time to the delivery location.

$$\forall j \in J : \forall k \in \mathbb{N}_0^{\leq r_j} : \mathcal{C}_t(lt_{j,k}) \geq \mathcal{S}(lt_{j,k}) + \mathcal{T}(\alpha(o_{j,k}), \alpha(o_{j,k+1}))$$

**JC9.** The completion time of an empty transfer is bounded by the transfer time to the pickup location from the vehicle's prior location.

$$\begin{gathered}\forall j_1, j_2 \in J : \forall k_1 \in \mathbb{N}_0^{\leq r_{j_1}} : \forall k_2 \in \mathbb{N}_0^{\leq r_{j_2}} : \\ \beta(lt_{j_1,k_1}) = \beta(et_{j_2,k_2}) \wedge \mathcal{S}(lt_{j_1,k_1}) < \mathcal{S}(et_{j_2,k_2}) \\ \wedge \nexists j_3 \in J, k_3 \in \mathbb{N}_0^{\leq r_{j_3}} : \beta(lt_{j_1,k_1}) = \beta(lt_{j_3,k_3}) \wedge \mathcal{S}(lt_{j_1,k_1}) < \mathcal{S}(lt_{j_3,k_3}) < \mathcal{S}(et_{j_2,k_2}) \\ \implies \mathcal{S}(et_{j_2,k_2}) = \mathcal{C}_t(lt_{j_1,k_1}) \wedge \mathcal{C}_t(et_{j_2,k_2}) \geq \mathcal{S}(et_{j_2,k_2}) + \mathcal{T}(\alpha(o_{j_1,k_1+1}), \alpha(o_{j_2,k_2}))\end{gathered}$$

A schedule is feasible when all of these constraints are satisfied, and infeasible otherwise.

Lastly, a scheduling problem is defined by an optimization objective. In this research, we focus on a makespan objective, which is defined as:

**Definition 4.4.3** (makespan)**.** The makespan $C_{max}$ of a schedule $\mathcal{S}, \mathcal{C}_t, \alpha, \beta$ is the latest completion time of the last *machine* operation of a job:

$$C_{max}(\mathcal{S}, \mathcal{C}_t, \alpha, \beta) = \max_{j \in J}(\mathcal{C}_o(o_{j,r_j}))$$

The makespan is defined as the maximum completion time of a job. Hence, the last loaded transfer, which delivers the final product to the unloading station, is not included in the makespan. This follows definitions used in the reference literature that follow the benchmarks of Bilge and Ulusoy [4] and Deroussi and Nore [12]. While the above definition of makespan implies that the last loaded transfer and unloading operations are unnecessary, they are still incorporated in the general definition. When we choose other objective functions, situations might occur where these tasks need to be part of the solution.

**Definition 4.4.4** (transporation-constrained job-shop scheduling problem with a makespan objective)**.** The optimal solution for a Transportation-constrained Flexible Job-Shop Scheduling Problem (TFJSSP) with a makespan objective is a feasible schedule $\mathcal{S}, \mathcal{C}_t, \alpha, \beta$ with the smallest possible makespan $C_{max}(\mathcal{S}, \mathcal{C}_t, \alpha, \beta)$.

**Example 4.4.1.**

Consider the illustrative TFJSSP introduced in Section 4.3. Describing this TFJSSP with the variables and derived functions introduced in Definition 4.4.1 is fairly straightforward. When we apply the constraints as specified in Definition 4.4.2 and optimize the problem using a makespan objective as defined in Definition 4.4.4, we obtain the schedule shown in Figure 7.

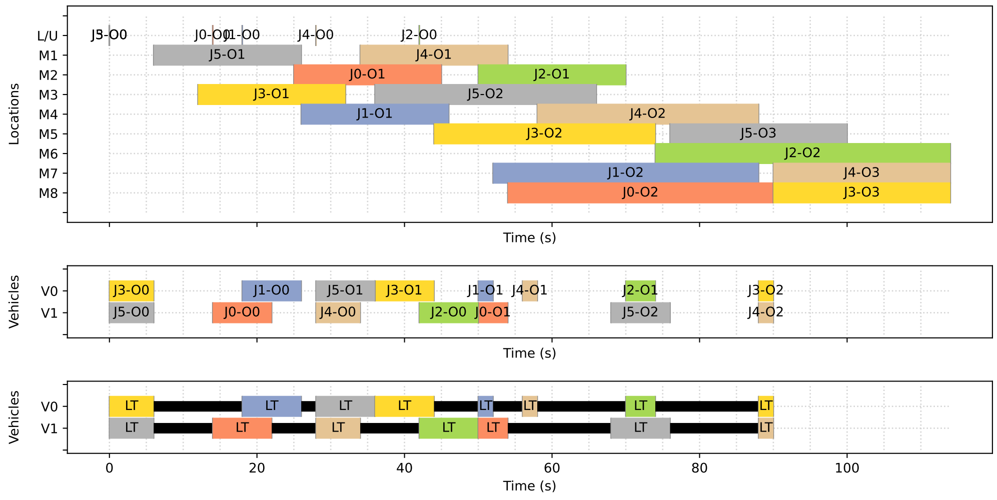


**Figure 7:** Optimal makespan solution for the illustrative TFJSSP.

The first panel shows the operation scheduling, where `J0-O0` denotes operation $o_{0,0}$, and so on. The machines are represented as `M1`, $\cdots$ ,`M8`; the loading station is denoted `L/U`. The second panel shows the loaded transfers where `J0-O0` denotes loaded transfer $lt_{0,0}$. For readability, we separated the empty transfers into a third panel where empty transfers are shown as a black interval. Together, the panels show the operation and transfer starting times $\mathcal{S}$, the transfer completion times $\mathcal{C}_t$, the machine allocation $\alpha$, and vehicle allocation $\beta$.

Observe that the found makespan is equal to $C_{max}(\mathcal{S}, \mathcal{C}_t, \alpha, \beta) = 114$, which is in line with the conclusion of Ham [16]. Note that in this case, we did not consider the last loaded transfers to the unloading station, as these can be ignored for the considered makespan objective.

## 4.5 | Constraint-Programming Solution

To find an optimal solution to the TFJSSP, a Constraint Programming (CP) model was created. CP has been shown to be a powerful mechanism in obtaining optimal solutions to scheduling problems, outperforming mathematical programming approaches such as those seen in [27, 16, 30, 17, 13].

### 4.5.1 | Modeling Setup

To achieve the best CP results, it is important to utilize the temporal functions that are part of CP Optimizer's syntax. Ham [16] specified a CP model for the TFJSSP with a makespan objective. This CP solution outperforms the MP solution of Homayouni and Fontes [19]—the only other exact solution—for the benchmark by Deroussi. The model utilized in this study is largely based on the model specified by Ham and applies various CP modeling techniques.

**Alternative resources:** The traditional FJSSP is commonly specified as a CP model by using the `alternative` constraint. This notion can also be applied to the TFJSSP. Alternative machines for each operation can be specified as an optional interval variable $machine\_operation\_options_{j,o,m}$ for each job $j$, operation $o$ and the alternative machines $m$. Then, by applying the `alternative` constraint only one alternative is chosen. For loaded transfers, this concept is extended not only to include alternative vehicles but also the starting location and destination of the loaded transfer. Based on the solver's chosen alternative it is then possible to exploit constraints that need spatial information of a vehicle.

**Transition matrix:** Due to the available spatial information, it becomes straightforward to specify the duration of a transfer. The optional interval variable for a loaded transfer specifies the starting location and destination from which the exact travel duration is known. Empty transfers are then modeled as setup times between these loaded transfers. This is achieved by specifying a transition matrix in the loaded transfer sequence that contains the delay for moving from the destination of a loaded transfer to the start location of the succeeding loaded transfer. The transition matrix is passed as a parameter to the `no_overlap` constraint to take effect.

This modeling technique is visualized in Figure 8. Two interval variables (blue and red) specify loaded transfers for the same vehicle. Between these two loaded transfers, the vehicle needs to travel to machine $m3$. Due to the added transition matrix, the `no_overlap` constraint then makes sure the interval variables do not collide and leave room for the empty transfer.

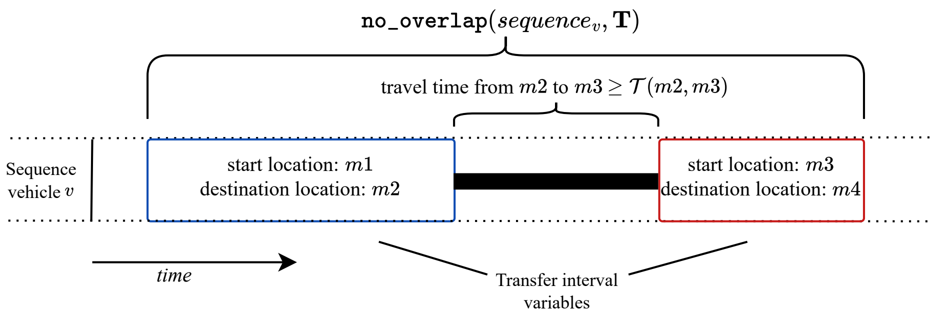

no_overlap($sequence_v$, $\mathbf{T}$)
travel time from $m2$ to $m3 \geq \mathcal{T}(m2, m3)$
Sequence vehicle $v$
start location: $m1$
destination location: $m2$
start location: $m3$
destination location: $m4$
time
Transfer interval variables

**Figure 8:** Gantt chart showing the no overlap constraint using a transition matrix.

**Example Continued.**

Reconsider the schedule obtained for the illustrative TFJSSP in Figure 7. The black boxes representing empty transfers in the bottom panel are in fact the result of a `no_overlap` constraint with an appropriate transition matrix.

**Loading and unloading operations:** For the first operation of a job to start, a loaded transfer delivers the starting materials from the loading station. Additionally, after the last operation, a loaded transfer to the unloading station marks the completion of the product. The corresponding loading and unloading operations are typically not specified explicitly in TFJSSP instances in the literature. To make sure these extra transfers are correctly performed, two operations are automatically added to each job when not specified. The first one marks the first operation of each job and takes place at the loading station. The second one marks the end of each job and takes place at the unloading station. These operations take zero time.

## 4.5.2 | Constraint Program

The CP model follows Definition 4.4.1-4.4.4 and is mostly similar to the specification by Ham [16]. The presented notations are slightly different from the syntactical notations used in Section 3.6, in line with the template we use for the mathematical definitions and to be consistent with the conventions in literature.

*Variables*

| | |
|---|---|
| $L, J, r_j (j \in J), V, \mathcal{P}, \mathcal{T}$ | As in Definition 4.4.1. Also derived notations are adopted. |

*Derived Variables*

| | |
|---|---|
| $\mathbf{T}$ | A transfer time matrix specifying transfer times between each possible pair of loaded transfers in line with transfer time function $\mathcal{T}$. For any pair of transfers $t_1$ from location $l_1$ to location $l_1'$ and $t_2$ from location $l_2$ to location $l_2'$, $\mathbf{T}[(l_1, l_1'), (l_2, l_2')] = \mathcal{T}(l_1', l_2)$. |

*Decision Variables*

| | |
|---|---|
| $operations_{o_{j,k}}$ | Interval variable for the $k$-th operation of job $j$. |
| $machine\_operation\_options_{o_{j,k},l}$ | Optional interval variable with processing time $\mathcal{P}(o_{j,k},\, l)$ for each location $l$ able to perform the $k$-th operation of job $j$. |
| $transfers_{o_{j,k}}$ | Interval variable for each loaded transfer $lt_{j,k}$. |
| $vehicle\_transfer\_options_{o_{j,k},v,sl,dl}$ | Optional interval variable for each vehicle $v$ performing the loaded transfer following the $k$-th operation of job $j$, by moving from source location $sl \in L_{o_{j,k}}$ to destination location $dl \in L_{o_{j,k+1}}$. |
| $transfer\_sequence_v$ | Sequence variable of loaded transfers for each vehicle $v$. |

The program is then constructed accordingly:

$$\boldsymbol{alternative}(operations_o, [machine\_operation\_options_{o,l}]) \quad \text{For all } o \in O \text{ with } l \in L_o \tag{4.1}$$

$$\boldsymbol{alternative}(transfers_o, [vehicle\_transfer\_options_{o,v,sl,dl}]) \quad \text{For all } o \in O, v \in V, \text{ source location } sl \in L_{o_{j,k}} \text{ and destination location } dl \in L_{o_{j,k+1}} \tag{4.2}$$

$$\boldsymbol{no\_overlap}(machine\_operation\_options_{o,l}) \quad \text{For all } l \in L \tag{4.3}$$

$$\boldsymbol{no\_overlap}(transfer\_sequence_v, \mathbf{T}) \quad \text{For all } v \in V \tag{4.4}$$

$$\boldsymbol{end\_before\_start}(transfers_{o_{j,k}}, operations_{o_{j,k+1}}) \quad \text{For all } j \in J \text{ and } k \in \mathbb{N}_0^{\leq r_j} \tag{4.5}$$

$$\boldsymbol{end\_before\_start}(operation_{o_{j,k}}, transfers_{o_{j,k}}) \quad \text{For all } j \in J \text{ and } k \in \mathbb{N}_0^{\leq r_j} \tag{4.6}$$

$$\begin{aligned}&\boldsymbol{presence\_of}(vehicle\_transfer\_options_{o_{j,k},v,sl,dl}) \implies \\ &\quad \boldsymbol{presence\_of}(machine\_operation\_options_{o_{j,k},sl}) \wedge \\ &\quad \boldsymbol{presence\_of}(machine\_operation\_options_{o_{j,k+1},dl})\end{aligned} \quad \text{For all } j \in J, k \in \mathbb{N}_0^{\leq r_j}, v \in V, sl \in L_{o_{j,k}} \text{ and } dl \in L_{o_{j,k+1}} \tag{4.7}$$

$$\boldsymbol{size\_of}(vehicle\_transfer\_options_{o_{j,k},v,sl,dl}) \geq \mathcal{T}(sl, dl) \quad \text{For all } j \in J, k \in \mathbb{N}_0^{\leq r_j}, v \in V, sl \in L_{o_{j,k}} \text{ and } dl \in L_{o_{j,k+1}} \tag{4.8}$$

Equation 4.1 constrains that each operation is executed on one of the available machines. Equation 4.2 specifies that the loaded transfer for an operation has multiple alternative starting and destination locations and can be performed by any available vehicle. Equation 4.3 and Equation 4.4 constrain that machines and vehicles can only perform one operation or loaded transfer at a time. Furthermore, Equation 4.4 ensures that empty transfers are correctly performed. Equation 4.5 and Equation 4.6 specify the precedence constraints between operations and loaded transfers. Equation 4.7 ensures that the start and destination locations of a loaded transfer match with the chosen machines of the relevant operations. Lastly, Equation 4.8 ensures that a loaded transfer takes up a minimally specified time.

The CP constraints match with the constraints specified in Definition 4.4.2. An overview can be found in Table 6.

| **Definition Constraints** | **CP constraints** |
|---|---|
| JC1 | Equation 4.3 |
| JC2 | Equation 4.4 |
| JC3 | Follows from $machine\_operation\_options_{o_{j,k},l}$ definition |
| JC4 | Equation 4.5 |
| JC5 | Equation 4.6 |
| JC6 | Empty transfers are implicit delays between loaded transfers and are therefore automatically coupled |
| JC7 | Empty transfers are implicit delays between loaded transfers and are therefore automatically coupled |
| JC8 | Equation 4.8 |
| JC9 | Using the transition matrix in Equation 4.4 |

**Table 6:** Connections between constraints of Definition 4.4.2 and the CP model

Chapter 5

# Conflict-Free Routing

New challenges arise when we incorporate more detailed vehicle routing in the TJSPs. When moving semi-finished jobs or materials inside the FMS, autonomous vehicles must avoid possible bottlenecks such as collisions, deadlocks, and livelocks. The optimization problem related to these three bottlenecks is often named the Conflict-Free Routing Problem (CFRP). This chapter covers a general mathematical definition of a generic CFRP. Furthermore, we evaluate various solving techniques to determine which one we should use in the decomposed solution of the CFTFJSSP.

## 5.1 | Introduction and State of the Art

The CFRP describes routing a fleet of AGVs to perform transportation tasks. This problem is most commonly seen as part of autonomous vehicle scheduling in flexible manufacturing systems. Transportation tasks are characterized by timing constraints, specifically release times and deadlines, and start and destination locations. The single objective of the CFRP is to find time-feasible routing assignments from the source to the destination locations without conflicts between vehicles.

As AGV systems in flexible manufacturing share the same workspace, it is essential to coordinate their movements to prevent these conflicts and ensure smooth transfers from one production station to another based on the given timing constraints. These characteristics differentiate a CFRP from other vehicle routing problems. While the CFRP in flexible manufacturing can be considered a variation of the classical Vehicle Routing Problem (VRP) [5], Qiu *et al.* [40] point out significant distinctions, sufficient to treat the CFRP as a separate problem: Classical VRPs usually deal with large metropolitan-scale path networks in contrast to the compact factory-scale networks seen in FMSs. This allows the classical VRP to ignore collisions and congestion, meaning that the shortest time path is always the shortest path, which might not be the case for CFRPs. Furthermore, solving the CFRP as a shortest-path problem does not consider the timing of these routes, making it unfit for most flexible manufacturing applications.

There are different ways to define a conflict-free routing problem, each with varying levels of complexity. Complexity is based on three characteristics:

1. *Routing network:* A path layout that identifies routing possibilities from one point to another. Routing networks are mostly defined as unidirectional or bidirectional graphs.
2. *Vehicle behavior:* The specification of the level of detail for the vehicle behavior. The problem may, for example, incorporate adjustable vehicle speed and battery drainage.
3. *Conflict definition:* The specification of a conflict. Generally, a conflict occurs when a collision, deadlock, or livelock occurs. However, the notion of conflict can often be simplified depending on the exact problem description.

Dedicated conflict-free routing studies such as [24, 25] usually consider complex CFRPs. In these types of studies, the routing network may include a bounded number of vehicles within a single edge of the routing graph. Furthermore, complex vehicle behavior is added to the problem structure providing vehicle characteristics such as variable speed, payload capacities, and charging behavior. These added characteristics may be prone to new types of conflicts and severely complicate the routing assignment.

While we can see the CFRP as a separate optimization problem, it is worth noting that, in fact, it is strongly related to TJSPs. In a flexible manufacturing system, higher-level job scheduling leads to a series of pickup and drop-off tasks. As the solution schedule also provides time windows on these transportation tasks, all the variables that form a CFRP are in place.

Several studies have closely examined CFRP in relation to FMSs, as seen in [34, 44, 31, 29]. In these types of studies, a simple definition of the CFRP is typically used. Here, the routing network is mostly presented as a simple graph. An important aspect is that all vehicles and distances between adjacent nodes are considered equal. This means that the movement from one node to another is always assumed to be equal to the same unit of time. Furthermore, conflicts can only occur when more than one vehicle is present in a node at the same time or when more than one vehicle is on an edge at the same time.

## 5.2 | An Illustrative Conflict-Free Routing Problem

As this research follows the line of research where we treat the CFRP as part of a TJSP, we follow the simple CFRP definitions seen in [34, 44, 31, 29]. The CFRPs that occur in the context of a TFJSSP are feasibility problems, where transfers have already been allocated to vehicles and are time-bounded by release times and deadlines derived from operation completion and starting times. Conflicts are restricted merely to situations where collisions occur. Due to the presence of deadlines, deadlocks and livelocks cannot occur as after a given time the solution will simply be evaluated as infeasible.

Let us give an illustrative example of the conflict-free routing problem that is considered in this research by using routing network layout 2 as presented by Lyu *et al.* [31]. The routing network is in the form of a graph and can be seen in Figure 9.

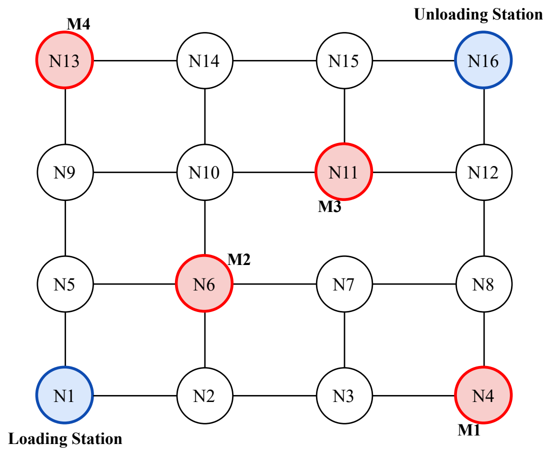


**Figure 9:** FMS layout 2 of the CFTFJSSP instances presented by Lyu *et al.* [31]

As the problem presented by Lyu *et al.* is the CFTFJSSP, we see that the points of interest—the loading/unloading stations and machines—in the routing network are similar to those in variable $L$ from the TFJS of Definition 4.4.1.

The CFRP is then formulated as follows. Given a set of transfers, find a feasible routing assignment such that no conflicts occur. As mentioned, we can presume that transfers have already been allocated to vehicles and are time-bounded by release times and deadlines.

Let us assume we have a single vehicle $v$, starting in node 1. The objective is simple; drive to the unloading station in node 16 within six time units. We assume that each transition from one node to the next takes a single time unit. As we can form a path of six edges, we find the routing schedule seen in Figure 10.

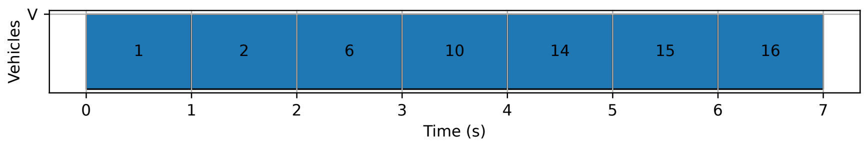


**Figure 10:** Routing schedule from node 1 to node 16.

Each time interval in the figure shows us the node that is occupied by a vehicle. After a time unit has passed the vehicle is presumed to have moved either to a new node that is connected in the routing network, or to the node it was currently in. It can be seen that the vehicle reaches its destination node 16 at time 6.

Now let us assume the same vehicle starts in node 11 and needs to transfer to node 7 within 2 time units. We can see there is an obstacle between node 11 and node 7, as there is no edge forming a connection between these nodes. As a result, vehicle $v$ needs to find a way around this connection, which is not possible within the given timing constraints. Hence, the CFRP instance is infeasible.

Besides obstacles, a vehicle can also encounter other vehicles. In the considered CFRP for this research we consider any collision between vehicles a conflict. A conflict can then be concluded when multiple vehicles are present in the same node at the same time (presence conflict), or when two vehicles swap nodes at the same time (swap conflict). Given sufficient time, a vehicle is always able to find a conflict-free route from one location to another. However, timing constraints often do not allow vehicles to take alternative routes to avoid these conflicts.

## 5.3 | Problem Definition

The challenge in our CFRP is to find conflict-free sequences of steps for all transfers through a routing network within the specified release times and deadlines. We introduce a unit time step so that, in the context of a CFTFJSSP, it is possible to adapt the granularity of timing between the processing of operations and the transfer steps.

**Definition 5.3.1.** A Conflict-Free Routing Problem (CFRP) instance is a tuple $(N, E, V, FN, \tau^{\Delta}, y_v(v \in V), T_v(v \in V), \delta, \epsilon)$, with the following elements:

| *Variables* | |
|---|---|
| $N$ | The set of nodes in the routing network. |
| $E \subseteq N^2$ | The set of edges connecting the nodes in the routing network, specifying valid movements from one node to another. |
| $V$ | The set of automated guided vehicles. |
| $FN \subseteq N$ | The set of free nodes where more than one vehicle $v \in V$ may be present simultaneously, including the node $l$ marking the start location of all vehicles $v \in V$. |
| $\tau^{\Delta} \in \mathbb{N}$ | Duration of a single step. We assume a uniform step duration. |
| $y_v \in \mathbb{N}_0, T_v = \langle t_{v,1}, \ldots, t_{v,y_v} \rangle$ | The sequence of transfers $t_{v,k}$ for vehicle $v \in V$ with $y_v$ denoting the number of transfers for vehicle $v \in V$. |

*Derived Variables*

|  |  |
|---|---|
| $T = \{t_{v,k} \mid v \in V, k \in \mathbb{N}^{\leq y_v}\}$ | The set of all transfers. |

*Functions*

|  |  |
|---|---|
| $\delta : T \to \mathbb{R}_0^+$ | Specification of the transfer's deadline. |
| $\epsilon : T \to N$ | Specification of the transfer's destination node. |

*Derived Functions*

|  |  |
|---|---|
| $\gamma : T \to \mathbb{R}_0^+$ : | Specification of the transfer's release time. A transfer is released at the deadline of the previous release. For all $v \in V$ and $k \in \mathbb{N}^{<y_v}$, $\gamma(t_{v,k+1}) = \delta(t_{v,k})$; $\delta(t_{v,0} = 0)$. |
| $\beta : T \to V$ | Derived vehicle allocation, giving the vehicle $v \in V$ assigned to a transfer $t_{v,k}$. |

The above definitions use the following assumptions:

- The considered vehicles are a fleet of identical automated guided vehicles.
- The routing network is a unidirectional graph.
- The first transfer for a vehicle starts from the start location $l$.
- For all transfers on the same vehicle, except the first, the start location is its preceding transfer's destination location.
- Travel time between adjacent nodes in the routing network is a single time step $\tau^{\Delta}$.
- Conflicts are situations where two vehicles are present in the same node or switch nodes at the same time instant.

A schedule for a CFRP instance as per Definition 5.3.3 can be described as follows:

**Definition 5.3.2.** A *conflict-free routing schedule* $z_t(t \in T), \mathcal{N}, \mathcal{S}$ describes the steps for all transfers with their starting times. A step describes the movement from one node to another node in the routing network.

*Decision Variables*

|  |  |
|---|---|
| $z_t \in \mathbb{N}_0$ | The number of steps in transfer $t \in T$. |

*Derived Variables*

|  |  |
|---|---|
| $S = \{s_{t,i} \mid t \in T, i \in \mathbb{N}^{\leq z_t}\}$ | The set of all steps $s$, where $s_{t,i}$ is a shorthand notation for the $i$-th step in transfer $t \in T$. |

*Decision Variables (Continued)*

|  |  |
|---|---|
| $\mathcal{N} : S \to N$ | Complete function giving the destination node of all steps. |
| $\mathcal{S} : S \to \mathbb{R}_0^+$ | Complete function giving the starting time of a step. |

*Derived functions*

| | |
|---|---|
| $\mathcal{C} : S \to \mathbb{R}_0^+$ | Complete function giving the completion time of a step, where for all $s \in S$, $\mathcal{C}(s) = \mathcal{S}(s) + \tau^{\Delta}$. |
| $Nxt_v : S \hookrightarrow S$ | For each vehicle $v \in V$, partial function $Nxt_v$ defines for each step of $v$, except the last one, the next step of that vehicle. Given the vehicle allocation $\beta$ and the strict sequencing of all transfers per vehicle, $Nxt_v$ is well defined. |

The following constraints apply:

**RC1.** Steps are performed sequentially during each transfer.

$$\forall t \in T : \forall i \in \mathbb{N}^{<z_t} : \mathcal{C}(s_{t,i}) \leq \mathcal{S}(s_{t,i+1})$$

**RC2.** Vehicles can only move to adjacent nodes.

$$\forall v \in V : \forall s \in S : (\mathcal{N}(s), \mathcal{N}(Nxt_v(s)) \in E$$

**RC3.** The last step for a transfer is to the transfer's destination node.

$$\forall t \in T : z_t > 0 \implies \mathcal{N}(s_{t,z_t}) = \epsilon(t)$$

**RC4.** A transfer should start with its first step not earlier than its start time.

$$\forall t \in T : z_t > 0 \implies \gamma(t) \leq \mathcal{S}(s_{t,1})$$

**RC5.** A transfer should end no later than its deadline.

$$\forall t \in T : z_t > 0 \implies \mathcal{C}(s_{t,z_t}) \leq \delta(t)$$

**RC6.** Only one vehicle can be present in a non-free node at a given time.

$$\begin{gathered}\forall s_1, s_2 \in S : \beta(s_1) \neq \beta(s_2) \wedge \mathcal{N}(s_1) = \mathcal{N}(s_2) \wedge \mathcal{N}(s_1) \notin FN \\ \implies \mathcal{S}(Nxt_{\beta(s_1)}(s_1)) \leq \mathcal{S}(s_2) \vee \mathcal{S}(Nxt_{\beta(s_2)}(s_2)) \leq \mathcal{S}(s_1)\end{gathered}$$

**RC7.** Vehicles cannot swap nodes at the same time.

$$\begin{gathered}\forall s_1, s_2 \in S : \beta(s_1) \neq \beta(s_2) \wedge \mathcal{N}(s_1) = \mathcal{N}(Nxt_{\beta(s_2)}(s_2)) \wedge \mathcal{N}(Nxt_{\beta(s_1)}(s_1)) = \mathcal{N}(s_2) \\ \implies \mathcal{C}(s_1) \leq \mathcal{S}(s_2) \vee \mathcal{C}(s_2) \leq \mathcal{S}(s_1)\end{gathered}$$

A schedule is feasible when all of these constraints are satisfied, and otherwise, it is infeasible.

**Definition 5.3.3** (conflict-free routing problem)**.** The Conflict-Free Routing Problem (CFRP) is the problem of finding a feasible conflict-free routing schedule $z_t, \mathcal{N}, \mathcal{S}$ for a given CFRP instance as defined in Definition 5.3.1.

In the above definitions, constraints **RC6.** and **RC7.** ensure that the routing schedule is, in fact, conflict-free. To illustrate this, let us consider the situation in Figure 11 where both steps $s_{t_1,i_1}$ and $s_{t_2,i_2}$ specify the movement to node $n_1$. A conflict occurs when both steps are performed, and $n_1$ holds two vehicles. To make sure this conflict is eliminated $s_{t_1,i_1}$ can only start when $Nxt_{v_2}(s_{t_2,i_2})$ started to move vehicle $v_2$ away from node 1, or vice versa. This is what constraint **RC6.** ensures.

Consider another situation in Figure 12, where steps $s_{t_1,i_1}$ and $s_{t_2,i_2}$ specify the movement to node $n_2$ and $n_1$, respectively. However, $v_2$ is located in $n_2$ and $v_1$ is at $n_1$. The conflict occurs when both steps are performed simultaneously, and two vehicles collide while trying to reach each other's position. The conflict can be resolved by making sure both steps cannot be performed simultaneously. This is what constraint **RC7.** ensures.

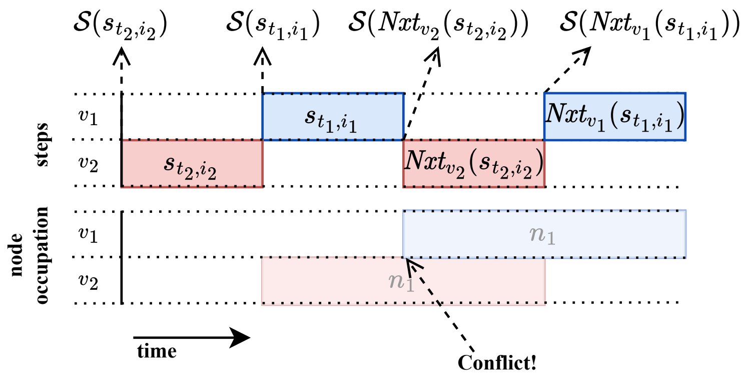


**Figure 11:** A node presence conflict

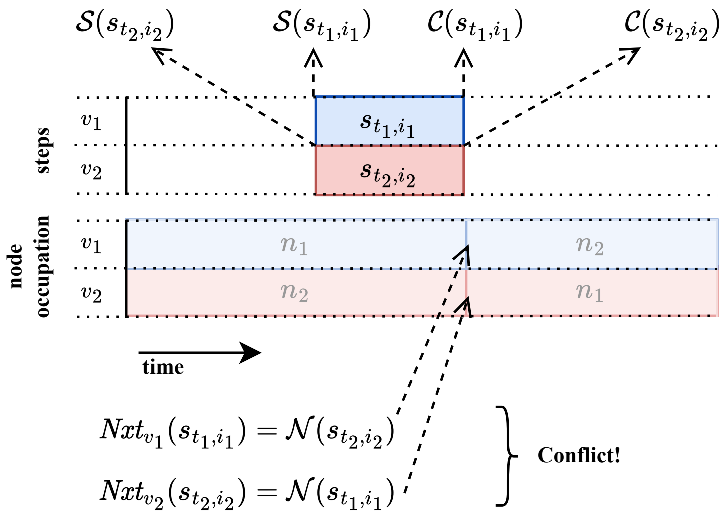

$\mathcal{S}(s_{t_2,i_2})$
$\mathcal{S}(s_{t_1,i_1})$
$\mathcal{C}(s_{t_1,i_1})$
$\mathcal{C}(s_{t_2,i_2})$
steps
$v_1$
$v_2$
$s_{t_1,i_1}$
$s_{t_2,i_2}$
node occupation
$n_1$
$n_2$
$n_2$
$n_1$
time
$Nxt_{v_1}(s_{t_1,i_1}) = \mathcal{N}(s_{t_2,i_2})$
$Nxt_{v_2}(s_{t_2,i_2}) = \mathcal{N}(s_{t_1,i_1})$
Conflict!

**Figure 12:** A switch conflict

**Example 5.3.1.**

Consider the illustrative example introduced in Section 5.2. The layout can be captured in a graph with nodes $N$ and edges $E$. Note that every edge in the layout corresponds to two edges in in unidirectional graph assumed as input for our CFRP. We can define the loading and unloading stations (node 1 and 16) as the free nodes. This implies that all vehicles can start in node 1. The remaining variables follow naturally.

Let us assume we need to find a conflict-free routing schedule for two vehicles $v_0$ and $v_1$ for the following set of transfers:

$$T = \{t_{0,1}, t_{0,2}, t_{0,3}, t_{1,1}, t_{1,2}, t_{1,3}\}$$
$$\boldsymbol{\delta} = [3, 6, 10, 3, 6, 10]$$
$$\boldsymbol{\epsilon} = [13, 6, 16, 4, 11, 13]$$

where $\boldsymbol{\delta}$ and $\boldsymbol{\epsilon}$ denote the deadline and destination functions $\delta, \epsilon$ in vector format. Observe that both vehicles need to do three transfers, both with deadlines 3, 6, and 10.

The resulting schedule can be found in Figure 13. Note that the two vehicles indeed reach their destinations exactly at their deadlines. Vehicle 0, for instance, reaches node 13 at time 3, node 6 at time 6, and node 16 at time 10.

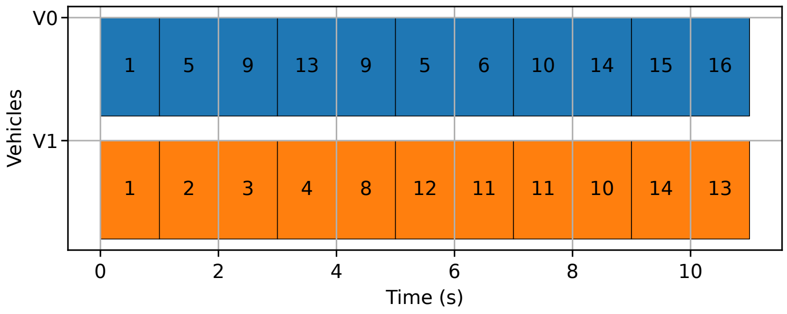


**Figure 13:** Routing schedule for the example set of transfers.

To clarify the paths the vehicles are driving, Figure 14 shows the steps both vehicle $v_0$ (blue) and vehicle $v_1$ (orange) take.

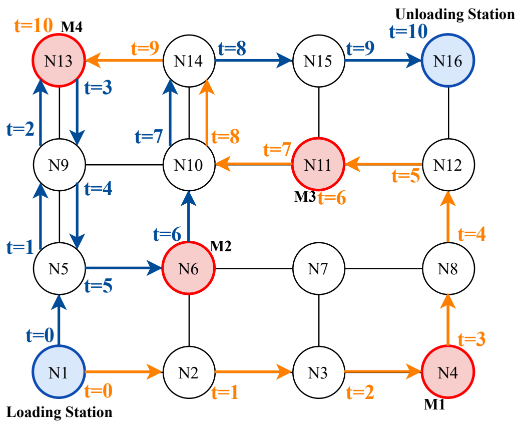


**Figure 14:** Routing paths with timestamps for the example set of transfers.

Observe that for their last transfers both vehicles need to take partially the same paths. Since vehicle $v_1$ has not a very strict deadline on its last transfer it chooses to wait in node 11 at time $t = 6$ until vehicle $v_0$ has passed by, in order to find a feasible schedule.

## 5.4 | Constraint Program

With the CFRP defined, a solving technique must be applied to find a solution. The goal of this study is to primarily use exact solving procedures such as CP for the scheduling of TJSPs. While CP performs well for scheduling problems, it is rarely used for vehicle routing problems where MP implementations form the standard. With this research's main goal to solve an integrated TJSP and CFRP, both solving techniques are assessed to compare their performance.

Let us start by developing a constraint program for the CFRP. CP performs really well when its temporal functions can be utilized. This requires some intelligent abstraction of the problem definition of Definition 5.3.3.

## 5.4.1 | Grid Structure

It is common practice to restrain routing graph structures in CFRPs in literature to *grids*, with nodes organized in rows and columns. A grid's inherent spatial structure makes it more efficient to determine adjacent locations than is possible for free graph structures. Figure 15 shows a 3 by 3 node graph projected inside a grid. It can be seen that the grid cells define a row-column coordinate for each node. All adjacent nodes are then referenced by subtracting or incrementing respective coordinate indexes:

**Definition 5.4.1.** Using the definitions in [46] as inpsiration, consider an $r$ by $c$ grid with $R = \mathbb{N}_0^{<r}$ and $C = \mathbb{N}_0^{<c}$. A cell $(i, j)$ in the grid is then defined as a pair of coordinates $n \in R \times C$. We can specify the set of adjacent cells, including the current cell, as $A : R \times C \to 2^{R \times C}$ where for each cell $(i, j)$:

$$A(i, j) = \{(i + p,\ j + q) \in R \times C \mid p, q \in \{-1, 0, 1\} \wedge pq = 0\}$$

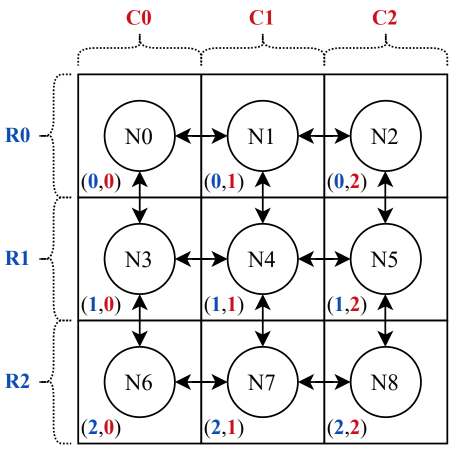


**Figure 15:** Graph to grid conversion.

**Example Continued.**

Reconsider our current running example from Section 5.2. The presented routing network is essentially a grid. However, it shows an obstacle between nodes 7 and 11. In order to deal with these obstacles, we consider such two nodes to be non-adjacent. This can be achieved by providing an appropriate $A$ function.

We limit our CP implementation to grid-structured graphs.

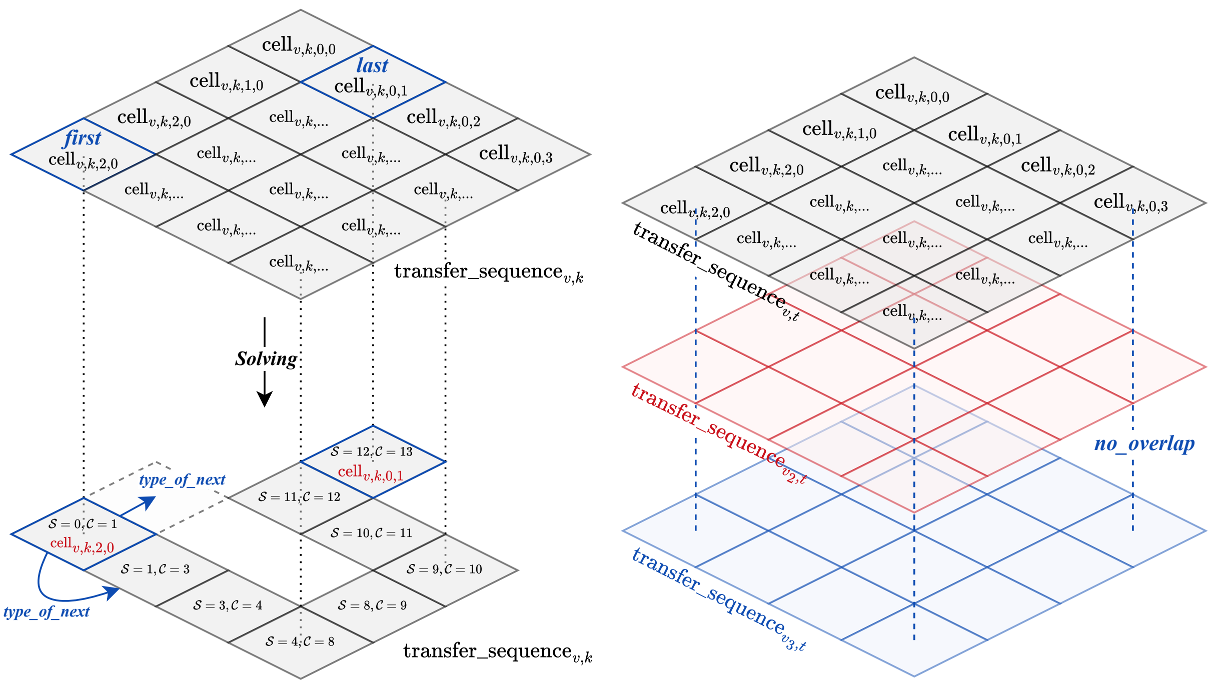
## 5.4.2 | Model Setup

Schedulers such as CP optimizer are often utilized for production scheduling but have no spefically defined syntax for vehicle routing problems. Therefore, we use inspiration from the prior works of Hà *et al.* [15]

and Booth and Beck [6]. Both contributions model the presence of vehicle $v$ at a location $n$ as an optional interval variable. These implementations do not address a CFRP but are a mere extension of the traditional VRP. Our model must extend their traditional VRP to allow vehicles to move through adjacent nodes without causing conflicts.

In our CP model, individual vehicle movements are specified using sequence variables that hold the movements through the grid. Where Definition 5.3.2 specifies a step—a change of location—the CP model indicates the occupancy of a location. Figure 11 and Figure 12 show us the difference between steps and location occupation.

This modeling setup can be visualized as seen in Figure 16a. A sequence variable $transfer\_sequence_{v,k}$ is introduced for all transfers $k$ on each vehicle $v$. Such a *transfer_sequence* is composed of unique optional interval variables $cell_{v,k,r,c}$ for each cell $(r, c)$ in the grid. By assigning a valid time interval to this interval variable, we indicate that the vehicle $v$ visited cell $(r, c)$ during transfer $k$ during the assigned time. All interval variables not visited are discarded and not used in the solution schedule. Observe in Figure 16 that we define a full grid of interval variables; after solving, a smaller sequence of interval variables remains in the solution schedule.


$cell_{v,k,0,0}$
last
$cell_{v,k,0,1}$
$cell_{v,k,1,0}$
first
$cell_{v,k,2,0}$
$cell_{v,k,0,2}$
$cell_{v,k,0,3}$
$cell_{v,k,...}$
$transfer\_sequence_{v,k}$
Solving
type_of_next
$\mathcal{S} = 0, \mathcal{C} = 1$
$\mathcal{S} = 1, \mathcal{C} = 3$
$\mathcal{S} = 3, \mathcal{C} = 4$
$\mathcal{S} = 4, \mathcal{C} = 8$
$\mathcal{S} = 8, \mathcal{C} = 9$
$\mathcal{S} = 9, \mathcal{C} = 10$
$\mathcal{S} = 10, \mathcal{C} = 11$
$\mathcal{S} = 11, \mathcal{C} = 12$
$\mathcal{S} = 12, \mathcal{C} = 13$
$transfer\_sequence_{v,t}$
$transfer\_sequence_{v_2,t}$
$transfer\_sequence_{v_3,t}$
no_overlap

**(a)** Path finding within the CP model. **(b)** Conflict avoidance within the CP model.

**Figure 16:** Constraint programming model setup of the CFRP.

Additional constraints ensure that each vehicle in a transfer sequence follows the correct moving behavior. The temporal constraints ***first*** and ***last*** allow constraining a certain cell to be visited at the start and end of the sequence, respectively. Additionally, using the type parameter of an interval variable, we can assign a unique spatial identifier to each interval variable in a transfer sequence. With the rules seen in Definition 5.4.1, we can specify that only certain interval variables can succeed one another. The ***type_of_next*** constraint allows us to force adjacent interval variables to be next in the sequence. Observe in Figure 16a $cell_{v,k,3,0}$ and $cell_{v,k,0,1}$ are defined as start and end locations of transfer $t_{v,k}$. Furthermore, it is shown that the first interval variable can be succeeded by two alternatives, of which only one is chosen in the solution schedule. Lastly, to ensure that multiple vehicles do not conflict within a cell, we can apply the ***no_overlap*** constraint on each interval variable $cell_{v,k,r,c}$ for all $v$ on the same location $(r, c)$ as seen in Figure 16b.

### 5.4.3 | Constraint Program

For the CP model, we follow Definition 5.3.3 and Definition 5.3.2. If the routing network is a grid where all adjacent nodes to a cell are reachable to a vehicle, the function $A$ of Definition 5.4.1 is a derived function. If the grid is irregular, as in the example of Section 5.2, then it needs to be provided as input.

| *Variables* | |
|---|---|
| $R$ | The set of rows in the routing network |
| $C$ | The set of columns in the routing network |
| $V, \tau^{\Delta}, y_v (v \in V), T_v (v \in V), \delta, \epsilon$ | As in Definition 5.3.1. Also derived notations are adopted. |

| *Derived Variables* | |
|---|---|
| $R \times C$ | The set of cell coordinates in the routing network |

| *Variables* | |
|---|---|
| $FC \subseteq R \times C$ | The set of free cells where more than one vehicle $v \in V$ may be present simultaneously, including the cell $l$ marking the start location of all vehicles $v \in V$. |

| *Derived Functions* | |
|---|---|
| $\zeta : T \to R \times C$ | Specification of the source cell of transfer $t_{v,k}$, which is either the start location $l \in FC$ if $k = 1$ or the destination node $\epsilon(t_{v,k-1})$, with $t_{v,k-1}$ the transfer preceding $t_{v,k}$. |
| $A : R \times C \to 2^{R \times C}$ | Adjacent cells, as defined in Definition 5.4.1. |

| *Decision Variables* | |
|---|---|
| $cell_{v,k,r,c}$ | Optional interval variable for each location possibly visited during transfer $t_{v,k}$. |
| $transfer\_sequence_{v,k}$ | Sequence variable of cell variables, determining the order of cells visited during transfer $t_{v,k}$. |

The program is constructed accordingly:

| | | |
|---|---|---|
| $\boldsymbol{end\_of}(cell_{v,k,\epsilon(t_{v,k})}) \leq \boldsymbol{start\_of}(cell_{v,k+1,\zeta(t_{v,k+1})})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v - 1}$ | (5.1) |
| $\boldsymbol{first}(transfer\_sequence_{v,k}, cell_{v,k,\zeta(t_{v,k})})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.2) |
| $\boldsymbol{start\_of}(cell_{v,k,\zeta(t_{v,k})}) = \gamma(t_{v,k})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.3) |
| $\boldsymbol{last}(transfer\_sequence_{v,k}, cell_{v,k,\epsilon(t_{v,k})})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.4) |
| $\boldsymbol{end\_of}(cell_{v,k,\epsilon(t_{v,k})}) = \delta(t_{v,k})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.5) |
| $\boldsymbol{presence\_of}(cell_{v,k,\zeta(t_{v,k})})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.6) |
| $\boldsymbol{presence\_of}(cell_{v,k,\epsilon(t_{v,k})})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.7) |
| $\boldsymbol{no\_overlap}(transfer\_sequence_{v,k})$ | For all $v \in V$ and $k \in \mathbb{N}^{\leq y_v}$ | (5.8) |
| $\boldsymbol{no\_overlap}([cell_{v_1,k_1,r,c}, cell_{v_2,k_2,r,c}])$ | For all $v_1, v_2 \in V, k_1, k_2 \in \mathbb{N}^{\leq y_v}$ and $(r,c) \in R \times C \setminus FC$ where $v_1 \neq v_2$ | (5.9) |

$$\textbf{\textit{type\_of\_next}}(transfer\_sequence_{v,k}, cell_{v,k,r,c}) = i \quad \text{For all } v \in V, k \in \mathbb{N}^{\leq y_v}, (r,c) \in R \times C, \text{ and } i \in A(r,c), \text{ where } (r,c) \neq \epsilon(t_{v,k}). \tag{5.10}$$

$$\textbf{\textit{start\_of}}(cell_{v,k,r,c}) + \tau^{\Delta} \leq \textbf{\textit{end\_of}}(cell_{v,k,r,c}) \quad \text{For all } v \in V, k \in \mathbb{N}^{\leq y_v} \text{ and } (r,c) \in R \times C \text{ where } (r,c) \neq \epsilon(t_{v,k}) \tag{5.11}$$

$$\textbf{\textit{end\_of}}(cell_{v,k,r,c}) = \textbf{\textit{start\_of\_next}}(transfer\_sequence_{v,k}, cell_{v,k,r,c}) \quad \text{For all } v \in V, k \in \mathbb{N}^{\leq y_v}, (r,c) \in R \times C \text{ and where } (r,c) \neq \epsilon(t_{v,k}) \tag{5.12}$$

$$\textbf{\textit{end\_of}}(cell_{v_2,k_2,r_1,c_1}) \leq \textbf{\textit{start\_of}}(cell_{v_1,k_1,r_1,c_1}) \vee \textbf{\textit{end\_of}}(cell_{v_1,k_1,r_2,c_2}) \leq \textbf{\textit{start\_of}}(cell_{v_2,k_2,r_2,c_2}) \quad \text{For all } v_1, v_2 \in V, k_1, k_2 \in \mathbb{N}^{\leq y_v} \text{ and } (r_1,c_1),(r_2,c_2) \in R \times C \text{ if cell } (r_1,c_1) \text{ is followed by cell } (r_2,c_2) \text{ in transfer } t_{v_1,k_1} \text{and vice versa in transfer } t_{v_2,k_2}. \tag{5.13}$$

Equation 5.1 ensures all transfers are performed sequentially. Equation 5.2, 5.4, 5.6, 5.7 ensure that the correct start and end cells are specified for each transfer and considered in the solution schedule. Equation 5.3 and Equation 5.5 ensure that the transfers meet their respective release times and deadlines. Equation 5.8 ensures a vehicle can visit only one cell simultaneously. Equation 5.9 ensures a cell can be visited by only one vehicle at a time. Equation 5.10 specifies that vehicles can only move to adjacent cells. Equation 5.11 specifies the duration of a step to be $\tau^{\Delta}$. Equation 5.12 states that steps within a transfer are performed sequentially. Finally, Equation 5.13 forces vehicles to not swap cells.

Table 7 shows how the CP model covers the constraints described in Definition 5.3.2.

| Definition Constraints | CP constraints |
|---|---|
| Input $T_v$ is a sequence. | Equation 5.1, Equation 5.6, Equation 5.8 |
| RC1 | Equation 5.11 |
| RC2 | Equation 5.2, Equation 5.10 |
| RC3 | Equation 5.4, Equation 5.7 |
| RC4 | Equation 5.3 |
| RC5 | Equation 5.5 |
| RC6 | Equation 5.9 |
| RC7 | Equation 5.13 |

**Table 7:** Connections between constraints of Definition 5.3.2 and the CP model

### 5.4.4 | Deviations

This model setup slightly deviates from the original mathematical definition in Definition 5.3.3. When using a CP approach, utilizing the temporal functions and special variables that CP offers is desirable. In each *transfer_sequence*, we define a list of optional locations to visit. The current CP implementation allows us to visit each of these locations only once within a transfer. However, the CFRP described by Definition 5.3.3 allows us to visit a node $n \in N$ multiple times during the same transfer. To the best of our knowledge, there is no other way to correctly implement the slightly more general CFRP definition while utilizing the special variables and temporal functions CP offers.

## 5.5 | Mathematical Programs

Another technique to solve the CFRP is MP. MP is more typically applied to solving the CFRP than CP as seen in [25, 33, 9, 44, 45, 29].

### 5.5.1 | Modeling Setup

Generally, the literature shows two MP approaches to solving the CFRP. The first approach considers a binary variable for a vehicle traversing from one node to another at a certain time as seen in [25, 33, 9, 29]. The second approach considers whether a vehicle is present in a node at a certain time, such as in [44, 31, 45]. For this research, we test both approaches and compare their respective results.

Using binary decision variables allows the MP models to be relatively compact compared to Definition 5.3.2. Our original definition considers *steps* as decision variables, which describe movement towards a *new location*. The MP models have decision variables considering *time* steps. In each time step, a vehicle performs a *step* from one location to another, possibly its current location.

### 5.5.2 | MP Variant 1

The first MP variant that we consider is an ILP that considers a binary variable for a vehicle traversing from one node to another at a certain time, where time is discretized according to the time step $\tau^{\Delta}$ with a time horizon corresponding to the latest transfer deadline. The model is an adaptation of the program developed by Nishi *et al.* [33].

*Variables*

| | |
|---|---|
| $N, E, V, \tau^{\Delta}, FN, y_v(v \in V), T_v(v \in V), \delta, \epsilon$ | As in Definition 4.4.1. Also derived notations are adopted. |

*Derived Variables*

| | |
|---|---|
| $A_n = \{a \in N \mid (n,a) \in E\} \cup \{n\}$ | The adjacent nodes for each node $n \in N$. |
| $H = \max_{t \in T}(\delta(t))$ | Time horizon, being the maximum deadline of all transfers $t \in T$. |
| $\mathbb{T} \subseteq \mathbb{N}_0$ | The discretized time domain. |

*Decision Variables*

$$x^{\tau}_{v,n_x,n_y} = \begin{cases} 1 & \text{if AGV } v \text{ steps from node } n_x \text{ to } n_y \text{ at time } \tau \in \mathbb{T}^{<H} \\ 0 & \text{otherwise} \end{cases}$$

The program is constructed accordingly:

$$\sum_{n_x, n_y \in N} x^{\tau}_{v,n_x,n_y} = 1, \qquad \forall \tau \in \mathbb{T}^{<H}, \forall v \in V \tag{5.14}$$

$$\sum_{n_y \in N \mid n_x \in A_{n_y}} x^{\tau}_{v,n_y,n_x} = \sum_{n_z \in A_{n_x}} x^{\tau+\tau^{\Delta}}_{v,n_x,n_z}, \qquad \forall \tau \in \mathbb{T}^{<H-\tau^{\Delta}}, \forall v \in V, \forall n_x \in N \tag{5.15}$$

$$\sum_{n_y \in A_l} x^{0}_{v,l,n_y} = 1, \qquad \forall v \in V \tag{5.16}$$

$$\sum_{n_y \in N \mid \epsilon(t_{v,k}) \in A_{n_y}} x^{\delta(t_{v,k})-\tau^{\Delta}}_{v,n_y,\epsilon(t_{v,k})} = 1, \qquad \forall v \in V, \forall k \in \mathbb{N}^{\leq y_v} \tag{5.17}$$

$$\sum_{v \in V} \sum_{n_y \in N \setminus FN} x^{\tau}_{v,n_x,n_y} \leq 1, \qquad \forall \tau \in \mathbb{T}^{<H}, \forall v \in V, \forall n_x \in N \tag{5.18}$$

$$\sum_{v \in V} (x^{\tau}_{v,n_y,n_x} + x^{\tau}_{v,n_x,n_y}) \leq 1, \qquad \forall \tau \in \mathbb{T}^{<H}, \forall v \in V, \forall n_x, n_y \in N \text{ with } n_x \in A_{n_y}, n_y \in A_{n_x} \tag{5.19}$$

Equation 5.14 specifies that a vehicle makes a step at each time stamp. Equation 5.15 requires a vehicle to travel to an adjacent node or stay in its current position at each time step. Equation 5.16 ensures the first step is always from the starting location. Equation 5.17 ensures all destination nodes must be visited at the respective transfer's deadline. Equation 5.18 restricts that only one vehicle can simultaneously be present in a node. Equation 5.19 restricts that vehicles cannot swap nodes.

Table 8 shows how the MP model relates to Definition 5.3.2. In Definition 5.3.2, a set of steps defines the movement for each transfer. Hence, constraints RC5 and RC7 ensure that one transfer transitions seamlessly into the next transfer. As the decision variable $x$ is defined for each time step, the transition from one transfer to the next occurs implicitly.

| Definition constraints | CP constraints |
|---|---|
| Input assumption: first transfer starts in $l$ | Equation 5.16 |
| RC1 | As the model uses time steps instead of spatial steps, sequencing is automatically guaranteed by Equation 5.14 |
| RC2 | Equation 5.15 |
| RC3 | Equation 5.17 |
| RC4 | As the model uses time steps, and assumes for all $v \in V$ and $k \in \mathbb{N}^{<y_v}$, $\delta(t_{v,k}) = \gamma(t_{v,k+1})$, this is automatically guaranteed. |
| RC5 | Equation 5.17 |
| RC6 | Equation 5.18 |
| RC7 | Equation 5.19 |

**Table 8:** Connections between constraints of Definition 5.3.2 and the MILP model variant 1.

## 5.5.3 | MP Variant 2

The second MP variant that we consider is the variant that has a binary variable indicating that a vehicle is present in a node at a certain time. The model is an adaptation of MP variant 1, presented in Section 5.5.2 and the MP model developed by Saidi-Mehrabad *et al.* [44]. As this model also exclusively uses binary decision variables, it is an ILP. The model is as follows:

*Variables*

| | |
|---|---|
| $N, E, V, \tau^{\Delta}, FN, y_v(v \in V), T_v(v \in V), \delta, \epsilon$ | As in Definition 4.4.1. Also derived notations are adopted. |

*Derived Variables*

| | |
|---|---|
| $A_n = \{a \in N \mid (n,a) \in E\} \cup \{n\}$ | The adjacent nodes for each node $n \in N$. |
| $H = \max_{t \in T}(\delta(t))$ | Time horizon, being the maximum deadline of all transfers $t \in T$. |
| $\mathbb{T} \subseteq \mathbb{N}_0$ | The discretized time domain. |

*Decision Variables*

$$x^{\tau}_{v,n} = \begin{cases} 1 & \text{if AGV } v \text{ is present in node } n \text{ at time } \tau \in \mathbb{T}^{\leq H} \\ 0 & \text{otherwise} \end{cases}$$

The program is constructed accordingly:

$$\sum_{n \in N} x^{\tau}_{v,n} = 1, \qquad \forall \tau \in \mathbb{T}^{\leq H}, \forall v \in V \tag{5.20}$$

$$x^{\tau}_{v,n_x} = \sum_{n_y \in N | n_x \in A_{n_y}} x^{\tau+\tau^{\Delta}}_{v,n_y}, \qquad \forall \tau \in \mathbb{T}^{\leq H-\tau^{\Delta}}, \forall v \in V, \forall n_x \in N \tag{5.21}$$

$$x^{0}_{v,l} = 1, \qquad \forall v \in V \tag{5.22}$$

$$x^{\delta(t_{v,k})}_{v,\epsilon(t_{v,k})} = 1, \qquad \forall v \in V, \forall k \in \mathbb{N}^{\leq y_v} \tag{5.23}$$

$$\sum_{v \in V} x^{\tau}_{v,n} \leq 1, \qquad \forall \tau \in \mathbb{T}^{\leq H}, \forall v \in V, \forall n \in N \setminus FN \tag{5.24}$$

$$x^{\tau+\tau^{\Delta}}_{v_x,n_x} + x^{\tau+\tau^{\Delta}}_{v_y,n_y} \leq 3 - x^{\tau}_{v_x,n_x} + x^{\tau}_{v_y,n_y}, \qquad \forall \tau \in \mathbb{T}^{\leq H-\tau^{\Delta}}, \forall v_x, v_y \in V, \forall n_x, n_y \in N \text{ with } n_x \in A_{n_y}, n_y \in A_{n_x} \tag{5.25}$$

Equation 5.20 specifies that a vehicle is present at a node in each time stamp. Equation 5.21 requires a vehicle to travel to an adjacent node or stay in its current position at each time step. Equation 5.22 ensures the first step is always from a vehicle's starting location. Equation 5.23 ensures all destination nodes must be visited at the respective transfer's deadline. Equation 5.24 restricts that only one vehicle can simultaneously be present in a node, excluding the free nodes. Equation 5.25 restricts that vehicles cannot swap nodes.

Table 9 shows how the MP model relates to Definition 5.3.2.

| Definition Constraints | CP constraints |
|---|---|
| Input assumption: first transfer starts in $l$ | Equation 5.22 |
| RC1 | As the model uses time steps instead of physical steps, sequencing is automatically guaranteed by Equation 5.20. |
| RC2 | Equation 5.21 |
| RC3 | Equation 5.23 |
| RC4 | As the model uses time steps, and assumes for all $v \in V$ and $k \in \mathbb{N}^{<y_v}$, $\delta(t_{v,k}) = \gamma(t_{v,k+1})$, this is automatically guaranteed. |
| RC5 | Equation 5.23 |
| RC6 | Equation 5.24 |
| RC7 | Equation 5.25 |

**Table 9:** Connections between constraints of Definition 5.3.2 and the MILP model variant 2.

## 5.6 | Model Comparison

We conclude this chapter by evaluating the performance of the discussed models' implementations. One of the models is to be used in a decomposition-based solution for the CFTFJSSP. Later, we see that the decomposition-based approach requires a lot of iterations in which new CFRP instances needs to be solved. Hence, it is of utmost importance that the chosen implementation and its respective solver can find solutions with a low computation time.

### 5.6.1 | Benchmark

A small benchmark, consisting of 54 problem instances, was generated to compare the three CP and ILP models regarding computation speed. The benchmark generates problem instances based on four parameters: routing network size, number of vehicles, number of transfers per vehicle, and deadline margin. For each transfer, a random destination node is assigned. The routing network is defined as a grid. For the MP models, this grid is transformed into a graph.

As we assume all steps to take up exactly a single time unit and do not consider obstacles in this benchmark, the respective time window for a transfer can be defined as the Manhattan distance between its source and destination node. In some cases, this is a very tight bound. It then leads to many infeasible solutions, as vehicles often cannot choose an alternative route within the defined time window when trying to avoid conflicts. This is where the deadline margin parameter comes into play. This parameter defines the additional time each transfer receives to complete the routing. Hence, the time window for each transfer is defined as the Manhattan distance plus the deadline margin.

The CP models are solved using the IBM ILOG CP Optimizer solver; all ILP models are solved using the IBM CPLEX Optimizer solver. All tests were conducted on an Intel i7-9750H 2.60 GHz processor with 16 GB RAM memory. The exact benchmark parameters are in Table 10. Each unique combination of one of the parameter values forms a problem instance in the benchmark. Each problem instance is tested on all the models (CP, MP1, MP2), with a time limit of 20 seconds for each instance.

| Parameter | Values | | |
|---|---|---|---|
| Routing network size (rows × columns) | $3 \times 3$ | $4 \times 4$ | $8 \times 8$ |
| Number of vehicles | 1 | 2 | 3 |
| Number of transfers per vehicle | 1 | 2 | 3 |
| Deadline margin | 1 | 3 | |

**Table 10:** Benchmark parameters for the CFRP

## 5.6.2 | Solution Quality

From the 54 benchmark problem instances, MP1 and MP2 found a solution to each instance, whereas the CP instance only obtained a solution for 19 of the 54 instances within the defined time limit. The solutions were verified to ensure no conflicts were present. To illustrate that the models indeed provide correct results, let us consider the solutions of experiment number 10 with a routing network size of $3 \times 3$ nodes, see also Figure 15. The following additional specifications apply:

$$\begin{aligned} V &= \{v_0, v_1\} \\ l &= 3 \\ FN &= \{l\} \\ T &= \{t_{0,1}, t_{0,2}, t_{0,3}, t_{1,1}, t_{1,2}, t_{1,3}\} \\ \boldsymbol{\delta} &= [5, 11, 16, 6, 13, 17] \\ \boldsymbol{\epsilon} &= [5, 6, 0, 2, 6, 7] \end{aligned}$$

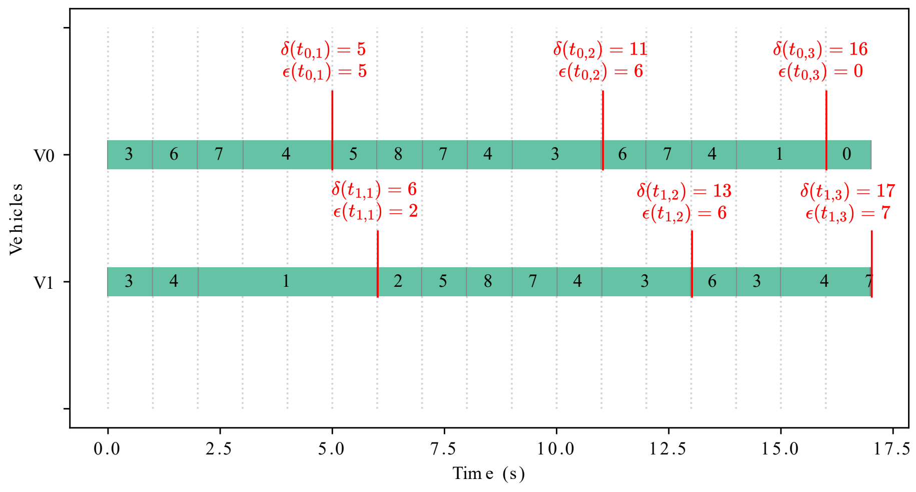
where, as before, $\boldsymbol{\delta}$ and $\boldsymbol{\epsilon}$ denote the deadline and destination functions $\delta, \epsilon$ in vector format. The found CP solution and the MP solutions can be seen in Figure 17.

Figure 17a shows the CP solution where each time interval shows the presence of a vehicle at a node in the routing network. Note that, while the CP model is programmed using a grid representation, the figure shows node numbering such that the representation is consistent in all three presented figures. Figure 17b shows the solution for MP variant 1. Each time interval represents a step from one node to an adjacent node. Figure 17c shows the solution for MP variant 2. Similar to the CP model, an interval shows the presence of a vehicle at a node in each time step. In comparison to the other solutions, the time horizon has been extended by a single time unit. This way we can see that indeed the vehicle does not leave the destination node 7 before the deadline.

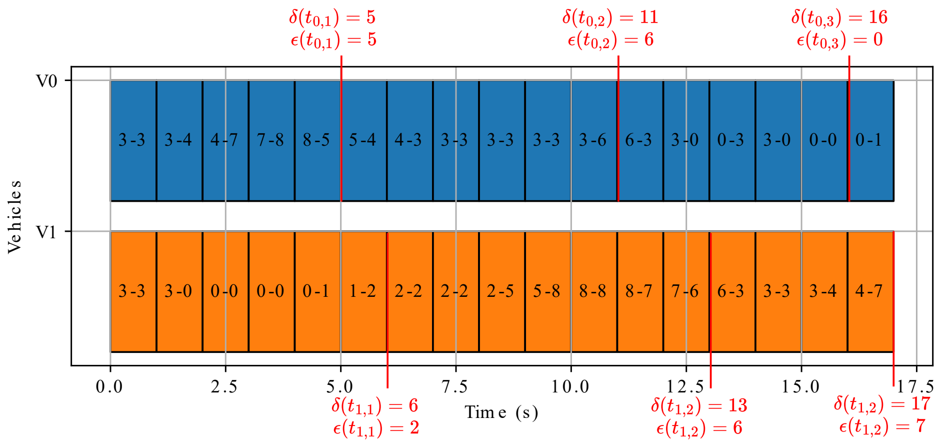
Observe that, indeed, all solutions arrive in the destination nodes at the respective deadlines. Furthermore, there are no two vehicles present in a node or cell at the same time. Lastly, it can be seen that no two vehicles ever swap their locations, meaning they are present on the same (bidirectional) edge at the same time. From these observations, we can conclude that the solution is indeed correct.

## 5.6.3 | Computational Results

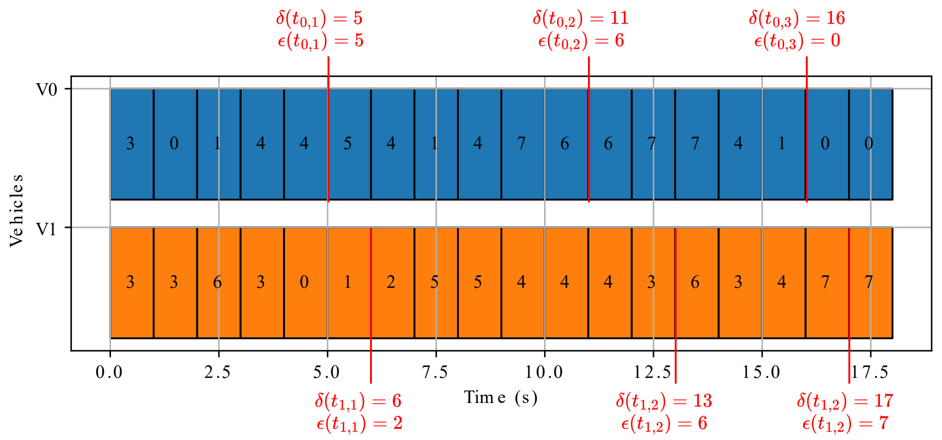
Let us compare all models in terms of their computation times. Figure 18 shows this comparison for each benchmark instance.



(a) CP solution.



(b) MP variant 1 solution.



(c) MP variant 2 solution.

Figure 17: CFR solutions for experiment 10.

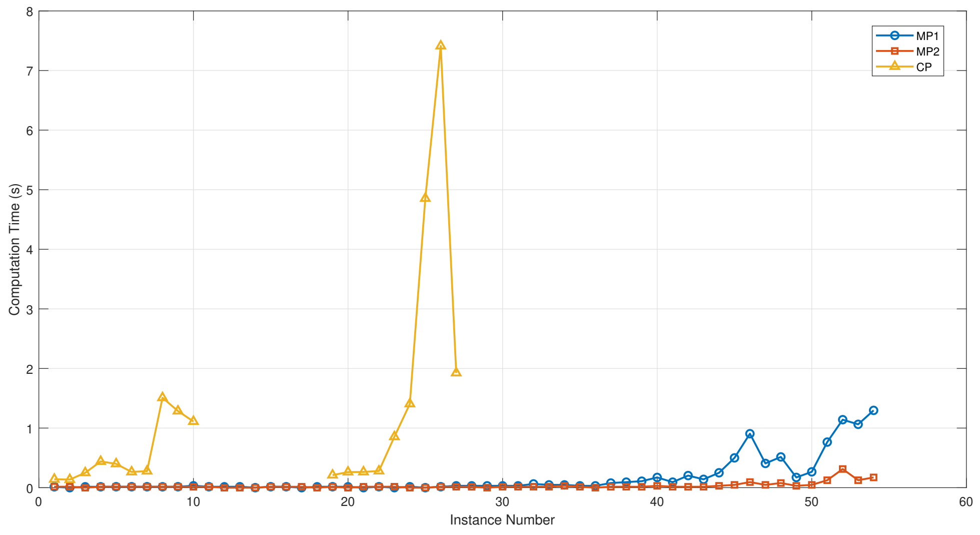


**Figure 18:** Computation time of each model for the benchmark instances

Observe that in almost all experiments, the CP model shows the worst results or was not able to find a solution at all within the given time limit. From this, we clearly conclude that the proposed CP model is not competitive for the CFRP compared to the suggested MPs.

### 5.6.4 | MP Comparison

In order to choose which MP variant shows the best performance for future implementation in the decomposed CFTFJSSP, we introduced a larger benchmark of 128 test instances. The key feature of this benchmark is to test the scalability of the presented implementations. Hence, more CFRP instances were introduced with bigger grid sizes, more vehicles, and more transfers per vehicle. The exact benchmark parameters are in Table 11. Each unique combination of one of the parameter values forms a problem instance in the benchmark.

| Parameter | Values | | | |
|---|---|---|---|---|
| Grid size (rows × columns) | $3 \times 3$ | $4 \times 4$ | $8 \times 8$ | $10 \times 10$ |
| Number of vehicles | 2 | 3 | 4 | 5 |
| Number of transfers per vehicle | 2 | 3 | 4 | 6 |
| Deadline margin | 1 | 3 | | |

**Table 11:** Benchmark parameters for the CFRP

The comparison of the computation times of both MP implementations can be seen in Figure 19.

The experiments shown were run from the smallest to the largest problem instance. While for the smaller problem instances, both models show comparative results, it is clear that MP variant 2 deals better when we scale up the problem instances. From this, we conclude that MP variant 2 is be the preferred model of choice for the decomposed CFTFJSSP model that we present later.

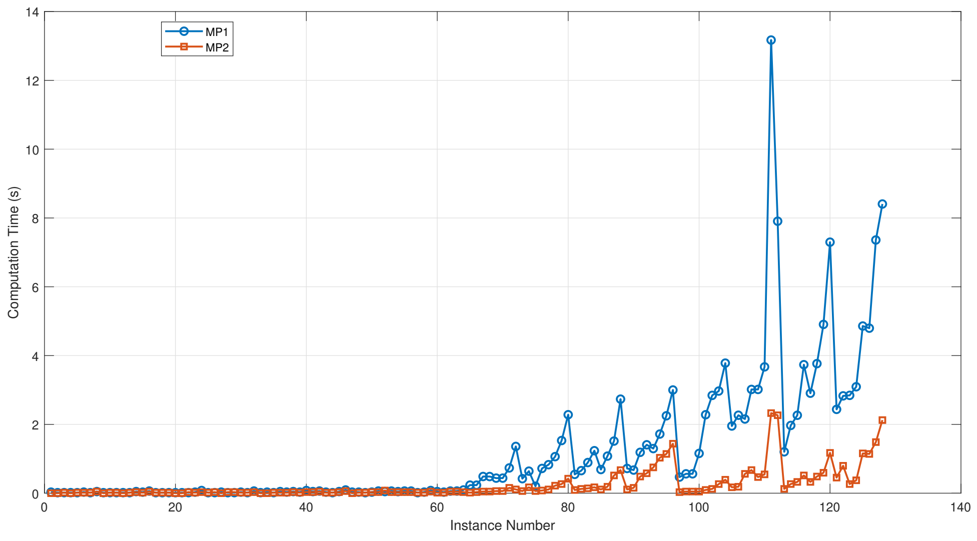


**Figure 19:** Computation time of the ILP models for the benchmark instances

# Chapter 6

# Conflict-Free-Transportation-Constrained Flexible Job-Shop Scheduling

The last step towards creating an exact decomposition-based solution for the CFTFJSSP is a clear definition of the problem itself. As the CFTFJSSP is a combination of the TFJSSP and CFRP, we can use the definitions from Chapter 4 and Chapter 5 to formalize CFTFJSSP. In this chapter, we provide this formalization and introduce a general-purpose definition of the CFTFJSSP.

## 6.1 | Introduction and State of the Art

In essence, we can describe the CFTFJSSP as a TFJSSP for which its transfer times need to be determined in such a way that the vehicles do not collide. As vehicles in the TFJSSP have no exact location awareness, the TFJSSP has to be extended with the notion of a routing network, as we see in the CFRP. Rather than constraining the minimum transfer time between two locations, the determination of the exact transfer times and steps then becomes part of the optimization problem.

This is where the TFJSSP and CFRP meet. Both problems consider the timing of a transfer as a decision variable. The TJFSSP solves the transfer times as a decision variable, in such a way that the transfer always takes up at least the minimal specified duration. The CFRP considers these starting and completion times as release times and deadlines and tries to find feasible routing assignments. By merging these two problems, we can see that by finding a feasible routing assignment, we can consider the found release times and deadlines as starting and completion times for the respective transfer.

Despite its relevance, there is little research related to the CFTFJSSP. This thesis took as reference the articles by Lyu *et al.* [31] and Liu *et al.* [29]. While both consider the CFTFJSSP, their exact definitions differ. This is well reflected in their considered routing networks.

The routing networks considered by Lyu *et al.* are grid-based graphs that exclusively consider horizontal or vertical adjacent cells. In Figure 9, we see an example routing network, where they introduce the notion of obstacles along the way. Furthermore, each layout marks a separate loading and unloading station. Hence, vehicles start and end at different locations.

Different from Lyu *et al.*, Liu *et al.* consider no obstacles in the routing network. They consider exclusively square grid network layouts. Moreover, vehicles can also travel to diagonal cells, meaning they consider horizontal, vertical, and diagonal adjacency. Furthermore, only one node marks the network's loading and unloading station. An example layout can be seen in Figure 20.

Our problem formulation given in this chapter is based on general routing graphs, covering both types of networks. Also the solution approach developed in the next chapter supports both types of networks.

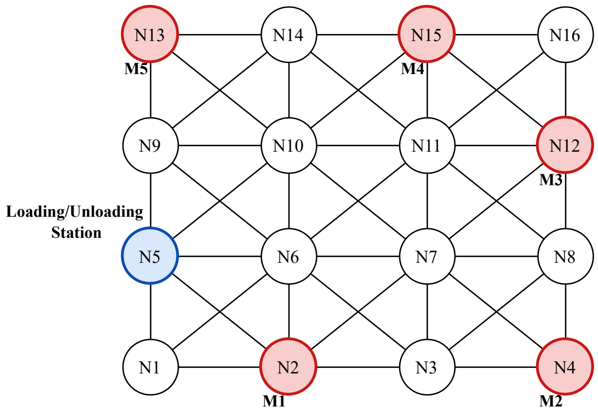


**Figure 20:** Example 4x4 layout of the benchmark by Liu *et al.* [29].

## 6.2 | An Illustrative Conflict-Free-Transportation-constrained Flexible Job Shop

Let us reconsider the example given in Section 5.2, and extend this conflict-free routing problem with a small job set. The jobs and their possible machine allocations are presented in Table 12.

| Job | Operation | $ls$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $us$ |
|---|---|---|---|---|---|---|---|
| $j_0$ | $o_{0,0}$ | 0 | - | - | - | - | - |
| | $o_{0,1}$ | - | 2 | 2 | - | - | - |
| | $o_{0,2}$ | - | - | - | 3 | 3 | - |
| | $o_{0,3}$ | - | - | - | - | - | 0 |
| $j_1$ | $o_{1,0}$ | 0 | - | - | - | - | - |
| | $o_{1,1}$ | - | 4 | 4 | - | - | - |
| | $o_{1,2}$ | - | - | - | 2 | 2 | - |
| | $o_{1,3}$ | - | - | - | - | - | 0 |

**Table 12:** Operation processing times on each machine.

The loading and unloading stations are denoted as $ls$ and $us$, respectively. The machines seen in the routing network of Figure 9 are denoted $m_1, \ldots, m_4$. Note that compared to the TFJSSP, there is no transition time matrix, as transfer times are determined by the CFRP. The combination of the jobs, their operations, their potential machine allocations, the number of available vehicles, and the routing network fully specify a CFTFJSSP instance. The optimization goal is now to find an allocation of operations to machines, resulting in required transfers, an allocation of these transfers to the available vehicles, and starting and completion times for operations, transfers, and steps within these transfers so that the overall makespan is minimized.

## 6.3 | Problem Definition

The definition of the conflict-free-transportation-constrained flexible job shop is given as follows:

**Definition 6.3.1.** A Conflict-Free-Transporation-constrained Flexible Job Shop (CFTFJS) is a tuple $(N, E, V, L, \tau^{\Delta}, J, r_j(j \in J), \mathcal{P})$, with the following elements:

| | |
|---|---|
| *Variables* | |
| $N$ | The set of nodes in the routing network. |
| $E \subseteq N^2$ | The set of edges connecting the nodes in the routing network, specifying valid steps from one node to another. We assume that, for all $n \in N$, $(n, n) \notin E$; that is, it is not possible to make a step without changing location. |
| $V:$ | The set of automated guided vehicles. |
| $L \subseteq N$ | The set of special locations, including the loading station $ls$, unloading station $us$, and machine locations. |
| $\tau^{\Delta} \in \mathbb{N}:$ | Duration of a single step. We assume a uniform step duration. |
| $J, r_j \in \mathbb{N}$ | The set of jobs, with every job $j \in J$ consisting of a sequence of $r_j + 2$ operations $\langle o_{j,0}, \ldots, o_{j,r_j+1} \rangle$ where operation $o_{j,k}$ denotes the $k$-th operation of the $j$-th job. Operation $o_{j,0}$ denotes the loading of materials at the loading station $ls$, and operation $o_{j,r_j+1}$ denotes the unloading of the finished product at the unloading station $us$. |
| *Derived Variables* | |
| $O = \{o_{j,k} \mid j \in J, k \in \mathbb{N}_0^{\leq r_j+1}\}$ | The set of operations including the loading and unloading operations for each job. |
| $FN = \{ls, us\}$ | The set of free nodes where more than one vehicle $v \in V$ may be present simultaneously, which are the loading and the unloading station. |
| *Functions* | |
| $\mathcal{P} : O \times L \hookrightarrow \mathbb{R}_0^+$ | Partial function denoting the processing time of an operation $o \in O$ at a location $l \in L$. A respective operation $o$ cannot be performed at the corresponding location $l$ when $\mathcal{P}(o, l)$ is undefined. For the operations $o_{j,0}$ and $o_{j,r_j+1}$ of any job $j \in J$, $\mathcal{P}$ is only defined for the loading station $ls$ and unloading station $us$, respectively. Processing times for operations other than $o_{j,0}$ and $o_{j,r_j+1}$ are assumed to be non-zero. |
| *Derived Variables (Continued)* | |
| $L_o = \{l \in L \mid \mathcal{P}(o, l) \text{ defined}\}$ | The set of locations that can perform operation $o$. |
| $LT = \{lt_{j,k} \in T \mid j \in J, k \in \mathbb{N}_0^{\leq r_j}\}$ | The set of loaded transfers where $lt_{j,k}$ is a shorthand notation for the loaded transfer between $o_{j,k}$ and $o_{j,k+1}$. |
| $ET = \{et_{j,k} \in T, \mid j \in J, k \in \mathbb{N}_0^{\leq r_j}\}$ | The set of empty transfers for each loaded transfer in $LT$, with $et_{j,k}$ forming the shorthand notation for the corresponding empty transfer for $lt_{j,k}$. The duration of this transfer can be 0 if a vehicle is already at the correct location. |
| $T = LT \cup ET$ | The complete set of all transfers. |

The above definitions use the following assumptions:

- Job ordering is arbitrary.
- Product materials are initially available at the loading station.
- The considered vehicles are a fleet of identical automated guided vehicles.
- All vehicles are available at the loading station at the start.
- Vehicles have unlimited range and do not need charging/maintenance.
- A vehicle must be present to support loading and unloading when the same machine processes any two consecutive operations of a job.
- Loading/unloading times are included in the transfer and/or processing times.
- Machines have sufficient buffer capacity.
- The routing network is a unidirectional graph.
- Only in the loading and unloading stations multiple vehicles can be present.
- Conflicts are situations where two vehicles are present in the same node (except $ls$ and $us$) or switch nodes at the same time instant.

This brings us to the definition of the schedule for the transportation-constrained flexible job shop:

**Definition 6.3.2.** A *schedule* $\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s$ for a CFTFJS describes the starting times, completion times, machine allocation, vehicle allocation, and steps with their starting times for all operations and transfers.

| | |
|---|---|
| *Decision Variables* | |
| $\mathcal{S}_{ot} : O \cup T \to \mathbb{R}_0^+$ | Complete function giving the starting time of an operation or transfer. |
| $\mathcal{C}_t : T \to \mathbb{R}_0^+$ | Complete function giving the completion time of a transfer. |
| $\alpha : O \to L$ | Complete function giving the allocation of an operation to a machine. |
| $\beta : T \to V$ | Complete function giving the allocation of a transfer to a vehicle. |
| $z_t \in \mathbb{N}_0$ | The number of steps in transfer $t \in T$. |
| *Derived Variables* | |
| $S = \{s_{t,i} \mid t \in T, i \in \mathbb{N}^{\leq z_t}\}$ | The set of all steps $s$, where $s_{t,i}$ is a shorthand notation for the $i$-th step in transfer $t \in T$. |
| *Decision Variables (Continued)* | |
| $\mathcal{N} : S \to N$ | Complete function giving the destination node of all steps. |
| $\mathcal{S}_s : S \to \mathbb{R}_0^+$ | Complete function giving the starting time of a step. |

*Derived Functions*

| | |
|---|---|
| $\mathcal{C}_o : O \to \mathbb{R}_0^+$ | Complete function giving the completion time of an operation, where for all $o \in O$, $\mathcal{C}_o(o) = \mathcal{S}(o) + \mathcal{P}(o, \alpha(o))$. |
| $\mathcal{C}_s : S \to \mathbb{R}_0^+$ | Complete function giving the completion time of a step, where for all $s \in S$, $\mathcal{C}(s) = \mathcal{S}(s) + \tau^{\Delta}$. |
| $Nxt_v : S \hookrightarrow S$ | For each vehicle $v \in V$, partial function $Nxt_v$ defines for each step of $v$, except the last one, the next step of that vehicle. Given a vehicle allocation $\beta$ and a sequencing of all transfers per vehicle (as enforced by the constraints specified below), $Nxt_v$ is well defined. |
| $\epsilon : T \to N$ | Function giving a transfer's destination node. Given machine and vehicle allocations $\alpha, \beta$, $\epsilon$ is well defined. |
| $\gamma : T \to \mathbb{R}_0^+$ : | Specification of transfer release times, where for all $t \in T$, $\gamma(t) = \mathcal{S}_{ot}(t)$. |
| $\delta : T \to \mathbb{R}_0^+$ : | Specification of the transfer deadlines, where $\delta = \mathcal{C}_t$. The release-time and deadline functions are only introduced to enable linking to the problem definitions in Chapter 4, Chapter 5. |
| $\mathcal{T} : L \times L \to \mathbb{R}_0^+$ | Shortest-path transfer times between any two locations in the routing network. These can be efficiently derived using a shortest-path algorithm. |

From Definition 4.4.2, constraints JC1 up to and including JC9 apply when using starting times $\mathcal{S}_{ot} : O \cup T \to \mathbb{R}_0^+$ instead of $\mathcal{S}$. From Definition 5.3.2, constraints RC1 up to and including RC7 apply when using starting times, $\mathcal{S}_s : S \to \mathbb{R}_0^+$ and completion times $\mathcal{C}_s : S \to \mathbb{R}_0^+$ instead of $\mathcal{S}$ and $\mathcal{C}$.

A schedule is feasible when all of these constraints are satisfied, and infeasible otherwise.

Lastly, the Conflict-Free-Transporation-constrained Flexible Job-Shop Scheduling Problem (CFTFJSSP) is a CFTFJS scheduling problem characterized by an optimization objective. In this research, we focus on a makespan objective, which is defined as:

**Definition 6.3.3** (makespan)**.** The makespan $C_{max}$ of a schedule $\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s$ is the latest completion time of the last *machine* operation of a job:

$$C_{max}(\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s) = \max_{j \in J}(\mathcal{C}_o(o_{j,r_j}))$$

**Definition 6.3.4** (conflict-free transportation-constrained job-shop scheduling problem)**.** The optimal solution for a Conflict-Free-Transporation-constrained Flexible Job-Shop Scheduling Problem (CFTFJSSP) is a feasible schedule $\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s$ with the smallest possible makespan $C_{max}(\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s)$.

**Example 6.3.1.**

Reconsider the illustrative example of Section 6.2, assuming two vehicles for transportation. Computing the shortest makespan solution based on the above definitions yields the schedule seen in Figure 21, where the unloading transfers and operations have been omitted (because they do not contribute to the makespan).

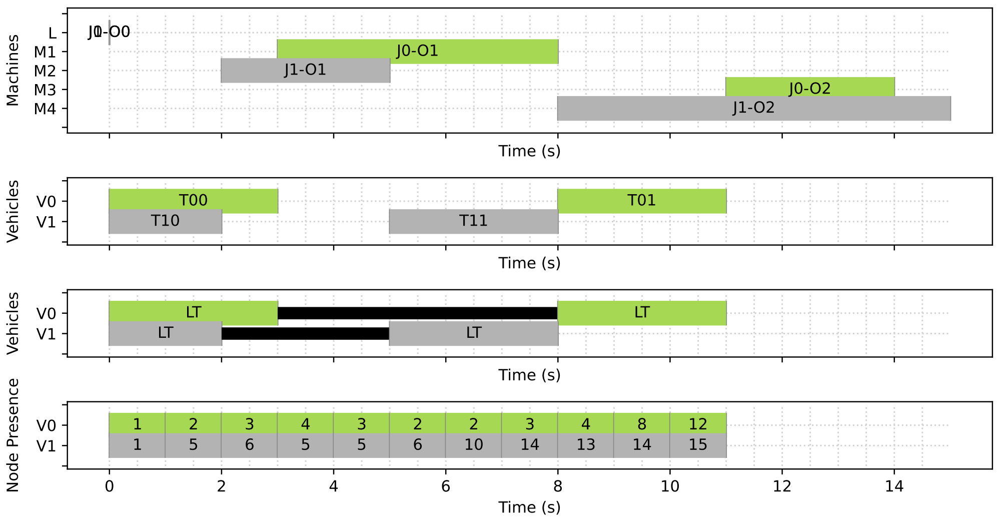


**Figure 21:** Optimal schedule for the illustrative CFTFJSSP instance

The first panel shows the operation scheduling, where `J0-O0` denotes operation $o_{0,0}$, and so on. The second and third panel show the loaded and empty transfers, respectively, where `Tvk` in the top panel denotes loaded transfer $lt_{v,k}$. The last panel shows the presence of a vehicle in a node, allowing us to verify the vehicle's chosen path. Together, the panels give the full schedule, consisting of the operation and transfer starting times $\mathcal{S}_{ot}$, the transfer completion times $\mathcal{C}_t$, the machine allocation $\alpha$, vehicle allocation $\beta$, number of steps $z_t$, destination nodes of steps $\mathcal{N}$, and the starting time of each step $\mathcal{S}_s$. Note that indeed the completion time of a transfer is the moment the vehicle arrives in the correct node for that machine. The makespan of the schedule is 15.

Dedicated conflict-free routing studies such as [24, 25] usually consider the makespan equal to the completion time of the last vehicle task. For the CFTFJSSP, this should be the completion time of the last loaded transfer to the unloading station. The reference literature for the CFTFJSSP [31, 29] reports the above makespan definition, though, taking the completion time of the last operation. However, some test results of [31, 29] report the makespan as the last completion time of the last transfer. Hence, let us introduce this alternate definition as well.

**Definition 6.3.5** (makespan*)**.** The makespan $C^*_{max}$ of a schedule $\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s$ is the latest completion time of any transfer (including the unloading operation):

$$C^*_{max}(\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, z_t, \mathcal{N}, \mathcal{S}_s) = \max_{j \in J}(\mathcal{C}_o(o_{j,r_j+1}))$$

**Example Continued.**

Consider again the illustrative example of Section 6.2 with two vehicles. Computing the shortest makespan solution as by Definition 6.3.5 yields the schedule seen in Figure 22.

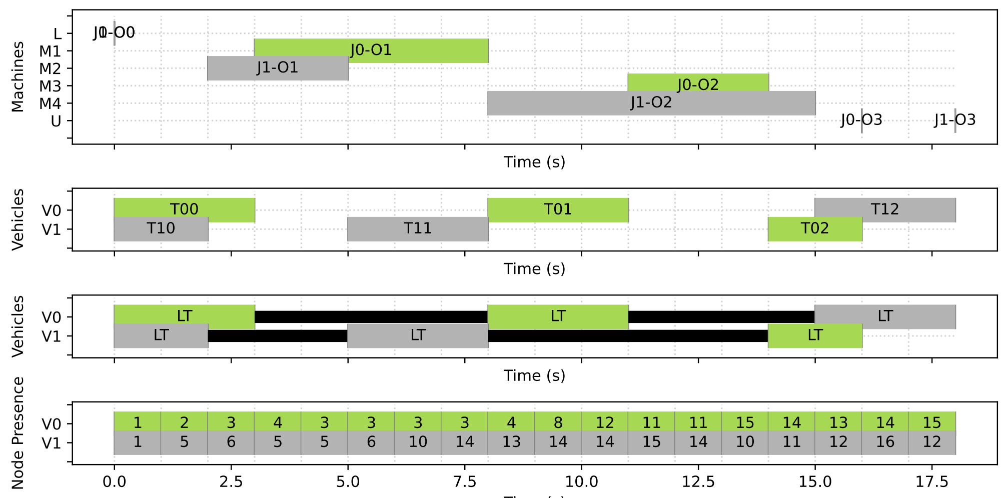


**Figure 22:** Optimal schedule for the illustrative CFTFJSSP instance taking into account the unloading transfers and operations

The figure shows us the same data as the previous example, this time for a different objective function. The unloading station is added, marked `U` in the top panel. The makespan increases to 18. Furthermore, more transfers are shown, which are needed to complete all jobs up to and including unloading.

# Chapter 7

# Logic-Based Benders Decomposition

Throughout this thesis, we have seen definitions for the TFJSSP and CFRP. We defined a model for various solving techniques and determined the best way to find exact solutions to these problems. Furthermore, a definition was given for the combined CFTFJSSP. In this chapter, we combine all these elements into a decomposition-based solution for the CFTFJSSP.

## 7.1 | Introduction

The Logic-Based Benders Decomposition (LBBD) [20] is a powerful method for solving complex optimization problems. Throughout this thesis, we have seen that the two foundational problems for CFTFJSSP, namely the TFJSSP and CFRP, are well-defined and can be solved as separate problems.

The discussed implementations of these problems can relatively easily obtain solutions for various benchmark instances. However, one can imagine that the overall combined problem has an immense complexity increase, often making it very hard to solve within reasonable time. The LBBD benefits from the problem structure, where we can relatively quickly solve the subproblems in order to obtain a solution for the overall problem.

Furthermore, we have seen in Chapter 4 and Chapter 5 that both problems can be best solved using different solving techniques. Where CP is the literature's favorite for scheduling problems, MP has been shown to work best for the routing problem. The LBBD method allows us to exploit the use of these different tools to cooperatively find the solution for the CFTFJSSP.

## 7.2 | Decomposition Setup

Let us define the CFTFJSSP described in Definition 6.3.4 as the overall problem, to be solved via an LBBD. The decomposition then follows with the TFJSSP from Definition 4.4.4 as the master problem and the CFRP from Definition 5.3.3 as the subproblem.

Observe that we can decompose the decision variables of a CFTFJSSP instance as defined in Definition 6.3.2 into a vector $\boldsymbol{x}$ for the TFJSSP decision variables (Definition 4.4.2) and a vector $\boldsymbol{y}$ for the CFRP decision variables (Definition 5.3.2). Let $\boldsymbol{\mathcal{S}_{ot}}, \boldsymbol{\mathcal{C}_t}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z_t}, \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}_s}$ denote the functions $\mathcal{S}_{ot}, \mathcal{C}_t, \alpha, \beta, \mathcal{N}, \mathcal{S}_s$ and variables $z_t$, in vector format. Then, the master problem decision variables are described as $\boldsymbol{x} = [\boldsymbol{\mathcal{S}_{ot}}, \boldsymbol{\mathcal{C}_t}, \boldsymbol{\alpha}, \boldsymbol{\beta}]$ and subproblem decision variables are $\boldsymbol{y} = [\boldsymbol{z_t}, \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}_s}]$. This gives us the following definitions:

**Definition 7.2.1** (Decomposed CFTFJSSP Master Problem)**.** The TFJSSP master problem for the decomposed CFTFJSSP considers the following decision variables:

*Master Problem Decision Variables*

| | |
|---|---|
| $\boldsymbol{\mathcal{S}_{ot}}$ | Vector of the starting times of all operations and transfers. |
| $\boldsymbol{\mathcal{C}_t}$ | Vector of the completion times of all transfers. |
| $\boldsymbol{\alpha}$ | Vector of the allocation of all operations on machines. |
| $\boldsymbol{\beta}$ | Vector of the allocation of all transfers on vehicles. |

Furthermore, the TFJSSP master problem considers constraint set $C(\boldsymbol{x})$, with elements JC1-JC9 from Definition 4.4.2.

**Definition 7.2.2** (Decomposed CFTFJSSP Subproblem)**.** The CFRP subproblem for the decomposed CFTFJSSP considers the following decision variables:

*Subproblem Decision Variables*

| | |
|---|---|
| $\boldsymbol{z_t}$ | The vector of the number of steps for all transfers |
| $\boldsymbol{\mathcal{N}}$ | The vector of all destination nodes for steps. |
| $\boldsymbol{\mathcal{S}_s}$ | Vector of the starting times of all steps. |

Furthermore, the CFR subproblem considers constraint set $C(\boldsymbol{y})$ consisting of the constraints RC1-RC2 from Definition 5.3.2 and constraint set $C(\boldsymbol{x}, \boldsymbol{y})$ with constraints RC3-RC7 from Definition 5.3.2.

The goal is to optimize the CFTFJSSP for makespan as specified by Definition 6.3.3, which can be adapted to the vector notations $\boldsymbol{x}, \boldsymbol{y}$:

$$C_{max}(\boldsymbol{x}, \boldsymbol{y}) = \max_{j \in J}(\mathcal{C}_o(o_{j,r_j})) \tag{7.1}$$

Note that the makespan is characterized by the function $\mathcal{C}_o$ which can be derived exclusively using decision variables in $\boldsymbol{x}$. Hence, we can alternatively denote the objective function as:

$$C_{max}(\boldsymbol{x}) = \max_{j \in J}(\mathcal{C}_o(o_{j,r_j})) \tag{7.2}$$

The makespan objective function from Equation 7.2 is only characterized by $\boldsymbol{x}$, the master problem. This results in the subproblem having no objective function. Hence, the chosen subproblem is a feasibility subproblem.

The decomposition process, as outlined in Algorithm 2, operates as follows. First, machines and vehicles are allocated in the master problem, the TFJSSP, to operations and transfers (both empty and loaded). Further, the starting and completion times for all operations and transfers are determined. The resulting solution defines the trial values $\boldsymbol{x}^k$ in iteration $k$ of the process. The aim of the subproblem is then to find a feasible vehicle routing assignment based on these trial values. If a feasible vehicle routing assignment is found, the solutions from the master problem and the subproblem together form the optimal solution for the overall problem.

## 7.3 | Benders Cuts

The last but most important step in realizing the LBBD for the CFTFJSSP is to determine a strong feasibility cut. For the CFRP subproblem, an ideal feasibility cut derived from an infeasible CFRP instance ensures that the master problem disregards as many trial values that lead to infeasibility as possible. For this section, let us evaluate various possible feasibility cut designs. In all examples, we denote the trial values in iteration $k$ as $\boldsymbol{x}^k = [\boldsymbol{\mathcal{S}}_{\boldsymbol{ot}}^k, \boldsymbol{\mathcal{C}}_{\boldsymbol{t}}^k, \boldsymbol{\alpha}^k, \boldsymbol{\beta}^k]$ with:

| | |
|---|---|
| $\boldsymbol{\mathcal{S}}_{\boldsymbol{ot}}^k$ | Vector of the starting times of all operations and transfers in iteration $k$. |
| $\boldsymbol{\mathcal{C}}_{\boldsymbol{t}}^k$ | Vector of the completion times of all transfers in iteration $k$. |
| $\boldsymbol{\alpha}^k$ | Vector of the allocation of all operations on machines in iteration $k$. |
| $\boldsymbol{\beta}^k$ | Vector of the allocation of all transfers on vehicles in iteration $k$. |

## 7.3.1 | A Simple Feasibility Cut

Defining strong cuts often requires some complex analysis and creativity. However, simple feasibility cuts can be designed without complex analysis.

The simplest feasibility cut simply disregards trial values that produced an infeasible result in past iterations. Such a cut—often called a *nogood* cut—is very weak in terms of how it affects the convergence rate towards the optimal solution.

We can strengthen our nogood cut with some basic evaluation of the subproblem at hand. In the case of the CFTFJSSP, we know an infeasible solution occurs when the subproblem cannot find a feasible routing assignment within the defined transfers' release times and deadlines. With this knowledge, we can derive that an infeasible set of trial values $\boldsymbol{x}^k$ might result in a feasible solution if a certain transfer time is increased. This means that either the release time is earlier or the deadline is later. This is what Corréa *et al.* [9] noticed for their combined scheduling and conflict-free routing problem, which can be seen as a simplified CFTJSSP. Hence, they introduced a feasibility cut that either completely disregards the trial values $\boldsymbol{x}^k$ or forces to change some variables $\boldsymbol{x}^k$ to result in a feasible solution. Later, [42] applied these same cuts for a similar problem variant.

For designing a straightforward feasibility cut for the CFTFJSSP, we can apply the same concepts as used by [9, 42]. Hence, a simple feasibility cut in iteration $k$ for the CFTFJSSP is given as the *disjunction* of the following options:

- Find a completely new arbitrary solution (nogood cut)

$$F_1^k(\boldsymbol{x}) := \boldsymbol{x} \neq \boldsymbol{x}^k \tag{7.3}$$

- Fixate all allocations and operation and transfer ordering but increase at least one transfer in length

$$\begin{gathered} F_2^k(\boldsymbol{x}) := \\ \alpha = \alpha^k \wedge (\forall o_1, o_2 \in O : \mathcal{S}^k(o_1) \leq \mathcal{S}^k(o_2) \implies \mathcal{S}(o_1) \leq \mathcal{S}(o_2)) \wedge \\ (\forall t_1, t_2 \in T : \mathcal{S}^k(t_1) \leq \mathcal{S}^k(t_2) \implies \mathcal{S}(t_1) \leq \mathcal{S}(t_2))) \wedge \\ \big(\exists t \in T : \mathcal{C}(t) - \mathcal{S}(t) > \mathcal{C}^k(t) - \mathcal{S}^k(t)\big) \end{gathered} \tag{7.4}$$

The cut is then defined as:

$$F_3^k(\boldsymbol{x}) := F_1^k(\boldsymbol{x}) \vee F_2^k(\boldsymbol{x}) \tag{7.5}$$

The resulting cut provides a suggestion of how an infeasible $\boldsymbol{x}^k$ can be transformed to result in a feasible $\boldsymbol{x}$. While this makes the cut stronger than the nogood cut $F_1^k(\boldsymbol{x})$, it still is limited in its impact on reducing the algorithm's solving time.

## 7.3.2 | Conflict-Driven Feasibility Cuts

Simple feasibility cuts are merely based on the observation that a subproblem has an infeasible schedule. Such cuts can only suggest that some variables should change in order to obtain a feasible schedule for the overall problem. However, there is no guarantee that the changed variables are related to the infeasibility of the subproblem schedule. Hence, the conflict may reoccur in the next iteration. A conflict-driven feasibility cut, on the other hand, uses extra information from an infeasible subproblem schedule to identify which variables caused the infeasibility in the first place. Hence, rather than changing any variable, the cut can directly enforce changes to prevent the conflict from reoccurring.

## Weak Conflict-Driven Feasibility Cuts for the CFTFJSSP

In the CFTFJSSP, an infeasible subproblem schedule occurs if there exists an unavoidable conflict. Conflicts occur when two or more vehicles visit the same node (presence conflict) or swap nodes simultaneously (swap conflict). The unavoidable conflict can be resolved by avoiding the specific steps that led to it. The CFTFJSSP implementation in this research has the capability to obtain the needed information about steps leading to a conflict. We can eliminate solutions with the same conflict structure by applying a cut directed at these steps.

To illustrate these conflict-driven feasibility cuts without verbose notations, let us first introduce a useful transfer notation for the remainder of this chapter:

| | |
|---|---|
| $t_{o,v,sl,dl,s,c}$ | Variable denoting a transfer for operation $o \in O$, which is performed by vehicle $v$ and starts from source location $sl \in L$, moves to the destination location $dl \in L$, with starting time $s \in \mathbb{R}_0^+$ and completion time $c \in \mathbb{R}_0^+$. |

**Example 7.3.1.**

The example is taken from the CFTFJSSP instance with job set 3, three vehicles, and routing network layout 2 as presented by Lyu *et al.* [31]. The FMS layout of this CFTFJSSP instance was already shown in Figure 9. Solving the TFJSSP master problem results in trial values $\boldsymbol{x}^1$. A fragment of the schedule corresponding to $\boldsymbol{x}^1$ can be seen in Figure 23.

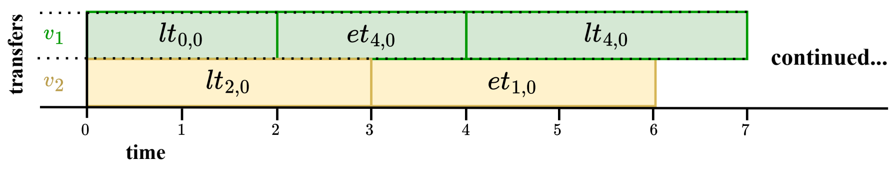


**Figure 23:** Gannt chart showing the transfer schedule results from iteration 1.

Using the notations introduced earlier, $lt_{0,0} = t_{o_{0,0},1,ls,m_2,0,2}$, $et_{4,0} = t_{o_{4,0},1,m_2,ls,2,4}$, $lt_{4,0} = t_{o_{4,0},1,ls,m_4,4,7}$, $lt_{2,0} = t_{o_{2,0},2,ls,m_4,0,3}$ and $et_{1,0} = t_{o_{1,0},2,m_4,ls,3,6}$, where $ls, m_2, m_4 \in L$ denote the loading station, machine 2 and machine 4, respectively. Note that $et_{0,0} = t_{o_{0,0},ls,ls,0,0}$ and $et_{2,0} = t_{o_{2,0},ls,ls,0,0}$ that are also part of schedule $\boldsymbol{x}^1$ are not shown in the figure as their respective transfer times are 0.

The CFR subproblem now needs to find a schedule $\boldsymbol{y}$ for the given trial values $\boldsymbol{x}^1$. For each transfer, there can only be a limited number of steps $z_t$ to meet the respective transfer's deadline. This limits the routing assignments—the sequences of steps and their destination nodes $\mathcal{N}$ through the routing network.

| Transfer | Release time | Deadline | Start Node | End Node | Routing Assignments |
|---|---|---|---|---|---|
| $lt_{0,0} = t_{o_{0,0},1,1,6,0,2}$ | 0 | 2 | 1 | 6 | $1 \to 2 \to 6$<br>$1 \to 5 \to 6$ |
| $et_{4,0} = t_{o_{4,0},1,6,1,2,4}$ | 2 | 4 | 6 | 1 | $6 \to 2 \to 1$<br>$6 \to 5 \to 1$ |
| $lt_{4,0} = t_{o_{4,0},1,1,13,4,7}$ | 4 | 7 | 1 | 13 | $1 \to 5 \to 9 \to 13$ |
| $lt_{2,0} = t_{o_{2,0},2,1,13,0,3}$ | 0 | 3 | 1 | 13 | $1 \to 5 \to 9 \to 13$ |
| $et_{1,0} = t_{o_{1,0},2,13,1,3,6}$ | 3 | 6 | 13 | 1 | $13 \to 5 \to 9 \to 1$ |

Transfers $lt_{4,0}, lt_{2,0}, et_{1,0}$ have only one step-assignment option given their respective release times and deadlines.

Consider a *possible, partial* assignment of steps for the considered transfers shown in Figure 24, where steps for vehicle 1 are shown in green and steps for vehicle 2 in yellow. The steps for $lt_{0,0}$ are omitted and the routing assignment for $et_{4,0}$ is chosen as $6 \to 2 \to 1$. The CFR subproblem is infeasible as an unavoidable presence conflict occurs in node 5. The conflict can be avoided if we prevent these *conflicting steps* and their respective timing from reoccurring. This can only be done by avoiding the related transfers in the master problem.

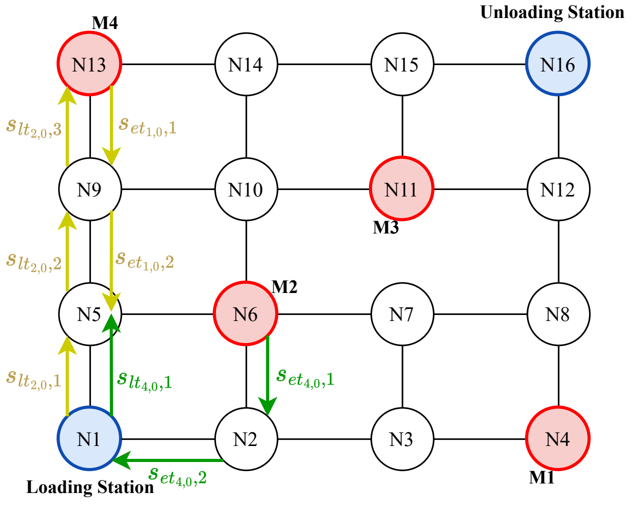


**Figure 24:** Routing network graph showing the steps in the subproblem tried at iteration 1. These steps unavoidably lead to infeasibility.

Let $Q^k$ be a set of conflict steps obtained in iteration $k$ of the subproblem. Set $Q^k$ shows us the steps to avoid when solving the subproblem with the given input constraints. Since our cut needs to be phrased in terms of master problem decision variables $\boldsymbol{x}$, we specify the set of conflicting *transfers* that caused the subproblem conflict in iteration $k$.

$$QT^k = \{t \in T \mid \exists i \in \mathbb{N}^{\leq z_{qt}} : s_{t,i} \in Q^k\} \tag{7.6}$$

**Example Continued.**

Reconsidering Example 7.3.1, we can define the set of conflicting transfers as:

$$QT^1 = \{t_{o_{4,0},1,m_2,ls,2,4}, t_{o_{4,0},1,ls,m_4,4,7}, t_{o_{2,0},2,ls,m_4,0,3}, t_{o_{1,0},2,m_4,ls,3,6}\} \tag{7.7}$$

To ensure the cut does not reoccur, we must guarantee a way to exclude this set $QT^1$ from reoccurring.

We can prohibit the occurrence of a specific set of conflicting transfers $QT$ for this vehicle assignment in a conflict-driven feasibility cut:

$$F_4^k(\boldsymbol{x}) := QT^k \not\subseteq T \tag{7.8}$$

### Strengthened Conflict-Driven Feasibility Cut

While the cut from Equation 7.8 makes sure that at least one of the transfers in set $QT^k$ cannot reoccur, we can still find a similar set $QT^{k+1}$ in the next iteration that causes an identical conflict, namely by having transfers between the same set of locations with the same release times and deadlines, but for different vehicles and/or operations. By strengthening the cut, we can exclude those similar conflict situations that can be derived from the set $QT^k$.

**Example Continued.**

The vehicles in a CFTFJS as defined in Definition 6.3.4 are all identical vehicles and all start at the exact same location $ls \in L$. Hence, when allocating conflicting transfers in $QT^k$ to different vehicles, the conflict in the subproblem persists. For instance, when assigning $lt_{2,0}$ and $et_{1,0}$ as shown in Figure 23 to vehicle $v_3$ instead of $v_2$, the same conflict occurs. With the same vehicle assignment, locations, and starting times and completion times but different operations, the conflict also re-occurs. For instance, the set of transfers

$$\{t_{o_{3,0},1,m_2,ls,2,4}, t_{o_{3,0},1,ls,m_4,4,7}, t_{o_{2,0},0,ls,m_4,0,3}, t_{o_{1,0},0,m_4,ls,3,6}\} \tag{7.9}$$

where operation 0 of job 3 takes the role of operation 0 of job 4 and vehicle 0 the role of vehicle 2, leads to essentially the same subproblem conflict.

Let $f_o : O \leftrightarrow O$ and $f_v : V \leftrightarrow V$ denote bijections on the set of operations $O$ and the set of vehicles $V$, respectively. Functions $f_o$ and $f_v$ may be seen are renamings of operations and vehicles. For a given $f_o$, $f_v$ and $QT^k$, a similar (isomporhic) set of conflicting transfers $SQT^k$ can be derived:

$$SQT^k(f_o, f_v) = \{t_{f_o(o),f_v(v),sl,dl,s,c} \mid t_{o,v,sl,dl,s,c} \in QT^k\} \tag{7.10}$$

A strengthened feasibility cut excludes all sets of conflicting transfers that are isomorphic to the original set $QT^k$.

$$F_5^k(\boldsymbol{x}) := (\forall f_o \in O \leftrightarrow O : \forall f_v \in V \leftrightarrow V : SQT^k(f_o, f_v) \not\subseteq T) \tag{7.11}$$

Note that this cut includes set $QT^k$, since the identity functions on $O$ and $V$ are also bijections.

For all presented cuts in this section, it holds that the cut from iteration $k$ is infeasible if trial values $\boldsymbol{x}^k$ are applied. Therefore, all cuts adhere to property **FCP1.** given in Section 3.5.4. From this, we can conclude that all the presented feasibility cuts qualify as valid Benders cuts.

The presented decomposition setup in Section 7.2 and the valid Benders cuts presented in this section form an exact solution technique for solving the CFTFJSSP. For the remainder of this thesis, we refer to this exact solution as the *LBBD-based CFTFJSSP*.

### Time-Shifted Conflict-Driven Feasibility Cut

As a final remark, one may also observe that besides different vehicles and operations, a similar conflict can be found anywhere in time as long as their relative starting times and completion times are shifted by an exact amount.

**Example Continued.**

Reconsider the set of transfers seen in Equation 7.9. Let us assume all transfers are shifted by time $\Delta$. This leads to a similar set of transfers

$$\{t_{o_{3,0},1,m_2,ls,2,4}, t_{o_{3,0},1,ls,m_4,4,7}, t_{o_{2,0},3,ls,m_4,0,3}, t_{o_{1,0},3,m_4,ls,3,6}\} \tag{7.12}$$

$$\Downarrow$$

$$\{t_{o_{3,0},1,m_2,ls,2+\Delta,4+\Delta}, t_{o_{3,0},1,ls,m_4,4+\Delta,7+\Delta}, t_{o_{2,0},3,ls,m_4,0+\Delta,3+\Delta}, t_{o_{1,0},3,m_4,ls,3+\Delta,6+\Delta}\} \tag{7.13}$$

that once more leads to essentially the same subproblem conflict.

We have not found a means to implement this in our CP solution for the TFJSSP master problem, discussed in the next section. Hence, we leave it for future work to implement this strengthened cut into the CFTFJSSP implementation.

## 7.4 | Implementation

The implementation of the LBBD-based CFTFJSSP follows straightforwardly using the solving techniques applied throughout earlier chapters. For the TFJSSP master problem, we use the CP model presented in Section 4.5. The CFRP subproblem is solved using MP variant 2 presented in Section 5.5.3. Our implementation uses the IBM CP Optimizer and IBM CPLEX Optimizer solvers for solving the constraint programming and mathematical programming models, respectively. The LBBD algorithm completing the LBBB-based CFTFJSSP implementation is captured in the software tooling that is discussed later in Chapter 8. Upon solving the CP master problem, the software tool extracts the trial values $\boldsymbol{x}^k$ from the CP master problem's solution, and passes them along as input to the MP subproblem. Upon an infeasible subproblem result, either the simple feasibility cut $F_3$ or the conflict-driven feasibility cut $F_5$ is applied. The choice of cut is performed in a clever way to improve convergence speed while ensuring optimality of LBBD-based CFTFJSSP.

### 7.4.1 | Implementing Conflict-Driven Feasibilty Cuts

Applying conflict-driven cuts requires us to extract the conflicting steps from the subproblem solution. When an optimization problem yields an infeasible solution, the solver typically does not provide any information about why it happened. However, IBM's CPLEX Optimizer solver, which is used for MP subproblem implementation, can, in fact, address constraint bottlenecks. The solver includes a conflict refiner tool that helps identifying the variables responsible for the infeasible result. The CFRP subproblem uses a binary decision variable $x^{\tau}_{v,n}$ to indicate the presence of a vehicle $v$ at a node $n$ at time $\tau$. When an infeasible result occurs, the conflict refiner determines which $x^{\tau}_{v,n}$ valuations caused the infeasibility, forming our set of conflicting steps $Q^k$. From here, it is straightforward to proceed with the remaining steps outlined in the previous section in order to specify the conflict-driven cuts for our master CP problem.

While the conflict refiner allows us to apply conflict-driven cuts, in some cases, the conflict-refining process has been shown to take up a lot of time. The conflict-driven cuts form the heart of the LBBD-based CFTFJSSP enabling fast convergence towards an optimal solution. However, for some trial values $\boldsymbol{x}^k$ the conflict refiner might have difficulties determining the set of conflicting steps $Q^k$. In these situations, it is a better idea to apply a simple feasibility cut in order to obtain a new set of trial values $\boldsymbol{x}^k$. With this new set of trial values, the conflict refiner might find the set $Q^k$ relatively fast, therefore leading to faster iterations of the LBBD-based CFTFJSSP. The implementation of this mechanism is realised by introducing a timeout for the conflict refiner. For each infeasible subproblem result, after a certain time, the conflict-refinement process is aborted, and the LBBD resorts to the simple feasibility cut from Equation 7.5.

Using the conflict refiner brings some additional risks. There is little information available on how IBM's CPLEX Conflict Refiner operates. Hence, if we want to ensure that the LBBD-based CFTFJSSP provides exact solutions, we need to guarantee that the conflict refiner does not occasionally make any mistakes. To make sure the correctness of the conflict refiner result is guaranteed, a checking mechanism is applied. The checking mechanism verifies whether the found set of conflicting transfers, indeed, always results in an infeasible subproblem solution. To obtain this verification, an additional CFRP MP model is introduced. This CFRP MP model takes the set of conflicting transfers given by Equation 7.6 as input and computes the CFRP without the other transfers that are part of $\boldsymbol{x}^k$. If the infeasibility remains, we can conclude that the set of conflicting transfers is indeed a valid set of conflicting transfers. When this is not the case, the LBBD procedure settles on once again using the simple feasibility cut from Equation 7.5.

## 7.4.2 | Overview of the LBBD implementation

The entire procedure of the LBBD-based CFTFJSSP implementation is captured in Figure 25. The proce-

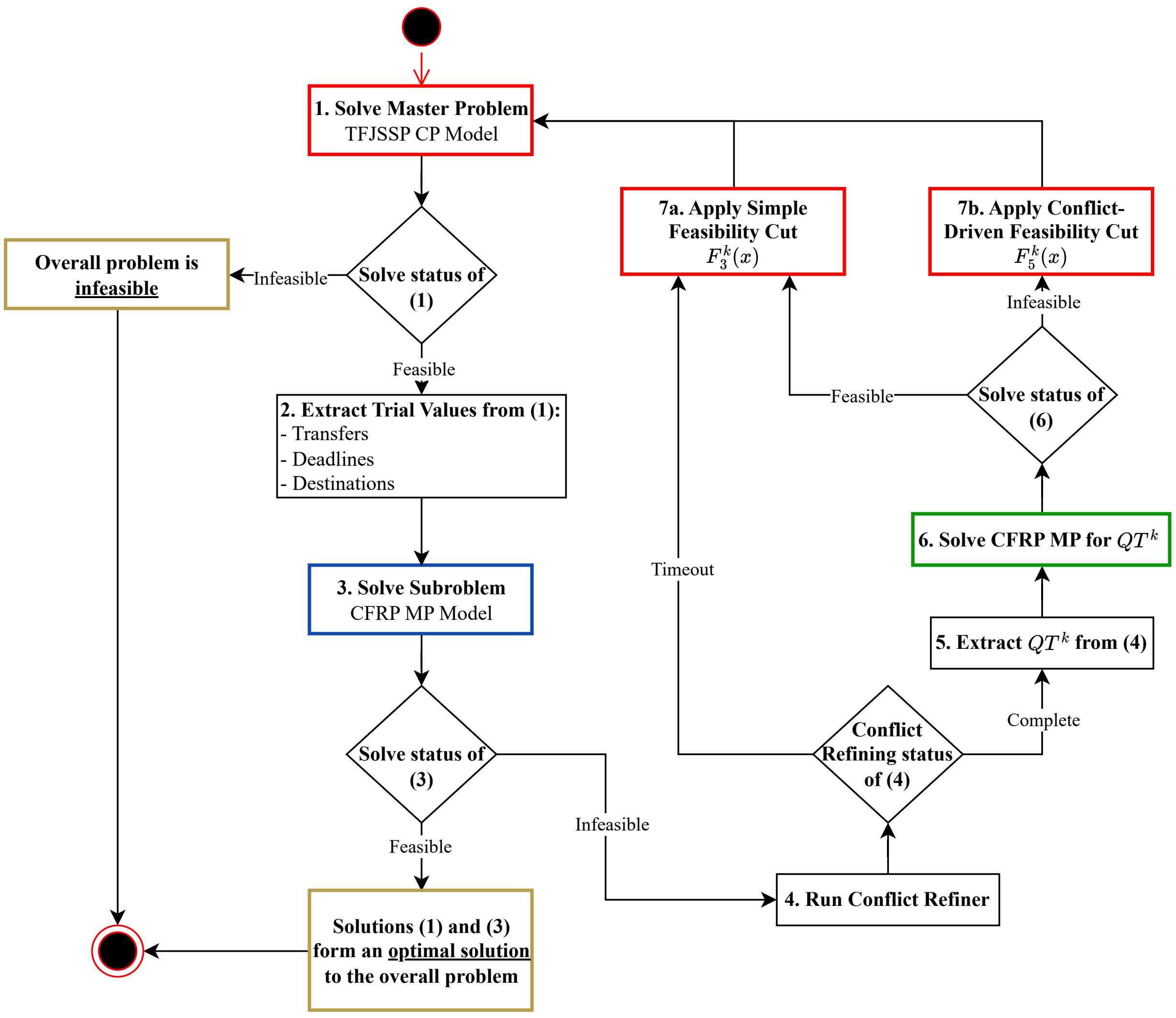

1. Solve Master Problem
TFJSSP CP Model
Solve status of (1)
Infeasible
Overall problem is infeasible
Feasible
2. Extract Trial Values from (1):
- Transfers
- Deadlines
- Destinations
3. Solve Subroblem
CFRP MP Model
Solve status of (3)
Feasible
Infeasible
Solutions (1) and (3) form an optimal solution to the overall problem
4. Run Conflict Refiner
Conflict Refining status of (4)
Timeout
Complete
5. Extract $QT^k$ from (4)
6. Solve CFRP MP for $QT^k$
Solve status of (6)
Feasible
Infeasible
7a. Apply Simple Feasibility Cut
$F_3^k(x)$
7b. Apply Conflict-Driven Feasibility Cut
$F_5^k(x)$

**Figure 25:** Flowchart showing the procedure of the LBBD-based CFTFJSSP implementation.

dure starts with step 1, which solves the TFJSSP CP master problem. When a feasible TFJSSP schedule is found, we can extract the transfers, their deadlines, and destination nodes from the master problem solution and feed them to the CFRP MP subproblem (step 2). Hereafter, in step 3, the subproblem can be solved. For an infeasible subproblem solution, we attempt to apply a conflict-driven feasibility cut. Hence, step 4 starts running the conflict refiner to obtain the set of conflicting steps $Q^k$. If the conflict refiner times out, we apply the simple feasibility cut, as by step 7a, and restart the process. If, on the other hand, a set $Q^k$ is found, we can obtain the set of conflicting transfers $QT^k$ (step 5). After checking if the set $QT^k$ is indeed a conflicting set of transfers (step 6), we can apply the conflict-driven feasibility cut (step 7b).

As mentioned before, the convergence speed of this LBBD-based CFTFJSSP is largely affected by the use of the conflict refiner in order to construct the conflict-driven feasibility cuts. While it is still uncommon to use these kinds of explanation mechanisms of solvers to devise cuts, Hooker [21] mentioned that these kinds of explanation mechanisms are very promising as they provide new types of powerful Benders cuts. This opens up new opportunities for research within LBBD, from which we only scratched the surface with the LBBD-based CFTFJSSP.

### 7.4.3 | Additions to the Master Problem

Determining which combinations of transfer starting and completion times in a CFTFJSSP lead to a routing conflict is complicated. This motivates the use of an LBBD. However, there are some distinguishable situations that lead to conflicts in any CFTFJSSP. It is beneficial to exclude such situations in the specification of the master problem in the LBBD (by adding extra constraints).

The TFJSSP master problem, as specified by Definition 4.4.4, does not consider the notion of a conflict at all. The constraints allow multiple vehicles to be at the same location at the same time. It is not straightforward to formulate constraints on the presence of vehicles in locations. But it is possible to exclude specific situations, in particular, the cases that multiple transfers end at the same location at the same time and that transfers for different operations and vehicles complete and start at the same location at the same time:

$$\begin{aligned}
&\boldsymbol{end\_of}(vehicle\_transfer\_options_{o_1,v_1,sl_1,dl}) \neq && \text{For all } o_1, o_2 \in O \text{ with } o_1 \neq o_2, && (7.14)\\
&\quad \boldsymbol{end\_of}(vehicle\_transfer\_options_{o_2,v_2,sl_2,dl}) && \quad v_1, v_2 \in V, \text{ with } v_1 \neq v_2, &&\\
& && \quad sl_1 \in L_{o_1}, sl_2 \in L_{o_2} \text{ and } dl \in L_{o_1} \setminus FN &&\\
&\boldsymbol{start\_of}(vehicle\_transfer\_options_{o_1,v_1,sl_1,dl_1}) \neq && \text{For all } o_1, o_2 \in O \text{ with } o_1 \neq o_2, && (7.15)\\
&\quad \boldsymbol{end\_of}(vehicle\_transfer\_options_{o_2,v_2,sl_2,dl_2}) && \quad v_1, v_2 \in V, \text{ with } v_1 \neq v_2, \text{ and} &&\\
& && \quad sl_1, dl_1 \in L_{o_1}, sl_2, dl_2 \in L_{o_2} \text{ with} &&\\
& && \quad sl_1 = dl_2, dl_2 \notin FN &&
\end{aligned}$$

Lastly, there is another interesting situation that can be avoided. Let us consider once more the routing network by Lyu *et al.* [31] in Figure 26 with three vehicles. By definition, all vehicles start in node 1,

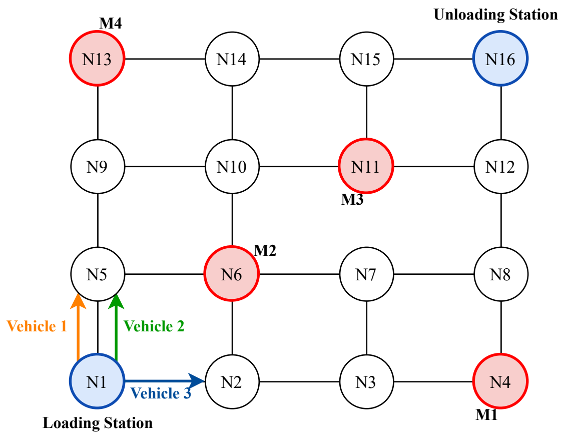


**Figure 26:** Conflict where more than three transfers leave the loading station

which marks the loading station. From that point, all vehicles can be assigned a transfer to one of the machines. All vehicles can either travel to node 2 or node 5. However, as the figure shows, if all these transfers start simultaneously, we end up in a conflicting situation. In this particular example, two vehicles are to move to node 5 at the same time; this results in a conflict in the CFR subproblem. In general, we can safely constrain the number of transfers starting at the same time from the loading station to the out-degree (the number of outgoing edges) of the loading station in the routing graph.

To capture this as a CP constraint for our master problem, we introduce $ST$ as the set of first transfers $transfer_{o_{j,0}}$ for all $j \in J$. Let $od$ be the out-degree of the starting location. We can then capture this in a constraint as follows:

$$|\{st_2 \in ST \mid \boldsymbol{start\_of}(st_1) = \boldsymbol{start\_of}(st_2), st_1 \neq st_2\}| < od \quad \text{For all } st_1 \in ST \tag{7.16}$$

# Chapter 8

# Tooling & Experiments

Using the LBBD-based CFTFJSSP implementation, it is now possible to solve CFTFJSSP instances. To conclude anything about the solution quality the LBBD-based CFTFJSSP provides, we need to perform experiments and compare results with other CFTFJSSP solutions. Unfortunately, no tools are available that allow us to reconstruct experimental results and test the benchmark instances that are available in the literature. In this chapter, we present an open-source software toolset that can be used to reconstruct experimental results obtained with LBBD-based CFTFJSSP and that can be used to introduce and test new CFTFJSSP solutions on benchmarks available in the literature.

## 8.1 | Software

To efficiently assess the performance of TJSPs, CFRPs, and CFTFJSSPs, it is crucial to evaluate the solution quality. This can be done by comparing different solving techniques or observing how different solvers or solve settings can impact the overall behavior of certain approaches. Additionally, in the study of CFTFJSSP, it is essential not only to focus on solving the TFJSSP and CFRP but also to analyze implementations and best practices learned from other job scheduling problems, such as the traditional flexible job shop or the TJSSP. Similar to what we have seen in Chapter 5, we may want to compare and test different implementations to determine which best suits the problem at hand.

Most of the mentioned optimization problems share similar structures and decision variables. Hence, a set of software tools was designed for the Python programming language to capture these common elements, enabling the quick implementation of new optimization models and the rapid testing of problem instances across different models. As we want to stimulate new competitive research regarding these kinds of optimization problems, these tools are made available to the community at large.

The software toolset is characterized by several key features, namely the creation of optimization models, problem-instance data parsing, the logic-based Benders decomposition, and benchmarking.

**Optimization Model Creation** This research exclusively applied exact-solving methods such as CP and MP. Both these methods use different IBM ILOG CPLEX solvers. As both solvers come from the same commercial toolset, both models can be programmed using the DoCPLEX Python API. While their exact modeling syntax is slightly different we can still capture a lot of common elements, allowing us to easily create new optimization models.

For both methods, an abstract class "optimization model" specifies all the methods and decision variables needed to solve one of the supported types of optimization problems. The current software toolset supports three dedicated job-scheduling problems: FJSP, TJSSP, and TFJSSP. All these problems are implemented as CP models. Furthermore, the tool contains the CFR models discussed in Chapter 5. By capturing these different types of optimization problems under a common class it becomes easier to later apply the LBBD or run benchmarking instances for each of these optimization problems.

**Model Data Parsing** Solving the model only makes sense when there is a problem instance to solve. As we do not want to constrain our models to solve only one specific problem instance, a separate parser is created that deals with extracting the problem instance information from a ".data" file. The parser supports all the standard optimization models introduced in the previous paragraph but also the optimization model for the CFTFJSSP that is decomposed in the standard optimization models. After parsing the data file, all necessary problem variables and constants are created. For example, the file for a CFTFJSSP specifies the number of jobs, the operations per job, machine options per operation, the number of vehicles, the grid size, the connections between nodes, and the special locations on the grid. The derived variables and functions as given in Definition 6.3.1 are automatically instantiated.

**Logic-Based Benders Decomposition** As we discussed in Section 3.5, the LBBD is a well-defined method and algorithm. The presented software toolset provides an LBBD class to which we can instantiate and assign master- and subproblems. The only requirement is that these master and subproblems are created using the "optimization model" class discussed earlier. This way, we can use the methods defined by the "optimization problem" class to run the LBBD successfully for both CP and MP models. In special circumstances, it is allowed to specify another LBBD class as a subproblem. This makes it possible to decompose subproblems further if the resulting subproblem after a single decomposition turns out to be still too complex. Furthermore, the LBBD can run the algorithms discussed in Section 3.5 for both optimization and feasibility subproblems. The LBBD class chooses the algorithm that fits with the assigned subproblem. What remains for the user is to extract the trial values from the master problem's solution and to specify the Benders cuts to be added to the master problem model in each iteration.

**Benchmarking** Lastly, a benchmarking tool allows the user to specify an optimization model or LBBD for benchmarking. A benchmark consists of a folder containing several problem instances (".data") files that need to be solved. Running the benchmarking tool executes all benchmark instances on the given optimization model or LBBD and stores the solver results, solutions, and figures to be used for further analysis.

Several benchmarks from the literature are provided together with the software tool. These include the benchmarks of Deroussi and Norre [10], Lyu *et al.* [31], and Liu *et al.* [29].

All software tools and benchmark instances can be found at https://github.com/TUE-EE-ES/TJSP-Toolset.

## 8.2 | Benchmarks

The benchmarks from Bilge and Ulusoy [4] (TJSSP) and Deroussi and Norre [12] (TFJSSP) form the most relevant benchmarks for the TJSSP and TFJSSP problems, respectively. There is no standardized benchmark for CFTFJSSP. The only two prior studies on CFTFJSSP variants in the literature use their own benchmarks. For the experimental work of this research, both these benchmarks published by Lyu *et al.* [31] in 2019 and Liu *et al.* [29] in 2023 are considered.

Of the two available benchmarks, Lyu *et al.* consider the most extensive set of problem instances. They provide 14 job sets and 6 differently sized routing networks (layouts). The combination of a job set, one of these layouts, and a number of vehicles forms a CFTFJSSP problem instance. Their goal is to evaluate how the makespan is influenced by the number of vehicles. Hence, every combination of a job set and layout is tested several times, each time with a different number of vehicles. As discussed in Section 6.1, the provided layouts are square grid-based graphs, some of which contain obstacles. In these layouts, only the horizontal and vertically adjacent cells are considered viable step candidates.

Liu *et al.* provide multiple small experiments to test various aspects of their CFTFJSSP solution. Several experiments evaluate how different numbers of vehicles and jobs influence the makespan and resource utilization. Other experiments are used to verify the effectiveness of the proposed algorithm. Seven of these experiments form a small benchmark. On this benchmark, Liu *et al.* tested their proposed algorithm and several other commonly seen algorithms in TJSP research to compare their effectiveness in finding the smallest makespan solutions. They consider no obstacles in the routing network and additionally allow diagonally connected cells as viable step candidates. As also discussed in Section 6.1, the provided routing networks are square grid-based graphs, not containing obstacles. In these layouts, not only the horizontal and vertically adjacent cells are considered viable step candidates, but also the diagonally adjacent cells.

## 8.3 | Goals of the Experimental Assessment

In this research, we want to evaluate the performance of the LBBD-based CFTFJSSP. While throughout TJSP research, various metaheuristic approaches were proposed for solving the considered optimization problems, we need methods to assess the solution quality of these metaheuristic approaches. This is where exact solutions come in place. By comparing the objective values of metaheuristic solutions with the ones found in an exact solution, we can assess the differences in solution quality. As the LBBD-based CFTFJSSP is such an exact solution, the focus of our experimental assessment comes down to the quality of the solution, i.e., the reported makespan. This brings us to our first goal of experimental assessment:

**EG1.** Find exact solutions for the benchmark instances. The goal is to validate that, indeed, the exact solutions outperform the metaheuristic approaches from the literature in terms of solution quality. A secondary result is that such results provide insight in the optimality gap of the heuristic approaches.

Through initial experimentation, it was determined that the time required to solve the master problem significantly affects the overall solving time of the CFTFJSSP. To limit the solving time of the master problem, we can simply introduce a timeout period to the solver. Such a timeout limits the time a solver can search for the optimal (exact) solution. If the solver cannot find an optimal solution within this time but has found at least a feasible solution, the process can continue with this feasible solution. In these cases, the optimality of the solution cannot be guaranteed, but it allows us to find decent solutions within a reasonable time. We call this procedure time-boxing.

A second goal of the assessment is to investigate the impact of time-boxing the master problem solver in the LBBD-based CFTFJSSP:

**EG2.** Evaluate the impact on the solution quality and solving time of different solver timeout periods for the master problem in the LBBD-based CFTFJSSP.

In the presented solution, additional constraints (Section 7.4.3) are specified to improve the solving time. However, the LBBD-based CFTFJSSP solution remains valid without these additional constraints. This brings up the third goal of experimental assessment:

**EG3.** Evaluate the impact of the additional constraints, as by Section 7.4.3, for the LBBD-based CFTFJSSP. A second result is that we can verify that both solutions indeed find the same makespan results, supporting the validity of the additional constraints.

Finally, similar to what is done by Lyu *et al.* [31], the LBBD-based CFTFJSSP can be used to evaluate the optimal number of vehicles for a given FMS layout and job set. This brings us our last goal of experimental assessment:

**EG4.** Evaluate the optimal number of vehicles for the benchmark of Lyu *et al.* [31].

## 8.4 | Experimental Setup

As explained earlier, both benchmarks available in literature [31, 29] are used for comparison. All tests were conducted on an Intel i7-9750H 2.60 GHz with 16 GB RAM memory. By default, the number of cores used by the solvers was set to match the number of available cores on the machine. The MP conflict refiner was given a default time limit of 180 seconds. Besides a master problem timeout, there is also a total solving time timeout of 5 hours per experiment.

We compare our results to the results seen in [31, 29]. Both the benchmarks in [31, 29] report the adjusted makespan $C^*_{max}$ as defined in Definition 6.3.5. For the remainder of this chapter, we consider the makespan to be $C^*_{max}$.

## 8.5 | Experimental Results

Let us present the experimental work in line with the defined goals of the assessment.

### 8.5.1 | Exact benchmark solutions and the impact of time-boxing the master problem.

#### Experimental results on the benchmark by Lyu *et al.* [31].

We start by comparing our LBBD-based CFTFJSSP to the benchmark solutions from Lyu *et al.* [31] . The purpose of the benchmark from Lyu *et al.* was to evaluate the exact number of vehicles needed for each of the 14 proposed job sets, where each job set was tested with one of 6 routing network layouts. This resulted in 14 sets of experiments. For each of the 14 sets of experiments, two to five problem instances were solved using a different number of vehicles. The experiment to solve each problem instance was executed from the least number to the highest number of vehicles as it is assumed that the makespan decreases when more vehicles are available. If the makespan decrease between subsequent experiments was at most 5%, the set of experiments was completed.

The purpose of the comparison is now to observe whether the LBBD-based CFTFJSSP indeed finds better makespan solutions compared to the genetic algorithm used by Lyu *et al.*. We denote each experiment as EX$ij$-$k$, which indicates the experiment of running job set $i$ with routing network layout $j$ using $k$ vehicles. To investigate the impact of time-boxing the master problem on the solution quality and sovling time, 30-second and 900-second timeout periods are tested. By using a timeout period for the master problem, we may lose the ability to find optimal solutions for all instances, but we can still find solutions that are feasible (and likely near-optimal).

There are some noteworthy differences between the experimental setups of Lyu *et al.* and this experiment. First, the reported solving times by Lyu *et al.* determine the time it takes to find the optimal number of vehicles (i.e. to find a decrease rate of at most 5%). In this research, we report the solving time for each of the individual problem instances to evaluate the performance of the LBBD-based CFTFJSSP solution. Hence, the reported solving times from Lyu *et al.* are not useful as references for this experiment. Second, as Lyu *et al.* proposed a genetic algorithm, they run each experiment 5-10 times to obtain a mean and a minimum makespan value. In our experiments, we are interested in the minimum makespan values. Hence, in all cases, when referred to the makespan of Lyu *et al.*, we consider the minimum makespan that they found.

The results and comparison of the benchmark instances can be seen in Table 13. The table shows the experiment numbers in the first column, followed by the numbers of jobs and machines for that experiment, the makespan found by Lyu *et al.*, and the solving time, solving status, and makespan found by the LBBD-based CFTFJSSP. Lastly, for each of the found solutions, the improvement percentage in makespan is given. The solutions for a 30-second and 900-second master-problem timeout period are

| Experiment | $\|J\|/\|M\|$ | $C^*_{max}$ [31] | Solving Time (s) | | Solving Status | $C^*_{max}$ | Difference (%) |
|---|---|---|---|---|---|---|---|
| | | | 30 | 900 | 30/900 | 30/900 | 30/900 |
| EX11-1 | 3/3 | 44 | 0.118 | 0.313 | Optimal | **42** | 4.55 |
| EX11-2 | | 41 | 0.227 | 0.435 | Optimal | **40** | 2.44 |
| EX11-3 | | 41 | 0.630 | 1.085 | Optimal | **40** | 2.44 |
| EX22-1 | 4/4 | 75 | 0.978 | 1.426 | Optimal | **63** | 16.00 |
| EX22-2 | | 51 | 155.415 | 117.425 | Optimal | **46** | 9.80 |
| EX22-3 | | 47 | 0.841 | 1.025 | Optimal | **46** | 2.13 |
| EX22-4 | | 47 | 25.069 | 21.793 | Optimal | **46** | 2.13 |
| EX32-1 | 5/4 | 89 | 15.054 | 23.451 | Optimal | **72** | 19.10 |
| EX32-2 | | 52 | 0.449 | 1.400 | Optimal | **44** | 15.38 |
| EX32-3 | | 47 | 1.978 | 0.868 | Optimal | **44** | 6.38 |
| EX32-4 | | 47 | 108.337 | 84.413 | Optimal | **44** | 6.38 |
| EX43-1 | 5/5 | 99 | 12.492 | 18.985 | Optimal | **81** | 18.18 |
| EX43-2 | | 61 | 18.348 | 14.896 | Optimal | **51** | 16.36 |
| EX43-3 | | 54 | 0.959 | 1.101 | Optimal | **51** | 5.56 |
| EX43-4 | | 54 | 2.735 | 2.671 | Optimal | **51** | 5.56 |
| EX53-1 | 6/5 | 119 | 30.358 | 900.410 | Feasible | 100/**98** | 15.97/17.65 |
| EX53-2 | | 68 | 12.824 | 20.264 | Optimal | **53** | 22.06 |
| EX53-3 | | 57 | 28.978 | 22.470 | Optimal | **46** | 19.30 |
| EX53-4 | | 52 | 54.961 | 47.068 | Optimal | **46** | 11.54 |
| EX53-5 | | 51 | 324.045 | 302.450 | Optimal | **46** | 9.80 |
| EX64-1 | 6/6 | 131 | 30.791 | 902.210 | Feasible | **114** | 12.98 |
| EX64-2 | | 82 | 3.062 | 1.584 | Optimal | **75** | 8.54 |
| EX64-3 | | **75** | 21.445 | 20.245 | Optimal | **75** | 0.00 |
| EX64-4 | | **75** | 5.755 | 7.046 | Optimal | **75** | 0.00 |
| EX74-1 | 7/6 | 150 | 30.978 | 901.840 | Feasible | 142/**140** | 5.33/6.67 |
| EX74-2 | | 90.5 | 8.097 | 12.272 | Optimal | **73** | 19.34 |
| EX74-3 | | 77 | 2.437 | 6.080 | Optimal | **72** | 6.49 |
| EX74-4 | | **72** | 5.536 | 8.425 | Optimal | **72** | 0.00 |
| EX74-5 | | **72** | 9.523 | 13.638 | Optimal | **72** | 0.00 |
| EX84-2 | 8/6 | 116.5 | 7.434 | 16.844 | Optimal | **93** | 20.17 |
| EX84-3 | | 100 | 5.314 | 8.225 | Optimal | **93** | 7.00 |
| EX84-4 | | 95 | 423.702 | 465.851 | Optimal | **93** | 2.11 |
| EX84-5 | | 94 | 69.965 | 101.502 | Optimal | **93** | 1.06 |
| EX95-2 | 7/7 | 107 | 89.360 | 238.150 | Optimal | **83** | 22.43 |
| EX95-3 | | 92 | ,105.446 | 149.557 | Optimal | **81** | 11.96 |
| EX95-4 | | 84 | 2219.608 | 2623.916 | Optimal | **81** | 3.57 |
| EX95-5 | | 83 | 2012.869 | 2183.748 | Optimal | **81** | 2.41 |
| EX105-3 | 8/7 | 75 | 54.119 | 55.733 | Optimal | **64** | 14.67 |
| EX105-4 | | 70 | 787.115 | 980.490 | Optimal | **64** | 8.57 |
| EX105-5 | | 67 | 48.078 | 66.159 | Optimal | **64** | 4.48 |
| EX105-6 | | 67 | 9.546 | 17.653 | Optimal | **64** | 4.48 |
| EX115-3 | 9/7 | 80 | 1085.837 | 12836.906 | Feasible | 59/**58** | 26.25/27.50 |
| EX115-4* | | 71.5 | 10499.869 | 13029.201 | Optimal | **55** | 23.07 |
| EX115-5* | | 67.5 | 223.769 | 244.333 | Optimal | **55** | 18.52 |
| EX115-6 | | 64 | 2147.101 | 2293.962 | Optimal | **55** | 14.06 |
| EX115-7 | | 63 | 15.353 | 14.962 | Optimal | **55** | 12.7 |
| EX126-2 | 8/8 | 107 | 4.376 | 10.585 | Optimal | **84** | 21.50 |
| EX126-3 | | 95 | 55.343 | 47.770 | Optimal | **84** | 11.58 |
| EX136-3 | 9/8 | 105 | 1113.198 | 967.606 | Optimal | **95** | 9.52 |
| EX136-4 | | 100.5 | 67.241 | 60.550 | Optimal | **95** | 5.47 |
| EX136-5 | | 97 | 44.471 | 38.383 | Optimal | **95** | 2.06 |
| EX136-6 | | 96 | 1374.140 | 1307.123 | Optimal | **95** | 1.04 |
| EX136-7 | | 96 | 117.110 | 123.613 | Optimal | **95** | 1.04 |
| EX146-4 | 10/8 | 118.5 | 85.655 | 79.656 | Optimal | **92** | 22.36 |
| EX146-5 | | 111.5 | 12.473 | 12.661 | Optimal | **92** | 17.49 |
| EX146-6 | | 102 | 18.337 | 19.573 | Optimal | **92** | 9.80 |
| EX146-7 | | 102 | 23.589 | 25.707 | Optimal | **92** | 9.80 |

**Table 13:** Table comparing the smallest makespan results of Lyu *et al.* [31] and the LBBD-based CFTFJSSP with a 30-second and 900-second timeout period for the master problem.

referred to as 30 and 900 respectively. If only a single result is reported this means the result is the same for both the 30-second and 900-second timeout periods. Furthermore, the best makespan values are denoted in bold.

It is important to note that the presented table is not in line with Table 6 as presented in [31], as this table is not in line with the presented benchmark instances in Appendix B of [31]. Table 13 strictly follows the experiments as presented in Appendix B of [31].

**Conclusions EG1** In all cases, the LBBD-based CFTFJSSP outperforms Lyu's genetic algorithm in terms of solution quality. We find optimal solutions in all but four problem instances (because of the time-boxing, as explained above). Lyu *et al.* found optimal solutions for only four problem instances, for two of the layout/job-set combinations. The overestimation by Lyu *et al.* of the achievable makespan is up to 37.9% (for EX115-3), which corresponds to the highest improvement percentage of 27.5 % reported in Table 13. Overall, we see that for more than half of the problem instances the percentual improvement is more than 9%, and for roughly a quarter of all instances this is even 15%.

The solving time for the LBBD-based CFTFJSSP varies from less than 1 second to several hours (for two EX115 instances; all other instances are solved within one hour). To what extent such solving times are acceptable in practical application is context-dependent. Solving times of several hours may be acceptable when schedules are generated, for instance, overnight.

**Conclusions EG2** Both using the 30-second timeout and 900-second timeout for the master problem, we solved the same problem instances to optimality. Furthermore, the solving times for these two sets of experiments are roughly similar. This indicates that the solving time for each master iteration is almost always below 30 seconds. Except for the instances in which only a feasible solution was found, further increasing the timeout period beyond 30 seconds does not give any improvement in solution quality and it does not fundamentally impact solving time.

The experiments EX53-1, EX64-1, EX74-1, and EX115-3 resulted in feasible solutions for both the 30 and 900-second timeout solutions. For these experiments, we see that there is a big impact in solving time when we increase the timeout duration. In EX115-3, this is an increase from roughly 18 minutes to 3.5 hours. In these experiments, at least one, but possibly more, master problem iterations meet the time limit. Hence, if a master problem iteration is timed out after 30 instead of 900 seconds the iteration time is reduced by 870 seconds. If this is the case for multiple iterations, a faster timeout will in these cases contribute a lot to reducing the solving time. However, in the experiments, we also see that a smaller timeout period leads to a slightly worse makespan in EX53-1, EX74-1, and EX115-3. When the solver has less time to find feasible alternatives, the found feasible solutions often have a higher makespan. Overall, we see that for problem instances that cannot be solved to optimality, depending on the context, a user can decide to trade solution quality for a reduced solving time.

**Limitations** The two * marked experiments were performed on a single core. When using the default settings of the IBM solver, these experiments timed out on the total solving time timeout of 5 hours without finding any feasible solution. During the execution of the experiments, it was discovered that changing solver settings such as the number of workers can have a big impact on the solving time of a problem instance. We hypothesize that when adjusting these solver settings we affect the way the solver traverses the solution space. Hence, by changing the number of cores (workers) we were able to find results for the * marked experiments.

While the solver was not able to find exact solutions for experiments EX53-1, EX64-1, EX74-1, and EX115-3 within the given timeout period for the master problem, it does obtain lower bounds for each of them. Although it is not clear how tight the lower bounds are, these provide insight into how far a solution may deviate from a hypothetical optimum. The percentage it deviates from a hypothetical optimal solution is called the optimality gap. Table 14 shows the obtained lower bounds and optimality gaps for these instances.

| Experiment | Best Lower Bound | $C^*_{max}$ 30/900 | Optimality Gap 30/900 |
|---|---|---|---|
| EX53-1 | 70 | 100/98 | 30.00/28.57 |
| EX64-1 | 85 | 114 | 25.44 |
| EX74-1 | 79 | 142/140 | 44.37/43.57 |
| EX115-3 | 55 | 59/58 | 6.78/5.17 |

**Table 14:** Comparing the feasible solutions under different solver timeout settings.

If an optimal solution is found, the makepan and the lower bound are the same. During the solver process, as time elapses, the provided lower bound increases until the found makespan and lower bound are the same. The optimality gap is quite large for experiments EX53-1, EX64-1, and EX74-1, which indicates that the solver was not close to finding the optimal solution soon. Note that this can still mean that the found makespan is close to the optimal makespan. For EX115-3 the bound is relatively small indicating that the solver was close to finding the optimal solution but was not able to find it within the time limits.

## Experimental results on the benchmark by Liu *et al.* [29]

Besides the benchmark from Lyu *et al.*, the LBBD-based CFTFJSSSP was also tested on the benchmark of Liu *et al.* [29]. The results can be found in Table 15. The table shows the experiment (exp.) numbers in the first column, followed by the numbers of jobs, machines and vehicles for that experiment. The third up to the sixth columns show the different solution methods Liu *et al.* tested. The GA-D method refers to a conventional genetic algorithm integrated with Dijkstra's algorithm with time windows. The AGA-D method refers to an adaptive genetic algorithm integrated with Dijkstra's algorithm with time windows. The SLGA-D method refers to the method combining a self-learning genetic algorithm with Dijkstra's algorithm with time windows. This is the main algorithm Liu *et al.* present in [29]. The HPSO-D method refers to a hybrid particle swarm optimization integrated with Dijkstra's algorithm with time windows. The remaining columns present the solution of the LBBD-based CFTFJSSP and the relative improvement percentage (diff.) in makespan compared to the best-found makespan by one of the algorithms reported by Liu *et al.*. Liu *et al.* only report the iteration time of their algorithm; they do not report overall solving times that can be used for comparison.

| Exp. | $\lvert J\rvert/\lvert M\rvert/\lvert V\rvert$ | GA-D $C^*_{max}$ [29] | AGA-D $C^*_{max}$[29] | SLGA-D $C^*_{max}$[29] | HPSO-D $C^*_{max}$ [29] | Solving Time (s) | | Solving Status | $C^*_{max}$ | Diff. (%) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 30 | 900 | 30/900 | 30/900 | 30/900 |
| EX1 | 2/3/2 | 17.20 | 17.20 | 17.20 | 17.20 | 1.075 | 1.221 | Optimal | **13** | 24.42 |
| EX2 | 3/3/2 | 18.00 | 18.00 | 18.00 | 18.00 | 0.197 | 0.234 | Optimal | **15**** | 16.67 |
| EX3 | 4/4/3 | 37.20 | 37.20 | 37.20 | 37.20 | 3.663 | 3.052 | Optimal | **33** | 11.29 |
| EX4 | 5/5/4 | 46.60 | 41.00 | 39.60 | 45.40 | 7.509 | 7.174 | Optimal | **31** | 21.72 |
| EX5 | 6/4/4 | 56.00 | 50.60 | 49.00 | 53.40 | 16.903 | 10.159 | Optimal | **34** | 30.61 |
| EX6 | 7/5/4 | 86.40 | 74.60 | 74.80 | 84.80 | 53.369 | 834.931 | Feasible/ Optimal | 70/**45** | 6.17/39.68 |
| EX7 | 8/3/3 | 92.80 | 90.20 | 87.40 | 87.60 | 33.001 | 582.954 | Feasible/ Optimal | 80/**78** | 8.47/10.76 |

**Table 15:** Table comparing the smallest makespan results of Liu *et al.* [29] and the LBBD-based CFTFJSSP.

The ** marked experiment 2 is not in line with experiment 2 as presented in Table 6 of [29]. We believe that, due to a spacing error, the operation processing times in the experimental description in [29] were shifted, leading to an inconsistent CFTFJSSP instance. Experiment 2 shown in Table 15 was run on a version that was corrected to the best of our understanding. Our modified version can be found in the open-source software toolset.

**Conclusions EG1** In all cases, the LBBD-based CFTFJSSP outperforms Liu's self-learning genetic algorithm in terms of solution quality. Furthermore, for all the benchmark instances, we were able to find optimal solutions when using a timeout of 900 seconds. The SLGA-D algorithm by Liu *et al.* was not able to find any optimal solution. Furthermore, the overestimation of their algorithms is up to 66.22% (in EX6). For the 900-second timeout results, the percentual improvement is more than 10%. The solving times range from less than 1 second to about 14 minutes.

**Conclusions EG2** Similar to the results seen for the benchmark by Lyu *et al.* we see that for EX1 up to EX5 we find optimal solutions with similar solving time when applying both timeout periods for the master problem. However, in EX6 and EX7, we see that by using a timeout period of 30 seconds, we are only able to find feasible solutions, while a 900-seconds time budget leads to optimal solutions. For the problem instances of these experiments, we see that the optimal master problem computation time is apparently between 30 and 900 seconds. From this, we conclude that for some of the feasible problem instances, increasing timeout times can in fact lead to finding optimal solutions.

#### Overall Conclusions EG1 and EG2

**Conclusions EG1** The LBBD-based CFTFJSSP finds exact solutions for all but four benchmark instances available in the literature. It outperforms all existing benchmark experiments in terms of solution quality. Overall, the improvement in solution quality is more than 9% on all benchmark instances.

**Conclusions EG2** The tested timeout settings for the master problem do not affect the solution for most of the presented benchmark instances. In case no optimal solution can be found for a master problem instance within the given time budget, shorter timeout periods can lead to a reduced solution quality. The best timeout value in a practical setting may depend on the context and application.

### 8.5.2 | Impact of additional master problem constraints.

For the second goal of experimental assessment, we evaluate the performance difference with and without using the additional master problem constraints specified in Section 7.4.3. The additional constraints make sure that some known sets of conflicting transfers are avoided to begin with. By removing the additional constraints, the master problem reduces in size and possibly becomes faster to solve. However, the master problem allows these known sets of conflicting transfers to occur, leaving the work for the conflict-driven cuts to avoid them in future iterations. Hence, the resulting decomposition most likely has a shorter iteration time but needs more iterations to converge towards an optimum.

For this experiment, a selected subset of benchmark instances was run without the additional constraints and we applied a master problem timeout period of 900 seconds. We can then compare the newly found solving time to those found in the experiments seen in Table 13 and Table 15. As we expect this modification to the LBBD-based CFTFJSSP to lead to more iterations, we limit the number of iterations to 100 to avoid very long search times. Simultaneously, the total solving time timeout period of 5 hours is removed as we now bound the solve process by limiting the iterations. While this may limit the number of solved problem instances, it gives enough reference material to draw conclusions within a reasonable time.

#### Experimental results on the benchmark by Lyu *et al.* [31]

The results for the benchmark instances of Lyu *et al.* can be seen in Table 16. The table shows the experiment numbers in the first column followed by the solving time, solving status, and makespan found by the LBBD-based CFTFJSSP without the additional constraints from Section 7.4.3. To limit the duration of the complete experimental assessment, EX136 and EX146 were omitted in this setup.

Several entries have an ‘unknown’ solving status in the table. This indicates that the maximum number of iterations was reached or the solver could not find a feasible solution within the 900-second time-out limit.

For those experiments in which a solution was found, while there was no restriction on the number of iterations for the experiments in Table 13, these experiments still outperform those seen in Table 16 in terms of solving time. For a fragment of the experiments, the solving times are similar to those seen in Table 13. However, there are some noteworthy differences. EX11-3, EX32-3, EX32-3, EX43-3, and EX43-4 are examples where we see that while the solution is still found very quickly it takes slightly more

| Experiment | Solving Time (s) | Solving Status | $C^*_{max}$ |
|---|---|---|---|
| EX11-1 | 0.928 | Optimal | 42 |
| EX11-2 | 0.249 | Optimal | 40 |
| EX11-3 | 37.794 | Optimal | 40 |
| EX22-1 | 1.011 | Optimal | 63 |
| EX22-2 | 26.212 | Optimal | 46 |
| EX22-3 | 7.562 | Optimal | 46 |
| EX22-4 | 5.125 | Optimal | 46 |
| EX32-1 | 21.760 | Optimal | 72 |
| EX32-2 | 15.760 | Optimal | 44 |
| EX32-3 | 20.530 | Optimal | 44 |
| EX32-4 | 41.357 | Optimal | 44 |
| EX43-1 | 15.266 | Optimal | 81 |
| EX43-2 | 19.569 | Optimal | 51 |
| EX43-3 | 11.766 | Optimal | 51 |
| EX43-4 | 23.308 | Optimal | 51 |
| EX53-1 | 900.397 | Feasible | 98 |
| EX53-2 | 19.438 | Optimal | 53 |
| EX53-3 | 53.017 | Optimal | 46 |
| EX53-4 | 519.340 | Optimal | 46 |
| EX53-5 | 490.353 | Optimal | 46 |
| EX64-1 | 900.971 | Feasible | 114 |
| EX64-2 | 1.551 | Optimal | 75 |
| EX64-3 | 10.596 | Optimal | 75 |
| EX64-4 | 308.253 | Optimal | 75 |
| EX74-1 | 901.870 | Feasible | 140 |
| EX74-2 | 11.593 | Optimal | 73 |
| EX74-3 | 1115.783 | Optimal | 72 |
| EX74-4 | 8879.627 | Optimal | 72 |
| EX74-5 | 12341.487 | Optimal | 72 |
| EX84-2 | 50.941 | Optimal | 93 |
| EX84-3 | 426.489 | Optimal | 93 |
| EX84-4 | 7110.728 | Optimal | 93 |
| EX84-5 | 1192.613 | Optimal | 93 |
| EX95-2 | 134.097 | Optimal | 83 |
| EX95-3 | 166.143 | Optimal | 81 |
| EX95-4 | 1035.719 | Optimal | 81 |
| EX95-5 | 748.990 | Optimal | 81 |
| EX105-3 | 750.924 | Optimal | 64 |
| EX105-4 | 467.254 | Optimal | 64 |
| EX105-5 | 3484.963 | Optimal | 64 |
| EX105-6 | 1968.337 | Unknown | 64 |
| EX115-3 | 28524.010 | Unknown | |
| EX115-4 | 2892.631 | Unknown | |
| EX115-5 | 23652.894 | Optimal | 55 |
| EX115-6 | 2815.540 | Unknown | |
| EX115-7 | 2924.164 | Unknown | |
| EX126-2 | 3.535 | Optimal | 84 |
| EX126-3 | 1966.226 | Optimal | 84 |

**Table 16:** Results for the LBBD-based CFTFJSSP without additional master problem constraints on the benchmark of of Lyu *et al.* [31].

iterations and solving time to find solutions without using the additional master problem constraints. In other experiments, such as EX53-4, EX53-5, EX64-4, EX74-3, EX74-4, EX74-5, EX84-3, EX84-4, EX84-5, EX105-3, EX105-5, EX115-5, and EX126-3, we see that the solving times from Table 16 are very high in comparison to those in Table 13. Especially for the experiments with a high job and machine count, the additional constraints seem to have the most impact. This indicates that the use of the additional constraints has a high impact in reducing solving time for these large problem instances and is therefore a valuable addition to our LBBD-based CFTFJSSP solution.

In some experiments, such as EX22-2, EX95-2, EX95-4, EX95-5, EX105-4, and EX136-3, the solving time reported in Table 16 outperforms those seen in Table 13. At each iteration of the LBBD-based CFTFJSSP, there is a big possible solution space. When we change the master problem, for example by removing the additional constraints, the way the solver traverses this space is changed. In some cases, this may lead to finding optimal solutions faster.

### Experimental results on the benchmark by Liu *et al.* [31]

The same experiments can be performed considering the benchmark instances by Liu *et al.* [29]. The results can be seen in Table 17. The table shows the experiment numbers in the first column followed by the solving time, solving status, and makespan found by the LBBD-based CFTFJSSP without the additional constraints from Section 7.4.3.

| Experiment | Solving Time (s) | Solving Status | $C^*_{max}$ |
|---|---|---|---|
| EX1 | 1.358 | Optimal | 13 |
| EX2** | 1.883 | Optimal | 15 |
| EX3 | 4.860 | Optimal | 33 |
| EX4 | 16.305 | Optimal | 31 |
| EX5 | 3.703 | Optimal | 34 |
| EX6 | 280.668 | Optimal | 45 |
| EX7* | 18.979 | Optimal | 78 |

**Table 17:** Results for the LBBD-based CFTFJSSP without additional master problem constraints on the benchmark of Liu *et al.* [29].

Observe that optimal solutions are found for all instances. Except for EX6 and EX7, there is a slightly increased computation time compared to the results from Table 15 for each of the experiments. For experiments EX6 and EX7, it was found that the additional constraints lead to long master problem solving times. The experiments without the extra constraints find optimal solutions relatively fast.

It is important to note that the * marked EX7 experiment was performed on a single core. When using the default settings of the IBM solver, this experiment resulted in an out-of-memory error. As discussed in Section 7.3.2, when a conflict occurs, many sets of similar conflicting transfers can be found. This may result in a considerable number of cuts being added to the master problem in each iteration. EX7 is considered a large problem instance—i.e., many jobs, operations, and machines—in the benchmark of Liu *et al.*. These naturally produce an increased number of conflicting transfer sets compared to smaller problem instances. The result is a lot of additional memory usage to store all these sets of transfers and constraints that follow from them. We hypothesize that the cuts that followed from a specific set of transfers made the solver run out of memory when the solver is running with multiple worker processes simultaneously on multiple cores. By once more limiting the solver to a single core, all memory is available to a single worker, avoiding the memory bottleneck.

### Conclusions EG3

Overall, we may conclude from the experiments discussed in this subsection that the addition of the additional master problem constraints in the LBBD-based CFTFJSSP has a positive impact on the performance of the LBBD-based CFTFJSSP. Due to the added constraints, in most cases, we reduce solving time. More importantly, the added constraint makes sure we can find solutions to all benchmark instances within reasonable times.

To better understand what aspects impact the effectiveness of the LBBD approach, more detailed analysis is required. Hence, to find the best combination of additional constraints, Bender's cuts, and solver settings such as the number of workers, time bounds, and iteration limits, we recommend for future work to perform additional experiments analyzing the number of iterations and iteration time as well.

### 8.5.3 | Evaluating the optimal number of vehicles.

The original experimental setup by Lyu *et al.* was to decide upon the optimal number of vehicles for each of the 14 sets of experiments. Lyu *et al.* determined that an optimal number of vehicles for their genetic algorithm was found when the decrease in makespan with one additional vehicle is at most 5%. As the LBBD-based CFTFJSSP provides optimal solutions, we can conclude the optimal number of vehicles when the same makespan is found with an additional vehicle.

Table 18 summarizes results from the earlier experiments and [31]. The first column shows the set of experiments we are considering, the second column specifies the optimal number of vehicles as reported by Lyu *et al.* [31], and the last column shows the optimal number of vehicles determined by the LBBD-based CFTFJSSP using the results from Table 13. All found results, except possibly the result for EX115, are optimal. Experiment EX115-3 was not solved to optimality. Moreover, the reported lower bound in Table 14 for EX115-3 is 55, which matches the optimal makespan for EX115-4. If the optimal solution to EX115-3 is in fact 55, the optimal number of vehicles for EX115 would be 3.

| Experiment | Lyu *et al.* | LBBD-based CFTFJSSP |
|---|---|---|
| EX11 | 2 | 2 |
| EX22 | 3 | 2 |
| EX32 | 3 | 2 |
| EX43 | 3 | 2 |
| EX53 | 4 | 3 |
| EX64 | 3 | 2 |
| EX74 | 4 | 3 |
| EX84 | 4 | 2 |
| EX95 | 4 | 3 |
| EX105 | 5 | 3 |
| EX115 | 6 | 4 (possibly 3) |
| EX126 | 4 | 2 |
| EX136 | 5 | 3 |
| EX146 | 6 | 4 |

**Table 18:** The optimal number of vehicles for each set of experiments.

#### Conclusions EG4

It can be observed that the LBBD-based CFTFJSSP outperforms Lyu *et al.* in finding the optimal number of vehicles. The approach of Lyu *et al.* [31] overestimates the number of vehicles needed for all layout/job-set combinations, except for experiment set EX11.

# Chapter 9

# Conclusions and Future Work

This thesis presented various contributions to the solution and optimization of the CFTFJSSP. In this last chapter, we discuss which conclusions can be drawn from the presented work and present several recommendations for future improvement and research.

## 9.1 | Conclusions

The main objective was to find the exact shortest makespan solutions for the CFTFJSSP using an iterative decomposition-based approach. This was achieved by:

1. Researching constraint programming solutions for the TFJSSP and developing a constraint programming model for solving the TFJSSP according to the specification by Ham [16].
2. Developing a CP and two MP models, both ILPs, for solving the CFRP.
    - It was not possible to create a CP model that utilizes the temporal functions that CP offers without simplifying the problem statement of the CFRP. Experimental evaluation showed that both ILP solutions outperform the CP solution in terms of solving time.
    - The first ILP considers a binary variable for a vehicle traversing from one node to another at a certain time. The second ILP considers a binary variable indicating that a vehicle is present in a node at a certain time. Experimental evaluation showed that the second ILP was superior in terms of solving time, especially for larger problem instances.
3. Developing an LBBD-based CFTFJSSP solution, using the CP model for the TFJSSP as a master problem and the second ILP model for the CFRP as subproblem.
    - The LBBD-based CFTFJSSP is the first exact approach for solving the CTFJSSP.
    - The procedure applies an innovative conflict-driven Benders cut based on the conflict-refiner capability of the IBM CPLEX solver. The cut analyses the conflict structure and excludes the occurrence of many similar solutions to the master problem instance that cause similar conflicts in the subproblem instance.
4. Experimental work evaluating the performance of the LBBD-based CFTFJSSP.
    - The LBBD-based CFTFJSSP was able to find better makespan solutions for all benchmark instances by Lyu *et al.* [31] and Liu *et al.* [29], therefore, outperforming the algorithms by Lyu *et al.* and Liu *et al.* in terms of solution quality.

- Adding additional master problem constraints to exclude known conflict situations makes sure the LBBD-based CTFJSSP has an improved solving time compared to using a master problem formulation without these constraints.
- For finding the optimal number of vehicles, the LBBD-based CFTFJSSP has been shown to have a faster convergence rate than the algorithm used by Lyu *et al.* .

Furthermore, a second objective was to provide clear, uniform problem definitions, model implementations, and software for future work to expand on the foundations of this research. This was achieved by:

1. Defining general-purpose mathematical definitions for the CFTFJSSP and for the TFJSSP and CFRP used in its decomposition.
   - The mathematical definitions provide a general problem description for the various optimization problems regarding the CFTFJSSP. The formalizations can be applied as a basis for any solution method that might be considered as future work.
2. Development of an open-source Python software toolset for further exploration and development of TJSPs.
   - The tooling contains a set of applications that can be used for creating optimization problems and LBBD-based optimization problem solutions or to improve the currently presented LBBD-based CFTFJSSP solution. Additionally, a simple parser application is provided that allows experimental evaluation on many different problem instances. Lastly, a benchmarking tool is included that allows to execute benchmarks on LBBD-based optimization models.
   - The toolset is made publicly available for future researchers to use for solution comparison and reference material.

## 9.2 | Future work

There are a lot of directions for further research when it comes to optimizing the CFTJSP and related optimization problems.

**Solving Techniques:** In this research, both the CP and MP solving techniques were applied. The exploration of other solving techniques such as SMT solving—as used by Riazi *et al.* [43, 42]—is still scarcely studied in relation to the CFTJSP and is an interesting direction for future work. Additionally, the currently presented models can still be improved or altered leading to possibly better performing solutions for the TFJSSP, CFRP and CFTFJSSP.

**Benders Cuts:** The performance of an LBBD-based solution mostly comes down to designing strong Benders cuts. Possibly, there exist other stronger cuts than the conflict-driven cuts presented in this thesis leading to even faster algorithm convergence and shorter solving times. One option for improving the current conflict-driven cut used in the LBBD-based CFTFJSSP is to further strengthen the cuts in order to exclude solutions that provide similar sets of conflicting transfers that are shifted in time, as discussed in Section 7.3.2.

**LBBD:** There are a lot of future research directions imaginable related to using the LBBD. First of all, one may research how the LBBD-based CFTFJSSP performs when we further decompose the problem. The CFTFJSSP has seven decision variables. Currently, these are split into a set of master problem decision variables and subproblem decision variables. However, one may consider the master or subproblem to take form as a separate LBBD and further decompose the set of master or subproblem decision variables into new sets. For instance, the allocation and scheduling in the master problem may be further decomposed into separate allocation and scheduling problems.

Another interesting take is to investigate whether one can take advantage of subproblems that *decouple* into smaller component problems. The example illustrated in Example 3.5.1 considers an allocation master problem and a scheduling subproblem. Note that for this example the scheduling for each machine is independent of the scheduling of another machine. Hence, one may consider decoupling the subproblem in such a way that we can solve the scheduling per individual machine. The resulting optimization problem is even simpler and the various decoupled components can be possibly solved in a multi-threaded fashion.

**Experimental Work, Hyperparameter Tuning:** During experimental evaluation, it became clear that the current LBBD-based CFTFJSSP performance in solving time is sensitive to small adjustments in the experimental setup. Adding/removing additional constraints or changing solver settings may lead to a different exploration of the solution space, sometimes resulting in drastically better or worse solving times. The impact of these changes is something that should be considered in future work as it can lead to inconsistent solution performance across problem instances in terms of solving time.

A first recommendation would be to further investigate the impact of the additional master problem constraints specified in Section 7.4.3. This can be achieved by comparing the number of iterations the procedure takes and the master problem solving time per iteration for each benchmark instance with and without applying the additional master problem constraints. This gives more insight in what aspects form the largest bottlenecks in the total solving time.

Second, as it was noticed that the master problem's solver settings—especially the number of workers and the time budget—impact the solving time, it is interesting to explore which solver settings optimally benefit the LBBD-based CFTFJSSP.

**Heuristics:** The decomposed nature of the LBBD allows us to easily replace a solution technique of the master and subproblem with another solution technique. This is particularly interesting if one desires to speed up the LBBD procedure at the price of solution quality. Both master and/or subproblem solutions techniques can be replaced by a heuristic solution approach such as a genetic algorithm for the TFJSSP or Dijkstra's algorithm with time-windows for the CFRP to further enhance performance in terms of solving time.

**Optimization Objective(s):** The current LBBD-based CFTFJSSP optimizes merely for the single objective of minimum makespan. While this is very common in the research landscape, it can be interesting to research other optimization objectives such as vehicle utilization, minimum tardiness, and energy consumption. A particularly interesting non-explored research area is to find multi-objective solutions to the CFTFJSSP.

# Bibliography

[1] Abdelmaguid, Tamer F. et al. "A hybrid GA/heuristic approach to the simultaneous scheduling of machines and automated guided vehicles". In: *International Journal of Production Research* 42.2 (Jan. 15, 2004), pp. 267–281. DOI: 10.1080/0020754032000123579.

[2] Baruwa, Olatunde T. and Piera, Miquel A. "A coloured Petri net-based hybrid heuristic search approach to simultaneous scheduling of machines and automated guided vehicles". In: *International Journal of Production Research* 54.16 (Aug. 17, 2016), pp. 4773–4792. DOI: 10.1080/00207543.2015.1087656.

[3] Benders, J. F. "Partitioning procedures for solving mixed-variables programming problems". In: *Numerische Mathematik* 4.1 (Dec. 1, 1962), pp. 238–252. DOI: 10.1007/BF01386316.

[4] Bilge, Ümit and Ulusoy, Gündüz. "A Time Window Approach to Simultaneous Scheduling of Machines and Material Handling System in an FMS". In: *Operations Research* 43.6 (Dec. 1995), pp. 1058–1070. DOI: 10.1287/opre.43.6.1058.

[5] Bodin, Lawrence and Golden, Bruce. "Classification in vehicle routing and scheduling". In: *Networks* 11.2 (June 1981), pp. 97–108. DOI: 10.1002/net.3230110204.

[6] Booth, Kyle E. C. and Beck, J. Christopher. "A Constraint Programming Approach to Electric Vehicle Routing with Time Windows". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by Louis-Martin Rousseau and Kostas Stergiou. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 129–145. DOI: 10.1007/978-3-030-19212-9_9.

[7] Broadbent, A.J. et al. "Free ranging AGV systems : promises, problems and pathways". In: *Proceedings of the 2nd International Conference on Automated Materials Handling, 1985* (1985).

[8] Chen, Zhi-Long. "Integrated Production and Outbound Distribution Scheduling: Review and Extensions". In: *Operations Research* 58.1 (Feb. 2010), pp. 130–148. DOI: 10.1287/opre.1080.0688.

[9] Corréa, Ayoub Insa, Langevin, André, and Rousseau, Louis-Martin. "Scheduling and routing of automated guided vehicles: A hybrid approach". In: *Computers & Operations Research*. Part Special Issue: Odysseus 2003 Second International Workshop on Freight Transportation Logistics 34.6 (June 1, 2007), pp. 1688–1707. DOI: 10.1016/j.cor.2005.07.004.

[10] Deroussi, L., Gourgand, M., and Tchernev, N. "A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles". In: *International Journal of Production Research* 46.8 (Apr. 15, 2008), pp. 2143–2164. DOI: 10.1080/00207540600818286.

[11] Deroussi, Laurent. "A Hybrid PSO Applied to the Flexible Job Shop with Transport". In: *Swarm Intelligence Based Optimization*. Ed. by Patrick Siarry, Lhassane Idoumghar, and Julien Lepagnot. Cham: Springer International Publishing, 2014, pp. 115–122. DOI: 10.1007/978-3-319-12970-9_13.

[12] Deroussi, Laurent and Norre, S. "Simultaneous scheduling of machines and vehicles for the flexible job shop problem". In: International conference on metaheuristics and nature inspired computing. Tunisia: Djerba Island, 2010, pp. 1–2.

[13] Eigbe, Eghonghon-Aye et al. "Sequence- and Time-Dependent Maintenance Scheduling in Twice Re-Entrant Flow Shops". In: *IEEE Access* 11 (2023), pp. 103461–103475. DOI: 10.1109/ACCESS.2023.3317533.

[14] El Khayat, Ghada, Langevin, André, and Riopel, Diane. "Integrated production and material handling scheduling using mathematical programming and constraint programming". In: *European Journal of Operational Research* 175.3 (Dec. 16, 2006), pp. 1818–1832. DOI: 10.1016/j.ejor.2005.02.077.

[15] Hà, Minh Hoàng et al. *A new constraint programming model and a linear programming-based adaptive large neighborhood search for the vehicle routing problem with synchronization constraints.* Oct. 17, 2019.

[16] Ham, Andy. "Transfer-robot task scheduling in flexible job shop". In: *Journal of Intelligent Manufacturing* 31.7 (Oct. 1, 2020), pp. 1783–1793. DOI: 10.1007/s10845-020-01537-6.

[17] Ham, Andy. "Transfer-robot task scheduling in job shop". In: *International Journal of Production Research* 59.3 (Feb. 1, 2021), pp. 813–823. DOI: 10.1080/00207543.2019.1709671.

[18] Ham, Andy M. and Cakici, Eray. "Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches". In: *Computers & Industrial Engineering* 102 (Dec. 1, 2016), pp. 160–165. DOI: 10.1016/j.cie.2016.11.001.

[19] Homayouni, Seyed Mahdi and Fontes, Dalila B. M. M. "Joint scheduling of production and transport with alternative job routing in flexible manufacturing systems". In: International Conference on "Multidimensional Role of Basic Science in Advanced Technology" ICMBAT 2018. Nagpur, India, 2019, p. 020045. DOI: 10.1063/1.5090012.

[20] Hooker, J.N. and Ottosson, G. "Logic-based Benders decomposition". In: *Mathematical Programming* 96.1 (Apr. 1, 2003), pp. 33–60. DOI: 10.1007/s10107-003-0375-9.

[21] Hooker, John. *Logic-Based Benders Decomposition: Theory and Applications.* Synthesis Lectures on Operations Research and Applications. Cham: Springer International Publishing, 2024. DOI: 10.1007/978-3-031-45039-6.

[22] Hosseini, Amir, Otto, Alena, and Pesch, Erwin. "Scheduling in manufacturing with transportation: Classification and solution techniques". In: *European Journal of Operational Research* (Oct. 2023), S0377221723007701. DOI: 10.1016/j.ejor.2023.10.013.

[23] Johnson, S. M. "Optimal two- and three-stage production schedules with setup times included". In: *Naval Research Logistics Quarterly* 1.1 (Mar. 1954), pp. 61–68. DOI: 10.1002/nav.3800010110.

[24] Kim, Chang W. and Tanchoco, J. M. A. "Conflict-free shortest-time bidirectional AGV routeing". In: *International Journal of Production Research* 29.12 (Dec. 1991), pp. 2377–2391. DOI: 10.1080/00207549108948090.

[25] Krishnamurthy, Nirup N., Batta, Rajan, and Karwan, Mark H. "Developing Conflict-Free Routes for Automated Guided Vehicles". In: *Operations Research* 41.6 (Dec. 1993), pp. 1077–1090. DOI: 10.1287/opre.41.6.1077.

[26] Kusiak, Andrew. "Smart manufacturing". In: *International Journal of Production Research* 56.1 (Jan. 17, 2018), pp. 508–517. DOI: 10.1080/00207543.2017.1351644.

[27] Laborie, Philippe et al. "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG". In: *Constraints* 23.2 (Apr. 2018), pp. 210–250. DOI: 10.1007/s10601-018-9281-x.

[28] Lee, Chung-Yee and Chen, Zhi-Long. "Machine scheduling with transportation considerations". In: *Journal of Scheduling* 4.1 (2001), pp. 3–24. DOI: 10.1002/1099-1425(200101/02)4:1<3::AID-JOS57>3.0.CO;2-D.

[29] Liu, Jiaojiao et al. "An integrated scheduling approach considering dispatching strategy and conflict-free route of AMRs in flexible job shop". In: *The International Journal of Advanced Manufacturing Technology* 127.3 (July 1, 2023), pp. 1979–2002. DOI: 10.1007/s00170-022-10619-z.

[30] Lunardi, Willian T. et al. "Mixed Integer linear programming and constraint programming models for the online printing shop scheduling problem". In: *Computers & Operations Research* 123 (Nov. 2020), p. 105020. DOI: 10.1016/j.cor.2020.105020.

[31] Lyu, Xiangfei et al. "Approach to Integrated Scheduling Problems Considering Optimal Number of Automated Guided Vehicles and Conflict-Free Routing in Flexible Manufacturing Systems". In: *IEEE Access* 7 (2019), pp. 74909–74924. DOI: 10.1109/ACCESS.2019.2919109.

[32] Maggu, Parkash Lal and Das, Ghanshiam. "On 2× n sequencing problem with transportation times of jobs". In: *Pure and Applied Mathematika Sciences* 12.1 (1980), p. 6.

[33] Nishi, T., Ando, M., and Konishi, M. “Distributed route planning for multiple mobile robots using an augmented Lagrangian decomposition and coordination technique”. In: *IEEE Transactions on Robotics* 21.6 (Dec. 2005), pp. 1191–1200. DOI: 10.1109/TRO.2005.853489.

[34] Nishi, Tatsushi, Hiranaka, Yuichiro, and Grossmann, Ignacio E. “A bilevel decomposition algorithm for simultaneous production scheduling and conflict-free routing for automated guided vehicles”. In: *Computers & Operations Research* 38.5 (May 1, 2011), pp. 876–888. DOI: 10.1016/j.cor.2010.08.012.

[35] Nouri, Houssem Eddine, Belkahla Driss, Olfa, and Ghédira, Khaled. “Simultaneous scheduling of machines and transport robots in flexible job shop environment using hybrid metaheuristics based on clustered holonic multiagent model”. In: *Computers & Industrial Engineering* 102 (Dec. 1, 2016), pp. 488–501. DOI: 10.1016/j.cie.2016.02.024.

[36] Nouri, Houssem Eddine, Driss, Olfa Belkahla, and Ghédira, Khaled. “A Classification Schema for the Job Shop Scheduling Problem with Transportation Resources: State-of-the-Art Review”. In: *Artificial Intelligence Perspectives in Intelligent Systems*. Ed. by Radek Silhavy et al. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2016, pp. 1–11. DOI: 10.1007/978-3-319-33625-1_1.

[37] Pinxten, Joost van. “Optimization of Product Flows in Flexible Manufacturing Systems”. PhD thesis. Eindhoven University Of Technology, 2018.

[38] Potts, C N and Strusevich, V A. “Fifty years of scheduling: a survey of milestones”. In: *Journal of the Operational Research Society* 60 (sup1 May 2009), S41–S68. DOI: 10.1057/jors.2009.2.

[39] Potts, C. N. “Technical Note—Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Times”. In: *Operations Research* 28.6 (Dec. 1980), pp. 1436–1441. DOI: 10.1287/opre.28.6.1436.

[40] Qiu, Ling et al. “Scheduling and routing algorithms for AGVs: A survey”. In: *International Journal of Production Research* 40.3 (Jan. 1, 2002), pp. 745–760. DOI: 10.1080/00207540110091712.

[41] Reddy, B. S. P. and Rao, C. S. P. “A hybrid multi-objective GA for simultaneous scheduling of machines and AGVs in FMS”. In: *The International Journal of Advanced Manufacturing Technology* 31.5 (Dec. 1, 2006), pp. 602–613. DOI: 10.1007/s00170-005-0223-6.

[42] Riazi, Sarmad and Lennartson, Bengt. “Using CP/SMT Solvers for Scheduling and Routing of AGVs”. In: *IEEE Transactions on Automation Science and Engineering* 18.1 (Jan. 2021), pp. 218–229. DOI: 10.1109/TASE.2020.3012879.

[43] Riazi, Sarmad et al. “Scheduling and Routing of AGVs for Large-scale Flexible Manufacturing Systems”. In: *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). Vancouver, BC, Canada: IEEE, Aug. 2019, pp. 891–896. DOI: 10.1109/COASE.2019.8842849.

[44] Saidi-Mehrabad, Mohammad et al. “An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs”. In: *Computers & Industrial Engineering*. Applications of Computational Intelligence and Fuzzy Logic to Manufacturing and Service Systems 86 (Aug. 1, 2015), pp. 2–13. DOI: 10.1016/j.cie.2015.01.003.

[45] Sun, Jiachen et al. “An Approach to Integrated Scheduling of Flexible Job-Shop Considering Conflict-Free Routing Problems”. In: *Sensors* 23.9 (May 6, 2023), p. 4526. DOI: 10.3390/s23094526.

[46] Thieme, Alex and Basten, T. “Minesweeper is Difficult Indeed!” In: 2022.

[47] Ulusoy, Gündüz, Sivrikaya-Şerifoğlu, Funda, and Bilge, Ümit. “A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles”. In: *Computers & Operations Research* 24.4 (Apr. 1, 1997), pp. 335–351. DOI: 10.1016/S0305-0548(96)00061-5.

[48] Umar, Umar Ali et al. “Hybrid multiobjective genetic algorithms for integrated dynamic scheduling and routing of jobs and automated-guided vehicle (AGV) in flexible manufacturing systems (FMS) environment”. In: *The International Journal of Advanced Manufacturing Technology* 81.9 (Dec. 1, 2015), pp. 2123–2141. DOI: 10.1007/s00170-015-7329-2.

[49] Xie, Chen and Allen, Theodore T. “Simulation and experimental design methods for job shop scheduling with material handling: a survey”. In: *The International Journal of Advanced Manufacturing Technology* 80.1 (Sept. 1, 2015), pp. 233–243. DOI: 10.1007/s00170-015-6981-x.

[50] Zhang, Q., Manier, H., and Manier, M.-A. “A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times”. In: *Computers & Operations Research* 39.7 (July 2012), pp. 1713–1723. DOI: 10.1016/j.cor.2011.10.007.

[51] Zheng, Yan, Xiao, Yujie, and Seo, Yoonho. “A tabu search algorithm for simultaneous machine/AGV scheduling problem”. In: *International Journal of Production Research* 52.19 (Oct. 2, 2014), pp. 5748–5763. DOI: 10.1080/00207543.2014.910628.

[52] Zhou, Binghai and Lei, Yuanrui. “Bi-objective grey wolf optimization algorithm combined Levy flight mechanism for the FMC green scheduling problem”. In: *Applied Soft Computing* 111 (Nov. 1, 2021), p. 107717. DOI: 10.1016/j.asoc.2021.107717.