Department of Informatics
University of Fribourg, Switzerland

Complex Job Shop Scheduling: A General Model and Method

Thesis

presented to the Faculty of Economics and Social Sciences
at the University of Fribourg, Switzerland,
in fulfillment of the requirements for the degree of
Doctor of Economics and Social Sciences by

Reinhard Bürgy
from Gurmels (FR)

Accepted by the Faculty of Economics and Social Sciences
on February 24th, 2014 at the proposal of
Prof. Dr. Heinz Gröflin (first advisor)
Prof. Dr. Marino Widmer (second advisor)
Prof. Dr. Dominique de Werra (third advisor)

Fribourg, 2014

To my wife Emmi and
our precious daughter Saimi

ACKNOWLEDGEMENTS

This thesis would not have been possible without the countless contributions of many people, to whom I wish to express my sincerest gratitude. In particular, I am greatly indebted to:

My thesis supervisor Prof. Heinz Gröflin, for the opportunity to work in the area of Operations Research and to discover the fascinating world of scheduling. This thesis would not have been possible without his enthusiastic, acute, illuminating, patient support and guidance. His integrity and insights to education and science will always be a source of inspiration for me.

Prof. Marino Widmer and Prof. Dominique de Werra, for kindly accepting to be reviewers of my thesis.

Tony Hürlimann, not only for providing his excellent mathematical modeling language LPL, but also for many insightful discussions throughout the last years.

Prof. Pius Hättenschwiler and his former collaborators Matthias Buchs and Michael Hayoz for the precious time I could spend with them as an undergraduate assistant. Their encouraging guidance let me gain priceless experience and knowledge.

Ivo Blöchliger, Christian Eichenberger, Andreas Humm, Stephan Krenn, Antoine Legrain, Marc Pouly and Marc Uldry for their friendly support.

My parents Bernadette and Rudolf and my sister Stefanie, for their love and unfaltering support and encouragement. Their warm generosity and precious values will always be an example to me.

My wife Emmi, for being with me. With all my heart, I thank you for your presence, patience, warmth and love.

ABSTRACT

Scheduling is a pervasive task in planning processes in industry and services, and has become a dedicated research field in Operations Research over the years. A core of standard scheduling problems, models and methods has been developed, and substantial progress has been made in the ability to tackle these difficult combinatorial optimization problems.

Nevertheless, applying this body of knowledge is often hindered in practice by the presence of features that are not captured by the standard models, and practical scheduling problems are often treated ad-hoc in their application context. This "gap" between theory and practice has been widely acknowledged also in scheduling problems of the so-called job shop type. This thesis aims at contributing to narrow this gap.

A general model, the Complex Job Shop (CJS) model, is proposed that includes a variety of features from practice, including no (or limited number of) buffers, routing flexibility, transfer and transport operations, setup times, release times and due dates. The CJS is then formulated as a combinatorial problem in a general disjunctive graph based on a job template and an event-node representation.

A general solution approach is developed that is applicable to a large class of CJS problems. The method is a local search heuristic characterized by a job insertion based neighborhood and named JIBLS (Job Insertion Based Local Search). A key feature of the method is the ability to consistently and efficiently generate feasible neighbor solutions, typically by moving a critical operation (keeping or changing the assigned machine) together with other operations whose moves are "implied". For this purpose, the framework of job insertion with local flexibility, introduced in [45] and the insertion theory developed by Gröflin and Klinkert [42] are used. The metaheuristic component of the JIBLS is of the tabu search type.

The CJS model and the JIBLS method are validated by applying them to a selection of complex job shop problems. Some of the selected problems have been studied by other authors and benchmarks are available, while the others are new. Among the first are the Flexible Job Shop with Setup Times (FJSS), the Job Shop with

Transportation (JS-T) and the Blocking Job Shop (BJS), and among the second are the Flexible Blocking Job Shop with Transfer and Setup Times (FBJSS), the Blocking Job Shop with Transportation (BJS-T) and the Blocking Job Shop with Rail-Bound Transportation (BJS-RT). Of particular interest is the BJS-RT, where the transportation robots interfere with each other, which is, to our knowledge, the first generic job shop scheduling problem of this type.

For the problems with available benchmarks it is shown that the JIBLS is competitive with the best (often problem-tailored) methods of the literature. Moreover, the JIBLS appears to perform well also in the new problems and provides first benchmarks for future research on these problems.

Altogether, the results obtained provide evidence for the broad applicability of the CJS model and the JIBLS, and for the good performance of the JIBLS compared to the state of the art.

CONTENTS

ACRONYMS

BJS	Blocking Job Shop
BJSS	Blocking Job Shop with Transfer and Setup Times
BJS-RT	Blocking Job Shop with Rail-Bound Transportation
BJS-T	Blocking Job Shop with Transportation
CJS	Complex Job Shop
JS	Job Shop
JSS	Job Shop with Setup Times
JS-T	Job Shop with Transportation
FBJS	Flexible Blocking Job Shop
FBJSS	Flexible Blocking Job Shop with Transfer and Setup Times
FJS	Flexible Job Shop
FJSS	Flexible Job Shop with Setup Times
FNWJS	Flexible No-Wait Job Shop
FNWJSS	Flexible No-Wait Job Shop with Setup Times
JIBLS	Job Insertion Based Local Search
MS	Machine Scheduling
MIP	Mixed Integer Linear Programming
MPS	Master Production Schedule
MRP	Material Requirement Planning
MRP II	Manufacturing Resource Planning II
NWJS	No-Wait Job Shop
NWJSS	No-Wait Job Shop with Setup Times
NWJS-T	No-Wait Job Shop with Transportation
RCPS	Resource-Constrained Project Scheduling
SCP	Short Cycle Property

CHAPTER 1

INTRODUCTION

The topic of the thesis, complex job shop scheduling, is introduced in this chapter as follows. Section 1.1 describes the concept of scheduling. Section 1.2 presents common scheduling activities in practice. Some generic scheduling problems are specified in Section 1.3, starting with the resource-constrained project scheduling problem, and specializing it to the machine scheduling problem and the classical job shop scheduling problem, which is the basic version of the problems treated in this thesis. A variety of complexifying features that arise in practice and are not taken into account in the classical job shop are discussed in Section 1.4. Finally, Section 1.5 gives an overview of the thesis.

This chapter is mainly based on the operations management textbooks of Jacobs et al. [24] and Stevenson [108], and on the scheduling textbooks of Baker and Trietsch [7], Blazewicz et al. [11], Brucker and Knust [17] and Pinedo [95, 96].

1.1. Scheduling

Schedules are part of our professional and personal life. They answer the question of knowing when specific activities should happen and which resources are used. We mention just a few examples: bus schedules, also called bus timetables, contain arrival and departure times for each bus and bus stop; school timetables consist of the schedule for each class indicating lessons, teachers, rooms and time; project schedules comprise start and finish times of their activities; tournament schedules indicate which teams play against each other at which time and location; production schedules provide information on the orders stating when they should be executed on which equipment.

The term scheduling refers to the process of generating a schedule, which is commonly described as follows. The objects that are scheduled are called activities. Activities are somehow interrelated by so-called technological restrictions, e.g. some activities must be finished before others can start. For its execution each activity needs some resources which may be chosen from several alternative resources. The resources typically have limited capacity. A schedule consists of an allocation of resources and a starting time for each activity so that the capacity and technological restrictions are satisfied. The goal in scheduling is to find an optimal schedule, i.e. a schedule that optimizes some objective. The objective is typically related to time, for example minimizing the makespan, i.e. the overall time needed to execute all activities, minimizing the throughput times or minimizing the setup times.

Scheduling is performed in every organization. It is the final planning step before the actual execution of the activities. Hence, it links the planning and execution phases. While a short time horizon is often considered, detailed schedules are needed long before their actual execution in some industries (e.g. in the pharmaceutical sector). As future is uncertain, schedules may have to be revised quite frequently, for example due to changes in the shop floor or new order arrivals. Typically, scheduling is done over a rolling time horizon, the earlier part of the schedule being frozen and the remaining part being rescheduled.

Schedules are sometimes generated in an ad-hoc, rather informal way, using e.g. rules of thumb and blackboards. However, a systematic approach is needed for many scheduling problems in order to be able to cope with its complexity. Models and information systems may support the decision makers, allowing to find good or optimal schedules. Two types of models can be distinguished: descriptive models offering what if analyses and optimization models attempting to answer what's best. The models may be embedded in information systems that range from simple spreadsheets of restricted functionality to elaborate decision support systems that offer various (graphical) representations of the models and solutions, interaction of the users and collaborative decision making. Which level of support is needed mainly depends on the difficulty and importance of the scheduling problem.

1.2. Some Scheduling Activities in Practice

In this section, some scheduling activities commonly arising in practice are described.

1.2.1. Scheduling in Production Planning and Control

A typical scheduling task in manufacturing companies with a make to stock strategy concerns production planning and control, for instance in a Manufacturing Resource Planning II (MRP II) approach, which is depicted in Figure 1.1 and explained here briefly.

MRP II is hierarchically structured. At the top level, it comprises the following three phases: i) master planning, ii) detailed planning, and iii) short term planning and control.

Master planning considers a long term planning horizon (i.e. typically up to a year) and consists of sales and operations planning and master scheduling. Sales and operations planning establishes a sales plan and a production plan. On an aggregated

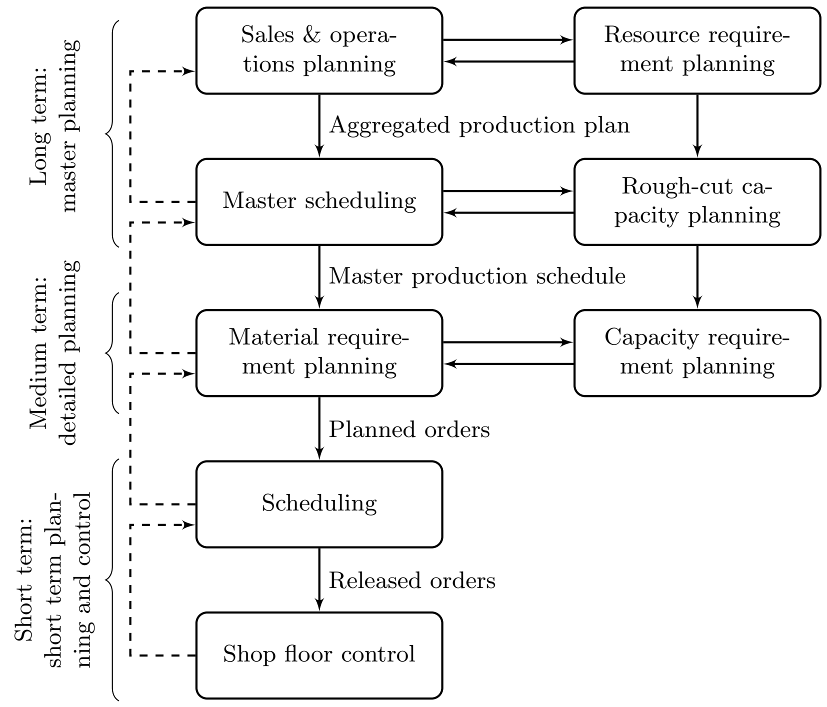

Long term:
master planning
Medium term:
detailed planning
Short term:
short term planning and control
Sales & operations planning
Resource requirement planning
Aggregated production plan
Master scheduling
Rough-cut capacity planning
Master production schedule
Material requirement planning
Capacity requirement planning
Planned orders
Scheduling
Released orders
Shop floor control

Figure 1.1.: Production planning and control phases according to MRP II, adapted from Schönsleben [102]. Tasks are depicted in rounded boxes. The lines indicate the exchange of information between the tasks.

level they describe the expected demands and the quantities that should be produced (or bought) for each product family and time unit. Resource requirements are also considered when establishing the production plan, looking at aggregated resources and focusing on critical resources. Based on the aggregated production plan, inventory stock levels and stock policies, the Master Production Schedule (MPS) is established in the master scheduling, stating the needed quantity for each end product and time period of the planning horizon. Feasibility of the MPS is considered in rough-cut capacity planning. If capacities are not sufficient, the MPS may be revised.

The detailed planning phase links the master planning with short term planning and control. It considers a medium term planning horizon (i.e. typically some months) and consists of the Material Requirement Planning (MRP) and the capacity requirement planning. Based on the MPS, bills of material, inventory stock levels and production lead time estimations, the MRP determines the needed quantity to fulfill the MPS for each component (raw material, parts, subassemblies) and time period. The outputs of this phase are planned (production and purchase) orders. Capacity requirement planning checks if enough capacity is present to produce the orders as planned. If capacities are not sufficient, the planning may be revised or capacities may be adjusted.

The final planning phase consists of scheduling the planned orders for the next days and weeks. A fine-grained schedule is established, determining the execution time and the assigned resources for each processing step of the planned orders. The schedule must be feasible; particularly, it must satisfy the capacity restrictions. The capacities of the resources are considered on a fine-grained level, e.g. a resource can execute at most one order at any time. To guarantee feasibility, the characteristic features of the production system must be considered in scheduling.

The outputs of the scheduling phase, called released orders, are then used in the shop floor to control the actual production.

Note that feedbacks from a phase to previous phases are common as indicated by the dashed lines in Figure 1.1. For example, unexpected events in the shop floor, such as machine failures or quality problems, may force to reschedule some orders. Minor problems may be dealt with at the control level. Whenever encountering major problems, the control level feeds back the inputs to the scheduling and a new schedule is generated.

At first glance, the decisions in scheduling appear to have a limited scope compared to, for example, system design decisions and longer term planning decisions. This view does not reflect the high impact of scheduling in production planning and control. In fact, good scheduling may lead to cost reductions and a greater flexibility in previous planning phases, such as working with less machines or accepting more customer orders, and bad scheduling may lead to due date violations and idle times so that costly actions are needed, such as the purchase of new machines or overtime work. Furthermore, scheduling may also reveal problems in the system design such as bottlenecks and problems in the longer term planning such as too optimistic production lead time and due date estimations.

Several trends, including mass customization and the adoption of complex automated production systems, suggest that the importance and difficulty of scheduling problems in production planning will increase in the future.

1.2.2. Project Scheduling

Projects are unique, one-time operations set up to achieve some objectives given a limited amount of resources such as money, time, machines and workers.

Projects arise in every organization. Typical examples are the development of a product, the construction of a factory and the development and integration of software.

The management of a project consists in planning, controlling and monitoring the project so that quality, time, cost and other project requirements are met. Important aspects in project planning include breaking down the project into smaller components, such as sub-projects, tasks, work packages and activities, and establishing a schedule of the project.

Scheduling a project consists in assigning starting times to all activities and assigning resources to the activities so that a set of goals are achieved and all constraints are satisfied. The goals and constraints are typically related to time, resources or costs. For example, the project duration may be minimized, the activities are interrelated, e.g. by precedence constraints, and can have deadlines, a smooth resource utilization may be sought, and total costs may be minimized.

1.2.3. Workforce Scheduling

In many organizations, a work plan for the work force must be established, determining for all workers when they are working and what they are working on. The goal may be to minimize costs.

The workers have to be scheduled so that the needed demand is met, e.g. enough workers are assigned to execute the production plan or to serve the customers. A variety of specific working constraints make the problem different from other scheduling problems. A machine may be used during the whole day and seven days a week. Workers, however, have specific working restrictions, such as a work time limit per day and week, and regulations on breaks and free days.

1.2.4. Scheduling Reservations and Appointments

In the service industry, services are sometimes requested and reserved prior to their consumption. Besides decisions on the timing and the allocation of resources, the reservation process allows controlling the acceptance of the requests.

Two main types of scheduling problems are distinguished. The first type, called reservation scheduling, occurs if the customers have no or almost no flexibility in time, i.e. the time of the service consumption is fixed. The main decision is whether to accept or deny a service request, and the objective is often to maximize resource utilization. Such types of decisions are quite common in many sectors including the transportation industry (car rentals, air and train transportation) and hotels.

The second type, called appointment scheduling, consists of scheduling problems where the customers have more flexibility in time. The main decision is the timing of the service. Appointment scheduling problems are common in practice, e.g. for business meetings, appointments at the doctor's office and at the hairdresser.

In both types of problems, cancellations and customers that do not show up or show up late must be taken into account, increasing the complexity of the problem.

1.2.5. Pricing and Revenue Management

In an integrated scheduling approach, service requests are not just accepted or denied, but variable pricing is established to control demand. The prices are typically set by analyzing specifics of the request, left capacities and demand forecasts. This so-called tactical pricing leads to different prices for the same or similar services. An increase of the profit, better resource utilization and the gain of new customers are some of the goals of tactical pricing.

While not being a standard technique in manufacturing, tactical pricing is commonly used in the service industry. Well-known examples are the pricing of flight tickets and hotel accommodations. For example, a flight request of a customer three days before departure is likely to be priced higher than the same request made three months earlier.

When services are requested prior to consumption, pricing and reservation scheduling may be combined, forming so-called revenue management problems. These problems are well-known in the airline and hotel industries.

Tactical pricing is also present in sectors with no service reservation such as in supermarkets and coffee shops. The prices may depend on the channels (e.g. are cheaper through the internet), the time of consumption (e.g. the coffee is cheaper in the morning), or product variations are created (e.g. budget or fine food lines).

1.3. Some Generic Scheduling Problems

As described in the previous section, scheduling is a pervasive activity in practice. In scheduling theory, a core of generic scheduling problems, models and methods has been developed for solving these problems. In this section we introduce some generic scheduling problems, starting with the resource-constrained project scheduling problem, and specializing it to the machine scheduling problem and the classical job shop scheduling problem.

1.3.1. The Resource-Constrained Project Scheduling Problem

A general scheduling problem is the Resource-Constrained Project Scheduling (RCPS) problem, which can be specified as follows. Given is a set of activities I and a set of resources R. Each resource $r \in R$ is available at any time in amount B_r. Each activity $i \in I$ requires an amount b_{ri} of each resource $r \in R$ during its non-preemptive execution of duration $d_i \geq 0$. Some activities must be executed before others can start. These so-called precedence constraints are given by a set P of pairs of activities $(i, j), i, j \in I$, where i has to be completed before j can start. The objective is to specify a starting time α_i for each activity $i \in I$ so that the described constraints are satisfied and the project duration is minimized.

Introduce a fictive start activity σ and a fictive end activity τ and let $I^+ = I \cup \{\sigma, \tau\}$. Activities σ and τ are of duration 0 and must occur before, respectively after all activities of I. Then, the RCPS problem can be formulated as follows:

$$\text{minimize } \alpha_\tau \tag{1.1}$$

$$\text{subject to:}$$

$$\alpha_i - \alpha_\sigma \geq 0 \text{ for all } i \in I, \tag{1.2}$$

$$\alpha_\tau - \alpha_i \geq d_i \text{ for all } i \in I, \tag{1.3}$$

$$\alpha_j - \alpha_i \geq d_i \text{ for all } (i, j) \in P, \tag{1.4}$$

$$\text{For all } r \in R:$$

$$\sum_{i \in I : \alpha_i \leq t \leq \alpha_i + d_i} b_{ri} \leq B_r \text{ at any time } t, \tag{1.5}$$

$$\alpha_\sigma = 0. \tag{1.6}$$

Any feasible solution $\alpha \in \mathbb{R}^{I^+}$ is called a *schedule*. The starting time α_σ is set to 0 (1.6) and α_τ reflects the project duration, which is minimized (1.1). Constraints (1.2), (1.3) and (1.4) ensure that all activities are executed in the right order between start and end. The capacity constraints (1.5) ensure that at any time, the activities in execution require no more of resource r than available.

Clearly, the capacity constraints (1.5) are not tractable in this form. Other, more tractable versions exist (see e.g. Brucker and Knust [17]). However, regardless of the formulation, the RCPS problem is a difficult problem to solve.

1.3.2. The Machine Scheduling Problem

An important special type of RCPS is the Machine Scheduling (MS) problem, where each resource $r \in R$ has capacity $B_r = 1$ and each activity has a requirement $b_{ri} = 0$ or $b_{ri} = 1$ for each $r \in R$. Consequently, a resource is either free or occupied by an activity, and for any two distinct activities i, j needing a same resource r, i must precede j or vice versa.

Let Q be a set containing all unordered pairs $\{i, j\}$ of distinct operations $i, j \in I$ needing a same resource, i.e. for some $r \in R, b_{ri} = b_{rj} = 1$. Then, the capacity constraints (1.5) can be rewritten in the substantially simpler form of disjunctive constraints:

$$\alpha_j - \alpha_i \geq d_i \text{ or } \alpha_i - \alpha_j \geq d_j \text{ for all } \{i, j\} \in Q, \tag{1.7}$$

and the MS problem can be formulated as a disjunctive program by (1.1)-(1.4), (1.6) and (1.7).

As early studies of the MS problem were in an industrial context, the resources are called *machines* and the activities are *operations*. These terms have become standard and will be used in this thesis, although a resource might not be a machine but any other processor, such as a buffer or a mobile device. In line with common notation, we use letter M for the set of machines (instead of letter R).

1.3.3. The Classical Job Shop Scheduling Problem

An important special type of MS is the classical Job Shop (JS) problem, which has the following features:

- Any operation $i \in I$ needs exactly one machine, say $m_i \in M$, for its execution. Consequently, the set Q contains all unordered pairs $\{i, j\}$ with $i, j \in I, i \neq j$ and $m_i = m_j$.
- The set I of operations is partitioned into a set of jobs $\mathcal{J}$: A job $J \in \mathcal{J}$ is a set of operations $\{i : i \in J\}$ and each operation $i \in I$ is in exactly one job $J \in \mathcal{J}$.
- The set of operations of each job $J \in \mathcal{J}$ is ordered in a sequence, i.e. $\{i : i \in J\}$ is sometimes referred to as the ordered set $\{J_1, J_2, \ldots, J_{|J|}\}$, J_r denoting the r-th operation of job J.
- The operations of a job $J \in \mathcal{J}$ have to be processed in sequence, i.e. J_r must be finished before J_{r+1} starts, $r = 1, \ldots, |J| - 1$. No other precedence constraints exist. Consequently, the set P consists of all pairs (J_r, J_{r+1}), $1 \leq r < |J|, J \in \mathcal{J}$.

Denote by I^{first} and I^{last} the subsets of operations that are first and last operations of jobs, respectively, and call two operations i, j of a job J to be consecutive if $i = J_r$ and $j = J_{r+1}$ for some r, $1 \leq r < |J|$. Then, constraints (1.2), (1.3) and (1.4) can be rewritten as (1.9), (1.10) and (1.11), respectively, giving the following formulation of the JS as a disjunctive program:

$$\text{minimize } \alpha_\tau \tag{1.8}$$

subject to:

$$\alpha_i - \alpha_\sigma \geq 0 \text{ for all } i \in I^{\text{first}}, \tag{1.9}$$

$$\alpha_\tau - \alpha_i \geq d_i \text{ for all } i \in I^{\text{last}}, \tag{1.10}$$

$$\alpha_j - \alpha_i \geq d_i \text{ for all } i, j \in I \text{ consecutive in some job } J, \tag{1.11}$$

$$\alpha_j - \alpha_i \geq d_i \text{ or } \alpha_i - \alpha_j \geq d_j \text{ for all } \{i, j\} \in Q, \tag{1.12}$$

$$\alpha_\sigma = 0. \tag{1.13}$$

A small job shop example is introduced in Figure 1.2. It consists of three machines and three jobs, each visiting all machines exactly once. The figure depicts a solution with makespan 14 in a Gantt chart. The bars represent the operations that are indicated by the attributed numbers, e.g. the bar with number 2.3 refers to the third operation of job 2. The routing of the jobs as well as the processing durations can be read directly in the chart.

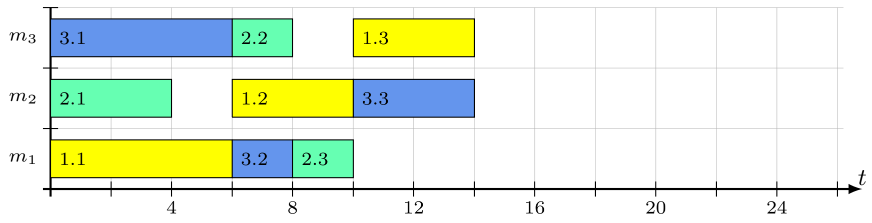


Figure 1.2.: A solution of a small job shop example.

The term "job shop" refers to a manufacturing process type that is generally used when a rather low volume of high-variety products is produced. The high flexibility established by general-purpose machines is a main characteristic of the job shop. The flexibility makes it possible to treat jobs that differ considerably in processing requirements including processing steps, processing times and setups.

Job shop scheduling problems or variations of it arise in industries adopting job shop or similar production systems such as the chemical and pharmaceutical sectors [100], the semiconductor industry [78] and the electroplating sector [68]. Job shop scheduling problems are also present in the service sector such as in transportation systems [71], in health care [93] and in warehouses [62].

1.4. Extensions of the Classical Job Shop

Although a core of generic scheduling problems, models and methods has been developed, many practical scheduling problems are treated ad-hoc in an application context, as also mentioned by Pinedo in [96], p. 431: "It is not clear how all this knowledge [about generic models and methods] can be applied to scheduling problems in the real world. Such problems tend to differ considerably from the stylized models studied by academic researchers." A typical obstacle in using generic scheduling models and methods are features arising in practice that are not included in the models (cf. Pinedo [96], p.432).

This is all the more true for the JS. In this section, we informally introduce the following features that arise in practice and are not taken into account in the JS: setup times, release times and due dates, limited number of buffers, transfer times, time lags, routing flexibility and transports.

1.4.1. Setup Times

After completion of an operation, a machine may be set up, i.e. made ready, for the next operation by various tasks including machine cleaning, tool changing and temperature adjusting. An initial machine setup before the processing of the first operation and a final setup after the last operation for setting a machine in desired end conditions may also be necessary.

m_1	*1.1*	*2.3*	*3.2*
1.1	0	2	3
2.3	0	0	2
3.2	0	2	0

m_2	*1.2*	*2.1*	*3.3*
1.2	0	0	2
2.1	3	0	2
3.3	3	2	0

m_3	*1.3*	*2.2*	*3.1*
1.3	0	3	3
2.2	2	0	0
3.1	2	2	0

Table 1.1.: Setup times on machines m_1 (left), m_2 (middle) and m_3 (right). The first operation is specified by the row, the second by the column.

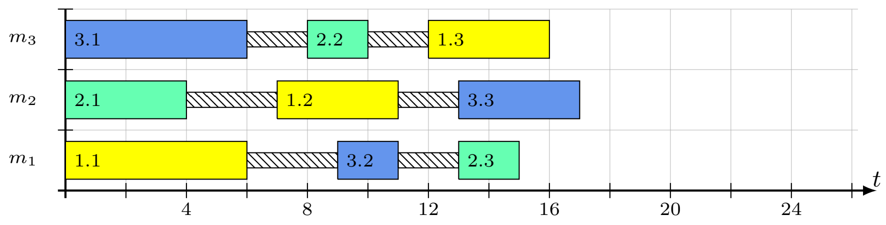


Figure 1.3.: A solution of an example with setup times.

The time needed for the setup, called setup time or change over time, may depend on the previous and next operation to be executed on the machine. Such setup times are called sequence-dependent. If the setup time depends just on the next operation to be executed, it is called sequence-independent.

In the JS, setup times are assumed to be negligibly small or included in the operation's processing times. Sequence-dependent setup times are not taken into account.

Setup times, particularly sequence-dependent setup times, are common in practice, occurring for example in the printing, dairy, textile, plastics, chemical, paper, automobile, computer and service industries as stated by Allahverdi et al. [3].

Let us introduce sequence-dependent setup times into the example of Figure 1.2. For each ordered pair of operations executed on the same machine a setup time is defined in Table 1.1. For instance, if operation 2.2 is executed directly after 3.1 on machine m_3 then a setup time of 2 occurs, see row 3.1, column 2.2 in block m_3.

A solution is depicted in Figure 1.3. The processing sequences on the machines are the same as in the solution of the JS (see Figure 1.2). Setups are depicted by narrow, hatched bars. Due to the setup times, starting and finishing times of the operations have changed and the makespan has increased from 14 to 17.

1.4.2. Release Times and Due Dates

In the JS it is assumed that all jobs are available in the beginning of the planning horizon. In practice, however, jobs may not be able to start at time 0 for various reasons including dynamic job arrivals or prior planning decisions. The earliest time a job can start is called release time.

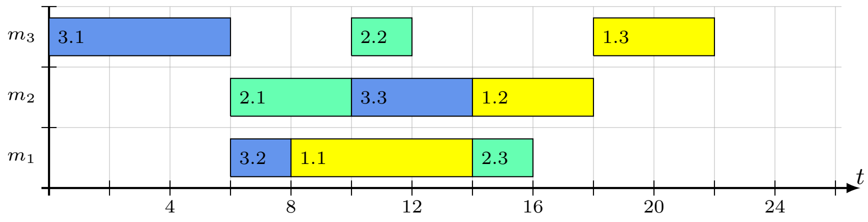


Figure 1.4.: A solution of an example with release times and due dates.

Furthermore, a job may have to be finished at some given time, called due date, for instance due to delivery time commitments to customers and planning decisions (e.g. prioritization of the jobs).

For each job, its release time and due date specifies a time window within which its operations should or must be executed. Completion after the due date is generally allowed but such lateness is penalized. A due date that must be satisfied is called deadline (cf. Pinedo [96], p. 14).

Consider time windows in the example of Figure 1.2. Assume that for job 1, 2 and 3, the release time is 7, 6 and 0, and the due date is 23, 16 and 14, respectively. The solution depicted in Figure 1.4 respects these times.

1.4.3. Limited Number of Buffers and Transfer Times

After completion of an operation, a job is either finished, goes directly to its next machine or waits somewhere until its next machine becomes available. Buffers, also called storage places, may be available for the jobs that must wait. If no buffer is available or all buffers are occupied, a job may also wait on its machine, thus blocking it, until a buffer or the next machine becomes available.

Transferring a job from a machine to the next (or to a buffer) needs some time during which both resources are simultaneously occupied. This time is called transfer time.

In the JS it is assumed that an unlimited number of buffers is available, and transfer times are negligible or part of the processing times. In practice, however, the number of buffers is limited for various reasons. Buffers may be expensive or inadequate for technological reasons, or the number of buffers is limited in order to efficiently limit and control work-in-process.

Many systems in practice have limited or no buffers, for example flexible manufacturing systems [107], robotic cells [30], electroplating lines [73], automated warehouses [62] and railway systems [88].

Consider no buffers and transfer times 0 for all transfers in the example of Figure 1.2. A solution is shown in Figure 1.5. Blockings are depicted by narrow bars filled in the color of the job that is blocking the machine. For example, job 2 waits on machine m_2 from time 4 to 6.

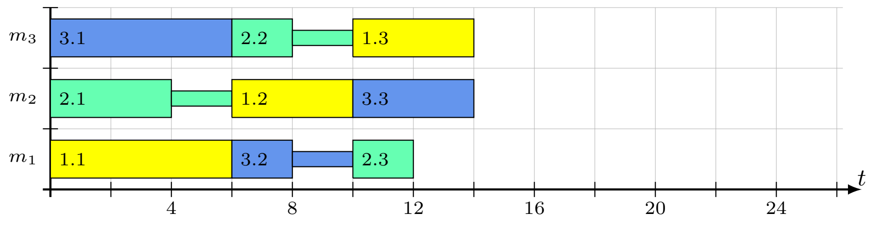


Figure 1.5.: A solution of an example without buffers and with transfer times 0.

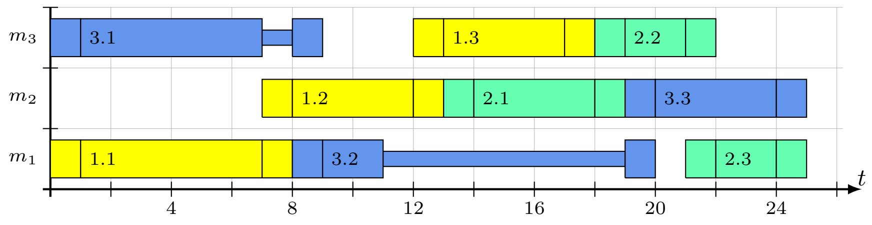


Figure 1.6.: A solution of an example without buffers and with transfer times 1.

Now consider no buffers and transfer times 1 for all transfers in the example of Figure 1.2. A solution is shown in Figure 1.6. Transfers are depicted by bars in the color of the job. Note that the processing sequences on the machines have changed compared to the solution in Figure 1.5 since those sequences are infeasible in case of positive transfer times. Indeed, the three jobs swap their machines at time 6 in the solution with transfer times 0. It is easy to see that such “swaps” are not feasible in the case of positive transfer times.

1.4.4. Time Lags and No-Wait

Consecutive operations in a job are sometimes not just coupled by precedence relations as in the JS, but the time after the end of one operation and the beginning of the next may be restricted by minimum and maximum time lags.

Clearly, a precedence relation between consecutive operations in a job can be modeled with such time lags by setting the minimum time lag to 0 and the maximum time lag to infinity (not present). If the minimum time lag is negative, the next operation might start before the previous is finished. If both (minimum and maximum) time lags are 0, the relation is referred to as no-wait, imposing that the job’s next operation has to start exactly when the previous operation is finished. More generally, if the minimum and maximum time lag is equal, then the relation is referred to as fixed time lag or generalized no-wait.

In practice, time lags arise in various industries. Particularly important is this feature if chemical reactions are present (e.g. in the chemical and pharmaceutical sectors [100], semiconductor and electroplating industries [68]), and if properties such as temperature and viscosity have to be maintained (e.g. in the metalworking industry [22]).

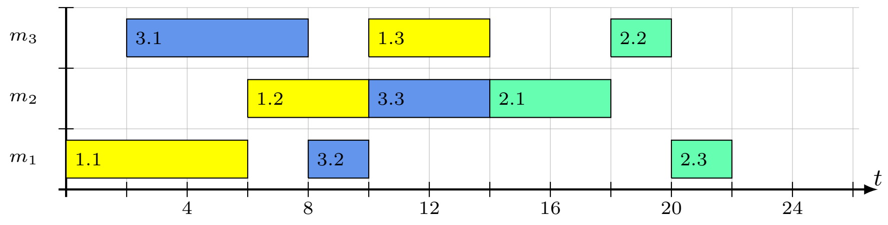


Figure 1.7.: A solution of an example with no-wait relations.

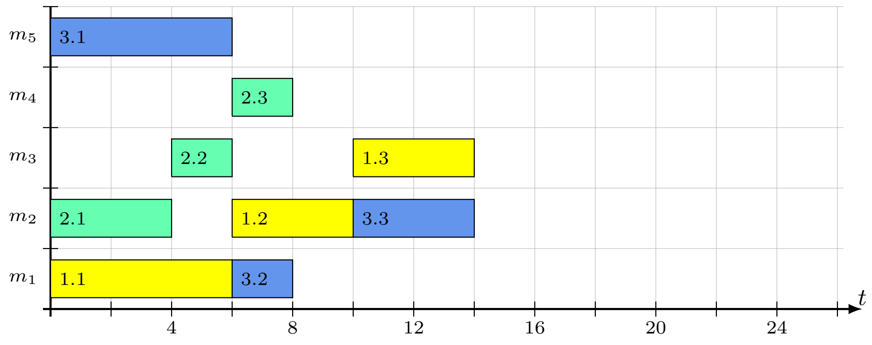


Figure 1.8.: A solution of an example with routing flexibility.

Consider the example of Figure 1.2 and introduce a no-wait relation between any two consecutive operations in a job. A solution of this example is depicted in Figure 1.7.

1.4.5. Routing Flexibility

In the JS, each operation is processed on a dedicated machine. In practice, however, an operation may be performed by various machines and one of these alternatives must be selected. This feature is commonly called routing flexibility. In a job shop with routing flexibility, not only starting times of the operations, but also an assignment of a machine for each operation must be specified.

Routing flexibility is mainly achieved by the availability of multiple identical machines and by a machine's capability of performing different processing steps. This feature can be found in almost all industries. Typical examples are multi-purpose plants in process industries [100] and flexible manufacturing systems [72].

Consider the example of Figure 1.2 and duplicate machines m_1 and m_3, i.e. add two additional machines, called m_4 and m_5, able to execute operations 1.1, 2.3, 3.2 and 1.3, 2.2, 3.1, respectively. A solution of this example is depicted in Figure 1.8.

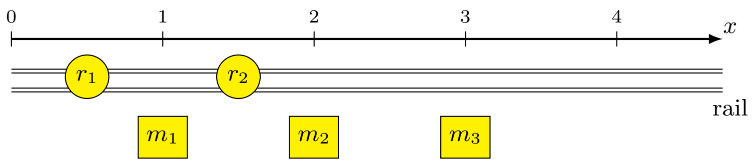


Figure 1.9.: The layout of a transportation system.

1.4.6. Transports

After completion of an operation, a job has to be transported to its next machine if this machine is not at the same location as the current machine. In the JS, such transport steps are neglected or included in the processing times. In many practical cases, however, transports need to be considered, e.g. because they take a considerable amount of time, or only a limited number of mobile devices is available to execute the moves.

Mobile devices often interfere with each other in their movements. Typically, interferences arise when these devices move in a common transportation network. Apart from scheduling the processing and transport operations, also feasible trajectories of the mobile devices must be determined in such cases.

In practice, versions of job shop problems with transportation arise in various sectors, for example in electroplating plants [73], in the metalworking industry [40], in container terminals [82], and in factories with overhead cranes [5] and robotic cells [35].

Consider the example of Figure 1.2. Introduce the transportation system sketched in Figure 1.9 consisting of robots that transport the jobs from one machine to the next. The robots move on a common rail line with a maximum speed of one. They cannot pass each other and must maintain a minimum distance from each other of one. Machines m_1, m_2 and m_3 are located along the rail at the locations 1, 2 and 3, respectively (measured on the x-axis). There are no buffers available. Transferring a job from a machine to a robot, or vice versa, takes one time unit.

A solution with one and two robots is depicted in Figure 1.10 and 1.11, respectively. The horizontal and vertical axis stands for the time and the location on the rail, respectively. The machining operations are depicted as bars at the location representing the location of the corresponding machine. Transport operations are not shown, but can be inferred from the trajectories of the robots that are drawn by lines. Thick line segments indicate that the robot is loaded with a job executing a transport operation, while thin sections correspond to idle moves.

1.5. Overview of the Thesis

In view of the limited applicability of the JS in practice, richer job shop models and general solution methods for these models are needed to bridge the gap between theory and practice. In this thesis, we aim toward closing this gap by proposing a

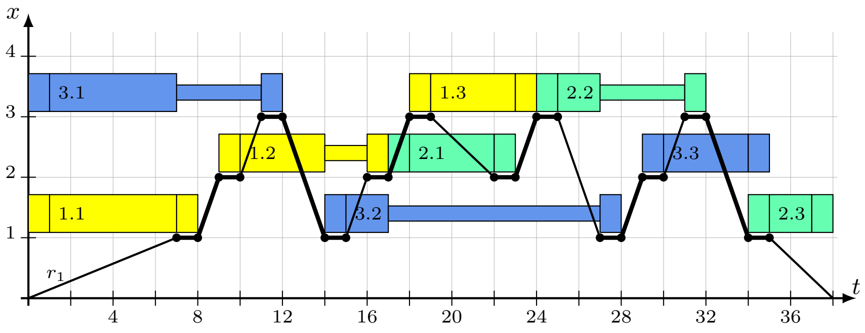


Figure 1.10.: A solution of an example with one robot.

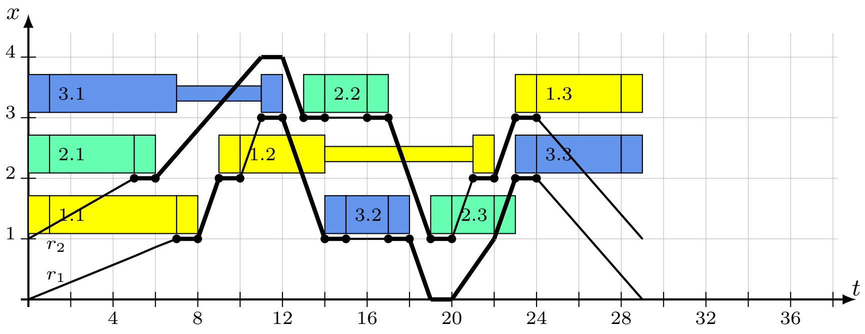


Figure 1.11.: A solution of an example with two robots.

complex job shop model and a solution method that can be applied to a large class of complex job shop problems.

The thesis consists of two parts. In Part I, a general model, called the Complex Job Shop (CJS) model, covering (to some extent) the practical features introduced in Section 1.4 is established and a formulation based on a disjunctive graph is developed in Chapter 2. Based on the disjunctive graph formulation, Chapter 3 develops a heuristic solution method that can be applied to a large class of CJS problems. The approach is a local search based on job insertion and is called the Job Insertion Based Local Search (JIBLS).

In Part II, the CJS model and the JIBLS solution method are tailored and applied to a selection of complex job shop problems obtained by extending the JS with a combination of the practical features described in Section 1.4. Some of these problems are known and some have not yet been addressed in the literature. In the known problems, the numerical results obtained by the JIBLS are compared to the best results found in the literature, and in the other problems first benchmarks are established and compared to results obtained by a Mixed Integer Linear Programming (MIP) approach.

The landscape of the selected complex job shop problems is provided in Figure 1.12. The vertical axis describes features defining the coupling of two consecutive operations in a job (i.e. buffers and time lags), and the horizontal axis provides the other addressed features (i.e. setups, flexibility and transportation). Each problem is represented by a box.

In Chapter 4, an extension of the JS characterized by sequence-dependent setup times and routing flexibility, called the Flexible Job Shop with Setup Times (FJSS), is considered. The FJSS and its simpler version without setup times, called the Flexible Job Shop (FJS), are known problems in the literature.

In Chapter 5, a version of the FJSS characterized by the absence of buffers, called the Flexible Blocking Job Shop with Transfer and Setup Times (FBJSS), is treated. Literature related to the FBJSS is mainly dedicated to its simpler version without flexibility and without setup times, called the Blocking Job Shop (BJS). While the BJS has found increasing attention over the last years, we are not aware of previous literature on the BJS with flexibility and with setup times, except for publication [45].

Chapter 6 addresses instances of FJSS and FBJSS problems where jobs are processed on machines in a sequence of machining operations and are transported from one machine to the next by mobile devices, called robots, in transport operations. We assume in this chapter that the robots do not interfere with each other, and consider two versions of the problem. In the first version, called the Job Shop with Transportation (JS-T), an unlimited number of buffers is available, and in the second version, called the Blocking Job Shop with Transportation (BJS-T), no buffers are available. While the JS-T is a standard problem in the literature, the BJS-T has not yet been addressed.

Building on the previous chapter, in Chapter 7 we address a version of the BJS-T with robots that interfere with each other in space, and call it the Blocking Job Shop with Rail-Bound Transportation (BJS-RT). The robots move on a single rail line along which the machines are located. The robots cannot pass each other, must maintain

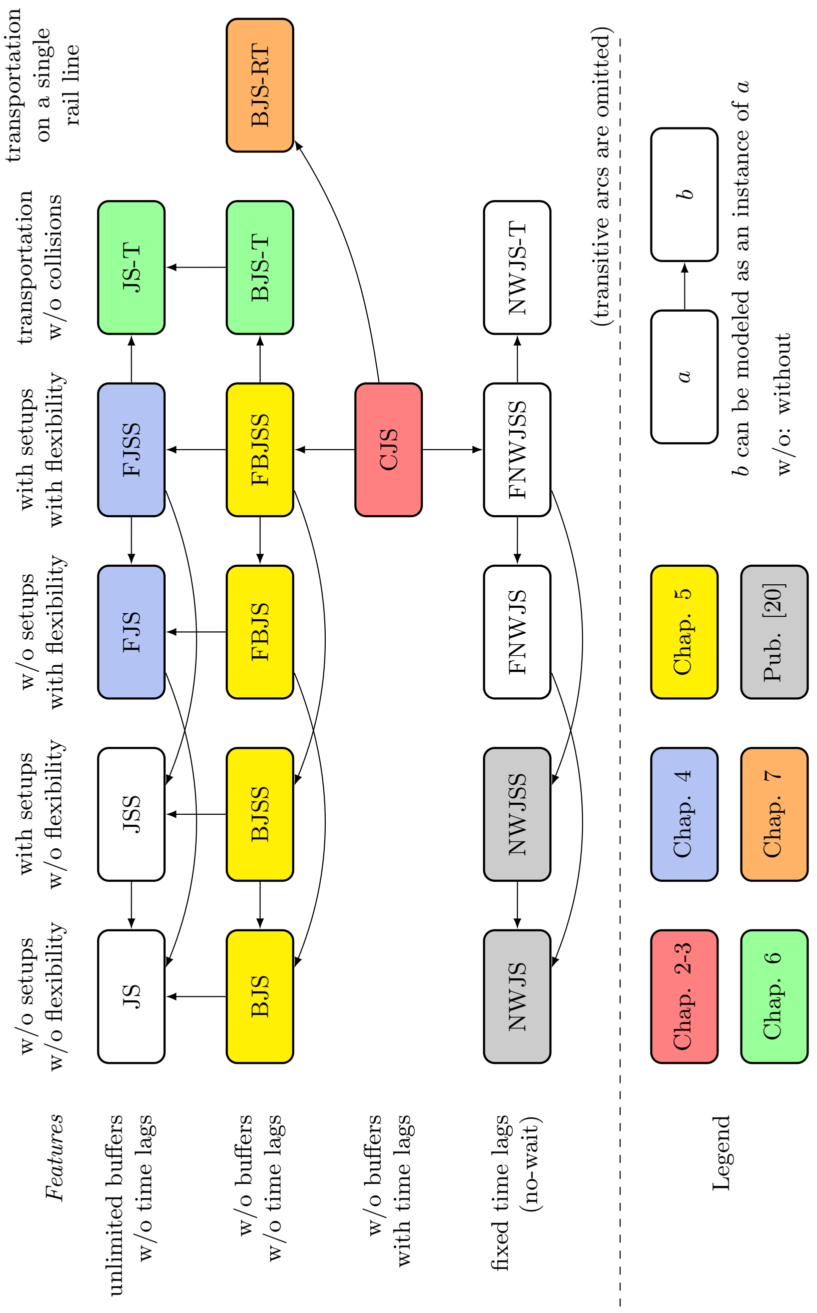

Features
unlimited buffers w/o time lags
w/o buffers w/o time lags
w/o buffers with time lags
fixed time lags (no-wait)
Legend
w/o setups w/o flexibility
with setups w/o flexibility
w/o setups with flexibility
with setups with flexibility
transportation w/o collisions
transportation on a single rail line
JS
JSS
FJS
FJSS
JS-T
BJS
BJSS
FBJS
FBJSS
BJS-T
BJS-RT
CJS
NWJS
NWJSS
FNWJS
FNWJSS
NWJS-T
(transitive arcs are omitted)
Chap. 2-3
Chap. 4
Chap. 5
Chap. 6
Chap. 7
Pub. [20]
a
b
b can be modeled as an instance of a
w/o: without

Figure 1.12.: A landscape of selected complex job shop problems.

a minimum distance from each other, but can “move out of the way”. Besides a schedule of the (machining and transport) operations, also feasible trajectories of the robots, i.e. the location of each robot at any time, must be determined. Building on the BJS-T and an analysis of the feasible trajectory problem, a formulation of the BJS-RT in a disjunctive graph is derived. Efficient algorithms to determine feasible trajectories for a given schedule are also developed. To our knowledge, the BJS-RT is the first generic job shop scheduling problem considering interferences of robots.

In Figure 1.12, complex job shop problems with fixed time lags, sometimes called generalized no-wait job shop problems, are also illustrated. These problems are not addressed in this thesis. However, the version with setups, called the No-Wait Job Shop with Setup Times (NWJSS) is addressed in the publication [20] and solved by a method that is also based on job insertion, but that is different to the approach taken in the JIBLS.

Part I.

Complex Job Shop Scheduling

In this part, a general model, called the Complex Job Shop (CJS) model, that covers (to some extent) the practical features introduced in Section 1.4 is established and a formulation based on a disjunctive graph is given. Furthermore, a local search heuristic based on job insertion is developed. It is applicable to a large class of CJS problems, and will be called the Job Insertion Based Local Search (JIBLS).

CHAPTER 2 MODELING COMPLEX JOB SHOPS

2.1. Introduction

In this chapter we develop a complex job shop model that covers the wide range of practical features discussed in the previous chapter: sequence-dependent setup times, release times and due dates, no buffers (blocking), transfers, time lags, and routing flexibility.

The chapter is organized as follows. First, standard formulations of the JS as a disjunctive program, as a mixed integer linear program and as a combinatorial optimization problem in a disjunctive graph are given in Section 2.2. Then, based on the article [42] of Gröflin and Klinkert, a general disjunctive scheduling model is presented in Section 2.3. This model does not "know" jobs and machines. We adapt the model in Section 2.4 by specializing it to capture essential features of the job shop such as machines and job structure, and by extending it to include routing flexibility. The obtained model also captures the aforementioned practical features and will be called the Complex Job Shop (CJS) model.

Graphs will be needed. They will be directed, unless otherwise stated, and the following standard notation will be used. An arc $e = (v, w)$ has a tail (node v), and a head (node w), denoted by $t(e)$ and $h(e)$ respectively. Also, given a graph $G = (V, E)$, for any $W \subseteq V$, $\gamma(W) = \{e \in E : t(e), h(e) \in W\}$, $\delta^-(W) = \{e \in E : t(e) \notin W$ and $h(e) \in W\}$, $\delta^+(W) = \{e \in E : t(e) \in W$ and $h(e) \notin W\}$ and $\delta(W) = \delta^-(W) \cup \delta^+(W)$. These sets are defined in G, we abstain however from a heavier notation, e.g. $\delta^+_G(W)$ for $\delta^+(W)$. It will be clear from the context which underlying graph is meant. Finally, in a graph $G = (V, E, d)$ with arc valuation $d \in \mathbb{R}^E$, a path (or cycle) in G of positive length will be called a *positive* path (or cycle) and a path of longest length a *longest* path.

2.2. Some Formulations of the Classical Job Shop

A number of formulations of the JS are given in well-known works (e.g. Manne [75], Balas [8], Adams et al. [1]) and in standard textbooks on scheduling (e.g. Blazewicz et al. [11], Brucker and Knust [17], Pinedo [96]). We recall here the disjunctive programming formulation given in Section 1.3, and derive from it standard formulations as a mixed integer linear program and as a combinatorial optimization problem in a disjunctive graph.

2.2.1. A Disjunctive Programming Formulation

Recalling that Q contains all unordered pairs $\{i, j\}$ of distinct operations $i, j \in I$ with $m_i = m_j$, the JS can be formulated as the following disjunctive program:

$$\text{minimize } \alpha_\tau \tag{2.1}$$
$$\text{subject to:}$$
$$\alpha_i - \alpha_\sigma \geq 0 \text{ for all } i \in I^{\text{first}}, \tag{2.2}$$
$$\alpha_\tau - \alpha_i \geq d_i \text{ for all } i \in I^{\text{last}}, \tag{2.3}$$
$$\alpha_j - \alpha_i \geq d_i \text{ for all } i, j \in I \text{ consecutive in some job } J, \tag{2.4}$$
$$\alpha_j - \alpha_i \geq d_i \text{ or } \alpha_i - \alpha_j \geq d_j \text{ for all } \{i, j\} \in Q, \tag{2.5}$$
$$\alpha_\sigma = 0. \tag{2.6}$$

For explanations we refer the reader to Section 1.3.

2.2.2. A Mixed Integer Linear Programming Formulation

A mixed integer linear programming formulation can be obtained in a straightforward way using the above disjunctive program and introducing a binary variable y_{ij} for each $\{i, j\} \in Q$, with the following meaning: y_{ij} is 1 if operation i is preceding j, and 0 otherwise. Letting B be a large number, the JS problem is the following MIP problem:

$$\text{minimize } \alpha_\tau \tag{2.7}$$
$$\text{subject to:}$$
$$\alpha_i - \alpha_\sigma \geq 0 \text{ for all } i \in I^{\text{first}}, \tag{2.8}$$
$$\alpha_\tau - \alpha_i \geq d_i \text{ for all } i \in I^{\text{last}}, \tag{2.9}$$
$$\alpha_j - \alpha_i \geq d_i \text{ for all } i, j \in I \text{ consecutive in some job } J, \tag{2.10}$$
$$\alpha_j - \alpha_i + B(1 - y_{ij}) \geq d_i \text{ for all } \{i, j\} \in Q, \tag{2.11}$$
$$\alpha_i - \alpha_j + By_{ij} \geq d_j \text{ for all } \{i, j\} \in Q, \tag{2.12}$$
$$y_{ij} \in \{0, 1\} \text{ for all } \{i, j\} \in Q, \tag{2.13}$$
$$\alpha_\sigma = 0. \tag{2.14}$$

2.2.3. A Disjunctive Graph Formulation

The JS is frequently formulated as a combinatorial optimization problem in a disjunctive graph. Each operation is represented by a node in this graph. The nodes of two consecutive operations in a job are linked by an arc representing the precedence constraint between these two operations. The nodes of two operations using a common machine are linked by a pair of disjunctive arcs representing the corresponding disjunctive constraint.

Specifically, the disjunctive graph $G = (I^+, A, E, \mathcal{E}, d)$ is constructed as follows. Each operation $i \in I^+ = I \cup \{\sigma, \tau\}$ is represented by a node and identify a node with the operation it represents.

The set of conjunctive arcs A consists of the following arcs representing the set of constraints (2.2)-(2.4) of the disjunctive programming formulation: (i) for each $i \in I^{\text{first}}$, an initial arc (σ, i) of weight 0, (ii) for each $i \in I^{\text{last}}$, a final arc (i, τ) of weight d_i, and (iii) for any two consecutive operations i, j in some job J, an arc (i, j) of weight d_i.

The set of disjunctive arcs E representing constraints (2.5) of the disjunctive program is given as follows. For any two distinct operations $i, j \in I$ with $m_i = m_j$, i.e. $\{i, j\} \in Q$, there are two disjunctive arcs $(i, j), (j, i)$ with respective weights d_i, d_j. The family $\mathcal{E}$ consists of all introduced pairs $\{(i, j), (j, i)\}$ of disjunctive arcs.

Definition 1 *Any set of disjunctive arcs $S \subseteq E$ is called a* selection *in G. A selection S is* complete *if $S \cap D \neq \emptyset$ for all $D \in \mathcal{E}$. Selection S is* positive acyclic *if subgraph $G(S) = (V, A \cup S, d)$ contains no positive cycle, and is* positive cyclic *otherwise. Selection S is* feasible *if it is positive acyclic and complete.*

Then, the JS is the following problem:

Among all feasible selections, find a selection S minimizing the length of a longest path from σ to τ in $G(S) = (V, A \cup S, d)$.

The disjunctive graph formulations given in some articles and textbooks differ slightly from the above formulation.

First, a disjunctive arc pair is sometimes represented by an undirected arc (an edge), and a complete selection is defined by specifying a direction for each edge, asking $|S \cap D| = 1$ for all $D \in \mathcal{E}$ (see Brucker and Knust [17]).

Second, the distinction between cycles and *positive* cycles is not needed in the JS, as any cycle in G is positive. Consequently, a feasible selection (sometimes called a "consistent" selection) is a complete selection S where $G(S)$ is acyclic (see Adams et al. [1] and Brucker and Knust [17]). Nevertheless, the more general Definition 1 is given here in view of the other, more complex job shop scheduling problems addressed in the sequel.

2.2.4. An Example

Consider the small JS example introduced in Figure 1.2. The corresponding disjunctive graph is shown in Figure 2.1. The conjunctive arcs are colored black, the

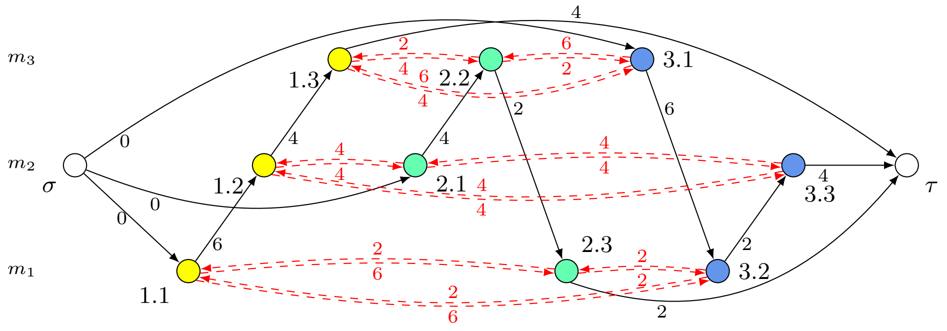


Figure 2.1.: Disjunctive graph G of the example.

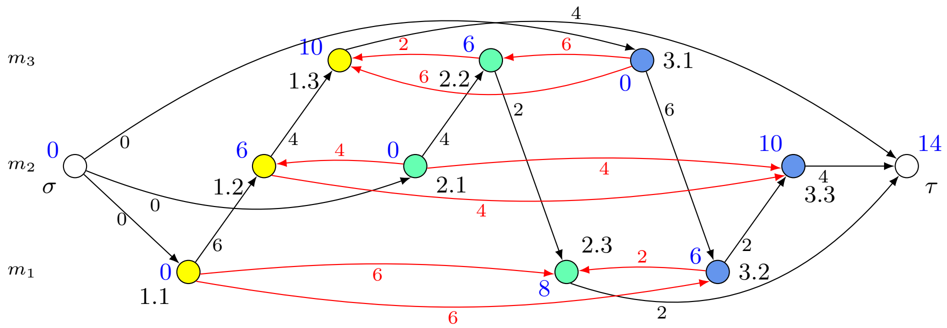


Figure 2.2.: Graph $G(S)$ of the feasible selection S corresponding to the schedule from Figure 1.2.

disjunctive arcs are drawn as dashed, red lines and the numbers depict the weights of the arcs.

The feasible selection that corresponds to the schedule from Figure 1.2 is shown in Figure 2.2. The selected disjunctive arcs are drawn as solid, red lines. The starting time of each operation $i \in I^+$, i.e. the length of a longest path from σ to i in $G(S)$, is depicted in blue.

This example contains $|\mathcal{E}| = 9$ disjunctive arc pairs, so there are $2^9 = 512$ possibilities for selecting exactly one arc from each pair. Only 64 of these selections are feasible. The makespans of all feasible selections are shown in the histogram of Figure 2.3. The best selection has makespan 14 and is the one displayed in Figure 2.2.

2.3. A Generalized Scheduling Model

In this section we present a generalized disjunctive scheduling model that can be used to formulate various scheduling problems, among them also the JS. It is based on the article “Feasible insertions in job shop scheduling, short cycles and stable sets” by Gröflin and Klinkert [42].

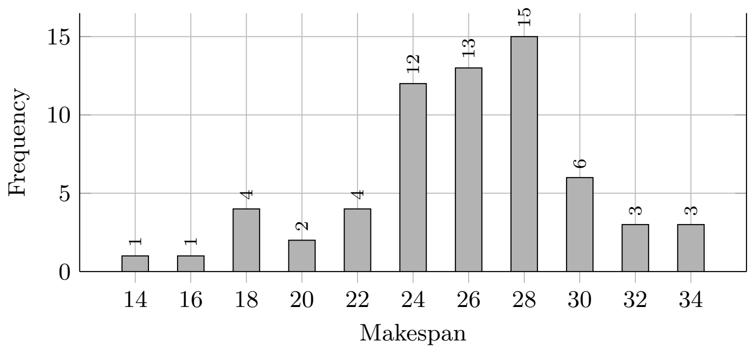


Figure 2.3.: Makespans of the 64 feasible selections.

2.3.1. A Disjunctive Programming Formulation

Let V be a finite set of events (e.g. starts of operations), σ, $\tau \in V$ a fictive start and end event, respectively, $A \cup E \subseteq V \times V, A \cap E = \emptyset$, two distinct sets of precedence constraints, and $d \in \mathbb{R}^{A \cup E}$ a weight function. A precedence constraint $(v, w) \in A \cup E$ with weight d_{vw} states that event v must occur at least d_{vw} time units before event w. Precedence constraints of set A and E are called *conjunctive* and *disjunctive* precedence constraints, respectively. In contrast to the conjunctive precedence constraints that always must hold, disjunctive constraints must hold if some other disjunctive constraints in E are violated. Hence, a family of disjunctive sets $\mathcal{E} \subseteq 2^E$ is defined together with E. Each disjunctive constraint is in at least one disjunctive set, i.e. $\bigcup_{D \in \mathcal{E}} D = E$. Then, the generalized disjunctive scheduling problem $\Pi = (V, A, E, \mathcal{E}, d)$ is the following problem:

$$\text{minimize } \alpha_\tau \tag{2.15}$$

$$\text{subject to:}$$

$$\alpha_w - \alpha_v \geq d_{vw} \text{ for all } (v, w) \in A, \tag{2.16}$$

$$\bigvee_{(v,w) \in D} \alpha_w - \alpha_v \geq d_{vw} \text{ for all } D \in \mathcal{E}, \tag{2.17}$$

$$\alpha_\sigma = 0. \tag{2.18}$$

Any feasible solution $\alpha \in \mathbb{R}^V$ of Π, also called a schedule, specifies times α_v for all events $v \in V$ so that all conjunctive precedence constraints expressed in (2.16) are satisfied and at least one disjunctive precedence constraint of each disjunctive set is satisfied, as expressed in (2.17). The makespan is minimized as expressed in (2.15).

2.3.2. A Disjunctive Graph Formulation

The scheduling problem Π is now formulated in a disjunctive graph $G = (V, A, E, \mathcal{E}, d)$, V denoting the node set, A the set of conjunctive arcs, E the set of disjunctive arcs, and $d \in \mathbb{R}^{A \cup E}$ the weights of the arcs. Each event is represented by a node and we identify a node with the event it represents. Each precedence constraint corresponds to an arc (v, w) with weight d_{vw}.

Denote by $\Omega^\Pi \subseteq \mathbb{R}^V$ the solution space of Π. Ω^Π can be described as follows using (complete, positive acyclic, positive cyclic) selections as given in Definition 1.

Let $\mathcal{S} \subseteq 2^E$ be the family of all feasible selections. Given a selection S, denote by $\Omega^\Pi(S) \subseteq \mathbb{R}^V$ the family of times $\alpha \in \mathbb{R}^V$ satisfying in $G(S) = (V, A \cup S, d)$:

$$\alpha_w - \alpha_v \geq d_{vw} \text{ for all arcs } (v, w) \in A \cup S, \tag{2.19}$$

$$\alpha_\sigma = 0. \tag{2.20}$$

Proposition 2 *The solution space Ω^Π of the scheduling problem Π is*

$$\Omega^\Pi = \bigcup_{S \in \mathcal{S}} \Omega^\Pi(S). \tag{2.21}$$

Proof. i) Consider any $S \in \mathcal{S}$ and $\alpha \in \Omega^\Pi(S)$. By (2.19) and (2.20), (2.16) and (2.18) obviously hold. As selection S is feasible, it is also complete and contains therefore at least one disjunctive constraint (v, w) for all disjunctive sets $D \in \mathcal{E}$. Hence by (2.19), (2.17) is satisfied, and $\alpha \in \Omega^\Pi$.

ii) Let $\alpha \in \Omega^\Pi$ and let $S \subseteq E$ be composed of all disjunctive constraints $(v, w) \in E$ that are satisfied by α, i.e. $\alpha_w - \alpha_v \geq d_{vw}$. Obviously, $\alpha \in \Omega^\Pi(S)$ for this selection S. We show that S is feasible, i.e. $S \in \mathcal{S}$. Indeed, S is complete as $\alpha \in \Omega^\Pi$ implies (2.17), i.e. α satisfies at least one disjunctive constraint (v, w) of each disjunctive set $D \in \mathcal{E}$. S is also positive acyclic since $\alpha \in \Omega^\Pi(S)$ is a *feasible potential function* in $G(S)$. By a well-known result of combinatorial optimization (see e.g. Cook et al. [26], p.25), $G(S)$ admits a feasible potential function – and hence a solution – if and only if no positive cycle exists in $G(S)$. ■

Given a feasible selection $S \in \mathcal{S}$, finding a schedule α minimizing the makespan is finding $\alpha \in \Omega(S)$ minimizing α_τ. As is well-known, this is easily done by longest path computations in $G(S) = (V, A \cup S, d)$ and letting α_i be the length of a longest path from σ to i for all $i \in V$.

The scheduling problem Π can therefore be formulated as follows:

Among all feasible selections, find a selection S minimizing the length of a longest path from σ to τ in $G(S) = (V, A \cup S, d)$.

Some remarks concerning the structure of the sets A, E and $\mathcal{E}$ are in order.

It is assumed that there is a path from node σ to each node $v \in V$ and from each node v to node τ in the conjunctive part (V, A, d) of G according to the meaning of σ and τ. Moreover, it is assumed that (V, A, d) is positive acyclic, otherwise no feasible solution exists.

As in [42], the disjunctive sets $D \in \mathcal{E}$ of the scheduling problems treated in this thesis satisfy:

$$|D| = 2 \text{ for all } D \in \mathcal{E}, \tag{2.22}$$

$$D \cap D' = \emptyset \text{ for any distinct } D, D' \in \mathcal{E}, \tag{2.23}$$

$$D \text{ is positive cyclic for all } D \in \mathcal{E}. \tag{2.24}$$

Thus, any disjunctive set $D \in \mathcal{E}$ is of the form $D = \{e, \overline{e}\}$. Arc $\overline{e}$ is said to be the mate of e and vice versa. Since $\bigcup_{D\in\mathcal{E}} D = E$, the disjunctive arc set E is partitioned into $|E|/2$ disjunctive pairs. Any complete selection S is composed of at least one arc of each pair, and if S is feasible, then by (2.24) it chooses exactly one arc from each pair.

In the disjunctive graph of the JS, the arcs of a disjunctive pair $e, \overline{e}$ have common end nodes, i.e. they are of the form $(v, w), (w, v)$. In the disjunctive graph of the generalized scheduling problem, a disjunctive arc pair need not be of the form $(v, w), (w, v)$ and may have distinct end nodes.

2.4. A Complex Job Shop Model (CJS)

In the generalized scheduling model, no mention of jobs or machines is made. We adapt this model by specializing it to capture essential features of the job shop such as machines and job structure and by extending it to include routing flexibility. The model includes (to some extent) the features mentioned in Section 1.4 and will be called the Complex Job Shop (CJS) model.

2.4.1. Building Blocks of the CJS Model and a Problem Statement

In the JS, a job is a sequence of operations on machines. After the processing of an operation, the job may be stored somewhere "out of the system" and "comes back" into the system for its next operation. Storage operations and buffers for storing the jobs are not modeled explicitly.

Here we will consider all "machines" used by the jobs from their start to their completion. These machines are not restricted to processors executing some machining, but can also be, for example, buffers for storage operations and mobile devices for transport operations. As in the JS, we assume here that each machine can handle at most one job at any time, and each operation needs one machine for its execution.

In the CJS model, a job can therefore be described as follows. Once started and until its completion, a job is *on a machine*. After completing an operation, a job might wait on the machine – thus blocking it – until it is transferred to its next machine. While transferring a job, both involved machines are occupied simultaneously.

An operation can be described by the following four steps: i) a take-over step, where the job is taken over from the previous machine, ii) a processing step (e.g. machining, transport or storage), iii) a possible waiting time on the machine, and iv) a hand-over step where the job is handed over to the next machine. The take-over step of an operation must occur simultaneously with the hand-over step of its predecessor operation, i.e. the starting time of these steps as well as their duration are the same. Take-over and hand-over steps will be sometimes referred to as transfer steps.

The duration of the processing step and transfer steps are given. The waiting time in step iii) is unknown, but can be limited by specifying a maximum sojourn time allowed on the machine. These times are also called maximum time lags.

We allow for routing flexibility. Each operation needs a machine for its execution. However, this machine is not fixed, but can be chosen from a subset of alternative machines.

We also allow for sequence-dependent setup times between two consecutive operations on a machine, and an initial setup time and a final setup time might be present for each transfer step. The initial setup times, also called release times, specify earliest starting times of the transfer steps and the final setup times, also called tails, define minimum times elapsing between the end of the transfer steps and the overall finish time (makespan).

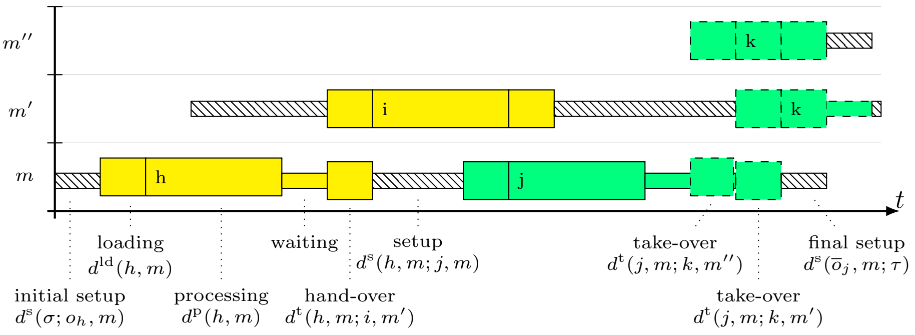
As in the JS, any pair of operations using the same machine must be sequenced, i.e. they cannot be executed simultaneously as a machine can handle at most one job at any time. In addition, we also allow to sequence pairs of transfer steps. The involved machines may not be the same. Such sequencing decisions are for instance needed to model collision avoidance of robots (cf. Chapter 7).

Informally, the CJS problem can be stated as follows. A schedule consists of an assignment of a machine and a starting time of the hand-over, processing and take-over step for each operation so that all constraints described above are satisfied. The objective is to find a schedule with minimal makespan.

2.4.2. Notation and Data

As in the JS, M denotes the set of machines, I the set of operations and $\mathcal{J}$ the set of jobs. For each operation $i \in I$, the following notation and data is given.

- i needs a machine for its execution, which can be chosen from a (possibly operation-dependent) subset of alternative machines $M_i \subseteq M$.
- The durations of the take-over, processing and hand-over steps of i are the following. Let h be the job predecessor operation of i and j its job successor operation. For any $m \in M_i, p \in M_h, q \in M_j$, the duration of the take-over, processing and hand-over step of i is $d^{\mathrm{t}}(h, p; i, m)$, $d^{\mathrm{p}}(i, m)$ and $d^{\mathrm{t}}(i, m; j, q)$, respectively. If i is the first operation of job J, its take-over step is called more appropriately a loading step of duration $d^{\mathrm{ld}}(i, m)$, and similarly, if i is the last operation of job J, its hand-over step is called an unloading step of duration $d^{\mathrm{ul}}(i, m)$.
- The maximum sojourn time of i on $m \in M_i$ is $d^{\mathrm{lg}}(i, m)$. (lg stands for time lag.)
- Let o_i be the take-over step and $\bar{o}_i$ the hand-over step of i. Denote by $O = \{o_j, \bar{o}_j : j \in I\}$ the set of all transfer steps.
- An initial setup time $d^{\mathrm{s}}(\sigma; o, m)$ and a final setup time $d^{\mathrm{s}}(o, m; \tau)$ is given for each transfer step $o \in \{o_i, \bar{o}_i\}$ of i.



m''
m'
m
k
i
k
h
j
t
loading
$d^{\mathrm{ld}}(h,m)$
waiting
setup
$d^{\mathrm{s}}(h,m;j,m)$
take-over
$d^{\mathrm{t}}(j,m;k,m'')$
final setup
$d^{\mathrm{s}}(\overline{o}_j,m;\tau)$
initial setup
$d^{\mathrm{s}}(\sigma;o_h,m)$
processing
$d^{\mathrm{p}}(h,m)$
hand-over
$d^{\mathrm{t}}(h,m;i,m')$
take-over
$d^{\mathrm{t}}(j,m;k,m')$

Figure 2.4.: An illustration of the job structure and our notation.

For any two distinct operations $i,j \in I$ and any common machine $m \in M_i \cap M_j$, if j immediately follows i on m, a setup of duration $d^{\mathrm{s}}(i,m;j,m)$ occurs on m between the hand-over step $\overline{o}_i$ of i and the take-over step o_j of j.

Additionally, let $\mathcal{V}$ be the set of pairs of transfer steps that must be sequenced. An element of $\mathcal{V}$ has the form $\{(o,m),(o',m')\}, o,o' \in O, m,m' \in M$, and setup times $d^{\mathrm{s}}(o,m;o',m')$ and $d^{\mathrm{s}}(o',m';o,m)$ are given with the following meaning. If transfer step o is executed on machine m and o' on m' then they cannot occur simultaneously, and a minimum time must elapse between them, i.e. o has to end at least $d^{\mathrm{s}}(o,m;o',m')$ before o' starts, or o' has to end at least $d^{\mathrm{s}}(o',m';o,m)$ before o starts. We sometimes call $\mathcal{V}$ the set of *conflicting transfer steps.*

In this thesis, set $\mathcal{V}$ will be used to model collision avoidance in a job shop setting with multiple mobile devices that move on a common rail (see Chapter 7). If not otherwise stated, we assume $\mathcal{V} = \emptyset$.

Figure 2.4 illustrates the described structure of the jobs and our notation in a Gantt chart. Four operations h,i,j and k are depicted. Operations h and i are consecutive in one job, and j and k are consecutive in another job. The operations can be executed on the following machines: $M_h = M_j = \{m\}, M_i = \{m'\}, M_k = \{m,m'\}$. Operation k as well as all involved transfer steps are depicted by dashed lines to indicate the open choice of the machine for k.

Some standard assumptions on the durations are made. All durations are non-negative. For each operation $i \in I$ and machine $m \in M_i$, $d^{\mathrm{lg}}(i,m) \geq d^{\mathrm{p}}(i,m)$. The difference $d^{\mathrm{lg}}(i,m) - d^{\mathrm{p}}(i,m)$ is the maximum time the job can stay on machine m after completion of operation i.

Setup times satisfy the following so-called weak triangle inequality (cf. Brucker and Knust [17], p. 11-12). For any operations i,j,k on a common machine m, $d^{\mathrm{s}}(i,m;j,m) + d^{\mathrm{p}}(j,m) + d^{\mathrm{s}}(j,m;k,m) \geq d^{\mathrm{s}}(i,m;k,m)$. The triangle inequality ensures that setup times between non-consecutive operations on a machine do not become active in the disjunctive graph formulation.

It is possible that two operations i,j that are consecutive in a job are on a same machine $m \in M_i \cap M_j$. In this case, both transfer time $d^{\mathrm{t}}(i,m;j,m)$ and setup time

$d^{\mathrm{s}}(i,m;j,m)$ are usually set to zero, essentially combining both operations into a single operation.

For any two operations i,j from distinct jobs, $i \in J, j \in J', J \neq J'$, with some common machine $m \in M_i \cap M_j$, the duration of the operations i and j on m must be positive, where the duration of an operation is the sum of its take-over, processing and hand-over durations. And similarly, for any conflicting transfer steps $\{(o,m),(o',m')\} \in \mathcal{V}$, the durations of the steps o on m and o' on m' must be positive. These assumptions state that if a pair of operations or a pair of transfer steps must be sequenced, then they must have a positive duration.

2.4.3. A Disjunctive Graph Formulation

We now formulate the CJS in a disjunctive graph. Main features of the disjunctive graph are the following, as illustrated in Figure 2.5.

For each operation and each alternative machine, four nodes are introduced representing the start and end of the take-over step and the start and end of the hand-over step. Note that the node representing the end of the take-over step also represents the start of the processing step.

The start and end of both transfer steps are linked by an arc, called take-over arc and hand-over arc, respectively. The end of the take-over and the start of the hand-over are joined by two arcs: a processing arc and a time lag arc. Two consecutive operations in a job are joined by a pair of transfer arcs and arcs synchronizing the hand-over of the previous operation with the take-over of the next operation.

Any two operations of the same job on a common machine are linked by a setup arc, and similarly, any two conflicting transfer steps belonging to the same jobs are linked by a setup arc.

Finally, a pair of disjunctive arcs is introduced between any pair of operations from distinct jobs on a same machine linking the end of the hand-over of one operation with the start of the take-over of the other operation. Additionally, for each pair of conflicting transfer steps belonging to distinct jobs, a pair of disjunctive arcs is introduced, linking the end of one transfer step with the start of the other transfer step.

Specifically, the disjunctive graph $G = (V, A, E, \mathcal{E}, d)$ of the CJS is constructed as follows. To each operation $i \in I$ and machine $i \in M_i$ a set of four nodes $V_{im} = \{v^1_{im}, v^2_{im}, v^3_{im}, v^4_{im}\}$ is associated . Node set V of G consists of the union of the V_{im}'s, together with two additional nodes σ and τ representing fictive start and end operations of duration 0 occurring before and after all other operations, respectively, so $V = \cup\{V_{im} : i \in I, m \in M_i; \{\sigma, \tau\}\}$.

The set of conjunctive arcs A comprises the following arcs:

1. For each operation $i \in I$ and machine $m \in M_i$, four arcs (v^1_{im}, v^2_{im}), (v^2_{im}, v^3_{im}), (v^3_{im}, v^4_{im}) and (v^3_{im}, v^2_{im}) with respective weights: $d^{\mathrm{ld}}(i,m)$ if $i \in I^{\mathrm{first}}$ and 0 otherwise; $d^{\mathrm{p}}(i,m)$; $d^{\mathrm{ul}}(i,m)$ if $i \in I^{\mathrm{last}}$ and 0 otherwise; $-d^{\mathrm{lg}}(i,m)$. The four arcs are referred to as take-over, processing, hand-over and time lag arc.

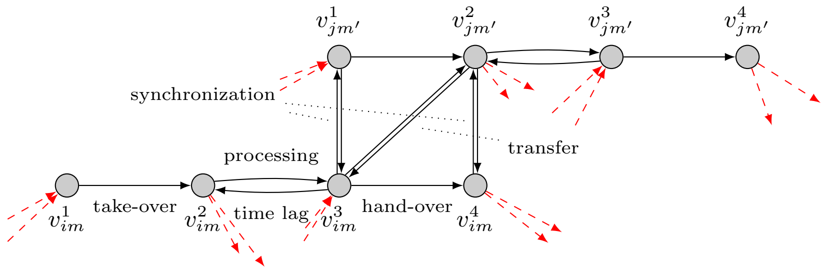


Figure 2.5.: Two consecutive operations i, j in a job on machines $m \in M_i, m' \in M_j$. The dotted red arcs illustrate potential disjunctive arcs.

2. For each operation $i \in I$ and machine $m \in M_i$, two initial setup arcs (σ, v^1_{im}) and (σ, v^3_{im}) of respective weights $d^{\rm s}(\sigma; o_i, m)$ and $d^{\rm s}(\sigma; \bar{o}_i, m)$, and similarly, two final setup arcs (v^2_{im}, τ) and (v^4_{im}, τ) of respective weights $d^{\rm s}(o_i, m; \tau)$ and $d^{\rm s}(\bar{o}_i, m; \tau)$.
3. For any two consecutive operations i and j of a job J and machines $m \in M_i$, $m' \in M_j$, two pairs of synchronization arcs $(v^3_{im}, v^1_{jm'})$, $(v^1_{jm'}, v^3_{im})$ and $(v^4_{im}, v^2_{jm'})$, $(v^2_{jm'}, v^4_{im})$ of weight 0 joining the starts and ends of the hand-over step of operation i and the take-over step of operation j, and a pair of transfer arcs $(v^3_{im}, v^2_{jm'})$, $(v^2_{jm'}, v^3_{im})$ of weight $d^{\rm t}(i, m; j, m')$ and $-d^{\rm t}(i, m; j, m')$ joining the start of the hand-over of operation i with the end of the take-over of j.
4. For any two operations $i = J_r$ and $j = J_s$ of a job J with $1 \leq r+1 < s \leq |J|$ and common machine $m \in M_i \cap M_j$, a setup arc (v^4_{im}, v^1_{jm}) of weight $d^{\rm s}(i, m; j, m)$. And similarly, for any conflicting transfer steps $\{(o, m), (o', m')\} \in \mathcal{V}$, if o precedes o' in the same job, a setup arc (v, w), of weight $d^{\rm s}(o, m; o', m')$, where v and w are the nodes representing the end of o on m and the start of o' on m'.

The set of disjunctive arcs E consist of the following arcs.

- For any two operations $i, j \in I$ of distinct jobs and any common machine $m \in M_i \cap M_j$, two disjunctive arcs $(v^4_{im}, v^1_{jm}), (v^4_{jm}, v^1_{im})$ with respective weights $d^{\rm s}(i, m; j, m), d^{\rm s}(j, m; i, m)$.
- For each pair of conflicting transfer steps $\{(o, m), (o', m')\} \in \mathcal{V}$, if o and o' belong to distinct jobs, two disjunctive arcs $(v', w), (w', v)$ of respective weights $d^{\rm s}(o, m; o', m')$, $d^{\rm s}(o', m'; o, m)$, where v and v' are the nodes representing the start and end of the transfer step o on m, and w and w' are the nodes representing the start and end of o' on m'.

The family $\mathcal{E}$ consists of all introduced pairs of disjunctive arcs.

As the generalized disjunctive scheduling model considered in Section 2.3, the CJS problem can now be formulated as a combinatorial optimization problem in the disjunctive graph G.

To capture the routing decisions, we introduce modes.

Definition 3 *A mode is a tuple $\mu = (\mu(i) : i \in I)$ assigning to each operation $i \in I$ a machine $\mu(i) \in M_i$, and let $\mathcal{M}$ be the set of all modes.*

Job	*Op. 1*	*Op. 2*	*Op. 3*
Job 1	6, $\{m_1, m_4\}$	4, $\{m_2\}$	4, $\{m_3, m_5\}$
Job 2	4, $\{m_2\}$	2, $\{m_3, m_5\}$	2, $\{m_1, m_4\}$
Job 3	6, $\{m_3, m_5\}$	2, $\{m_1, m_4\}$	4, $\{m_2\}$

Table 2.1.: Processing duration and alternative machines for all operations in the Example.

A mode μ selects a node-induced subgraph G^μ in G defined as follows. Let $V^\mu = \cup\{V_{i,\mu(i)} : i \in I; \{\sigma, \tau\}\}$, $A^\mu = A \cap \gamma(V^\mu), E^\mu = E \cap \gamma(V^\mu)$ and $\mathcal{E}^\mu = \{\{e, \overline{e}\} \in E : e, \overline{e} \in E^\mu\}$. The resulting graph $G^\mu = (V^\mu, A^\mu, E^\mu, \mathcal{E}^\mu, d)$ is the disjunctive graph *associated to the mode* μ. For simplicity, the restriction of the weight vector d to $A^\mu \cup E^\mu$ is denoted again by d.

Definition 4 *For any mode $\mu \in \mathcal{M}$ and any set of disjunctive arcs $S \subseteq E^\mu$, (μ, S) is called a* selection *in G. Selection (μ, S) is* complete *if $S \cap D \neq \emptyset$ for all $D \in \mathcal{E}^\mu$. Selection (μ, S) is* positive acyclic *if subgraph $G(\mu, S) = (V^\mu, A^\mu \cup S, d)$ contains no positive cycle, and is* positive cyclic *otherwise. Selection (μ, S) is* feasible *if it is positive acyclic and complete.*

The CJS can now be formulated as the following problem:

Among all feasible selections, find a selection (μ, S) minimizing the length of a longest path from σ to τ in subgraph $G(\mu, S) = (V^\mu, A^\mu \cup S, d)$.

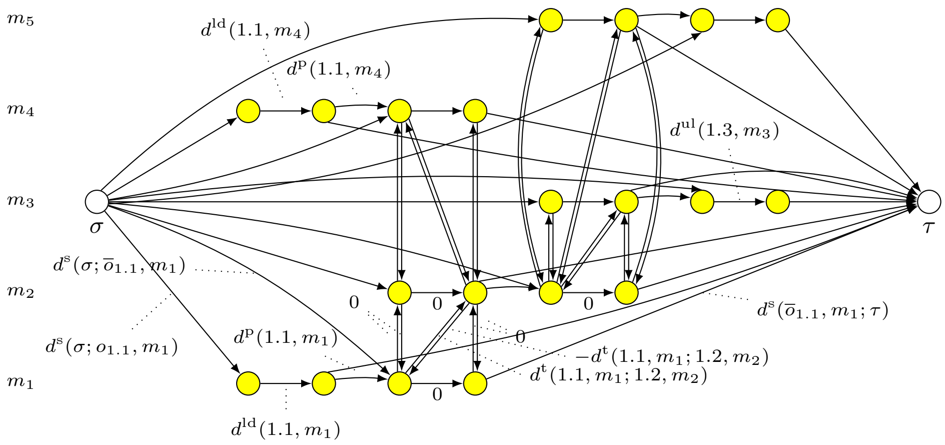
2.4.4. An Example

We illustrate the CJS model in the example introduced in Figure 1.6, Chapter 1, with the routing flexibility given in Section 1.4. For ease of reading, we recall here its data and features. This example will be used throughout Part I and be called *the Example.*

The Example consists of three jobs 1, 2, 3, five machines $M = \{m_1, m_2, m_3, m_4, m_5\}$. Each job has three operations, the alternative machines and the processing duration being given in Table 2.1. For instance, operation 2.3 (see Job 2, Op. 3) has a processing duration of 2 and can be executed on machines m_1 and m_4. The processing duration is assumed to be independent of the chosen machine.

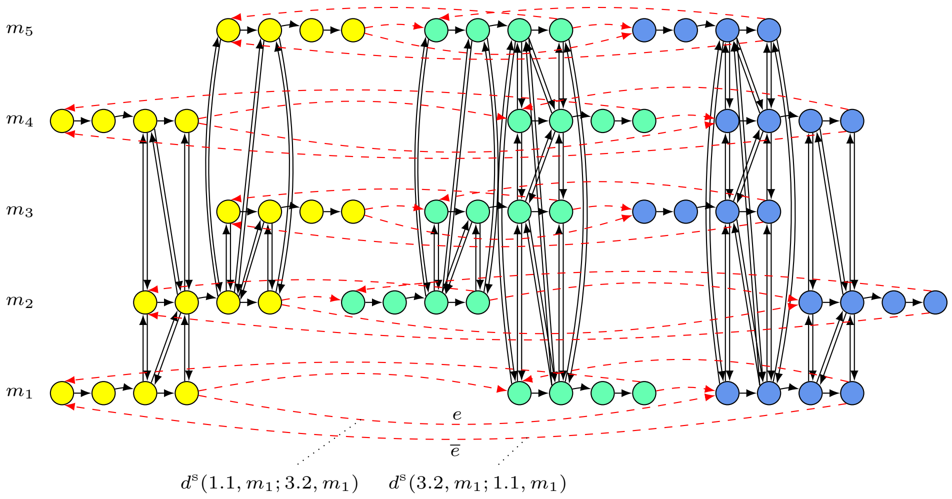
All setup times $d^{\text{s}}(i, m; j, m)$, $d^{\text{s}}(\sigma; o, m)$ and $d^{\text{s}}(o, m; \tau)$ are 0. All transfer times $d^{\text{t}}(i, m; j, m')$, loading times $d^{\text{ld}}(i, m)$ and unloading times $d^{\text{ul}}(i, m)$ are 1. There are no maximum time lags present, so $d^{\text{lg}}(i, m) = \infty$, and the time lag arcs are omitted in disjunctive graph G. We remark that the Example is an instance of the flexible blocking job shop, which will be discussed in Chapter 5 in detail.

In Figure 2.6, job 1 (the yellow job) of the Example is depicted, and some selected arc weights are indicated. Figure 2.7 depicts all jobs of the Example in the disjunctive graph using yellow, green and blue nodes for job 1, 2 and 3, respectively. Start node σ and end node τ are omitted for clarity. The arc weight of one disjunctive arc pair denoted by $e, \overline{e}$ is also indicated.



m_5
m_4
m_3
m_2
m_1
$d^{\mathrm{ld}}(1.1, m_4)$
$d^{\mathrm{p}}(1.1, m_4)$
$d^{\mathrm{ul}}(1.3, m_3)$
σ
τ
$d^{\mathrm{s}}(\sigma; \overline{o}_{1.1}, m_1)$
$d^{\mathrm{s}}(\sigma; o_{1.1}, m_1)$
$d^{\mathrm{s}}(\overline{o}_{1.1}, m_1; \tau)$
0
0
0
0
0
$-d^{\mathrm{t}}(1.1, m_1; 1.2, m_2)$
$d^{\mathrm{t}}(1.1, m_1; 1.2, m_2)$
$d^{\mathrm{p}}(1.1, m_1)$
$d^{\mathrm{ld}}(1.1, m_1)$

Figure 2.6.: Job 1 of the Example.



m_5
m_4
m_3
m_2
m_1
e
$\overline{e}$
$d^{\mathrm{s}}(1.1, m_1; 3.2, m_1)$
$d^{\mathrm{s}}(3.2, m_1; 1.1, m_1)$

Figure 2.7.: All three jobs of the Example.

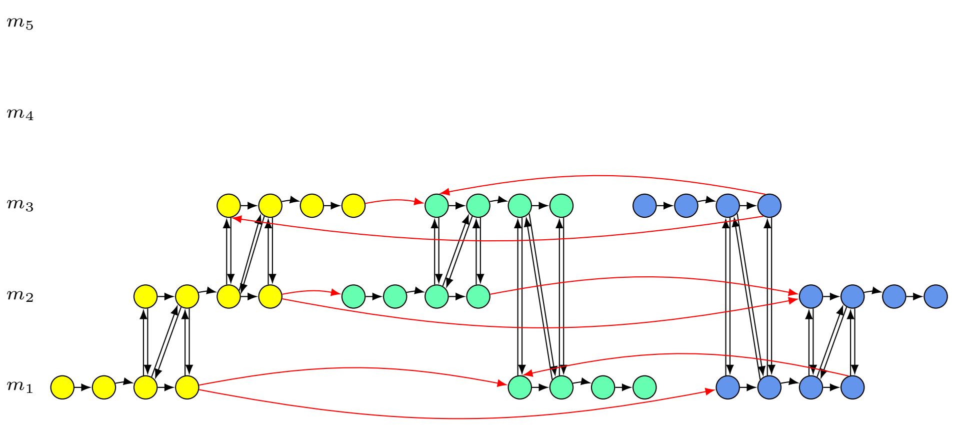


Figure 2.8.: The selection that corresponds to the solution of Figure 1.6.

The selection that corresponds to the schedule depicted in Figure 1.6 is shown in Figure 2.8. Note that no operations are assigned to machines m_4 and m_5 in this schedule.

2.4.5. Modeling Features in the CJS Model

We briefly discuss how the CJS model captures the features mentioned in Section 1.4.

Routing flexibility, sequence-dependent setup times, transfer times and maximum time lags are modeled explicitly.

Release times can be modeled using the initial setup times. The final setup times (so-called tails) can be used to incorporate due dates. Specifically, instead of minimizing the makespan, we may want to minimize the maximum lateness of all jobs. To achieve this objective, the final setup times are set as follows. For each job $J \in \mathcal{J}$, let $d^{\mathrm{d}}(J)$ be the due date of J. For each operation $i \in J$ of job J and machine $m \in M_i$, the final setup times $d^{\mathrm{s}}(o_i, m; \tau)$ and $d^{\mathrm{s}}(\bar{o}_i, m; \tau)$ are set to $-d^{\mathrm{d}}(J)$ (see e.g. Sourd and Nuijten [106]).

The absence of buffers can easily be integrated into the model (see the Example). No storage operations are needed in this case.

If an unlimited number of buffers is available, as in the JS, we may introduce a buffer b_J for each job J. While not processed, a job J is stored in its buffer b_J. The disjunctive graph of this case is illustrated in Figure 2.9 showing two consecutive operations i, j of some job J and a storage operation i' executed between i and j on buffer b_J. Minimum and maximum storage times can be integrated by setting the processing time $d^{\mathrm{p}}(i', b_J)$ and the time lag $d^{\mathrm{lg}}(i', b_J)$ accordingly.

To some extent, we can model a limited number of buffers in the same fashion. Suppose, for example, that machine m has two buffers b, b' and they are used after completing an operation on m, and can handle at most one job at any time. If the buffers are used sequentially, i.e. the job has to go sequentially through them, then

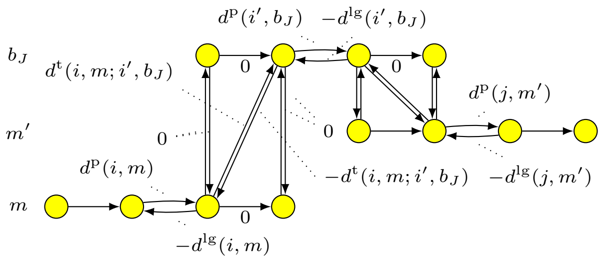

$d^{\mathrm{p}}(i', b_J)$
$-d^{\mathrm{lg}}(i', b_J)$
b_J
$d^{\mathrm{t}}(i, m; i', b_J)$
m'
m
0
$d^{\mathrm{p}}(j, m')$
$d^{\mathrm{p}}(i, m)$
$-d^{\mathrm{t}}(i, m; i', b_J)$
$-d^{\mathrm{lg}}(j, m')$
$-d^{\mathrm{lg}}(i, m)$

Figure 2.9.: Two consecutive machining operations i, j of some job and buffer operation i' executed between i and j.

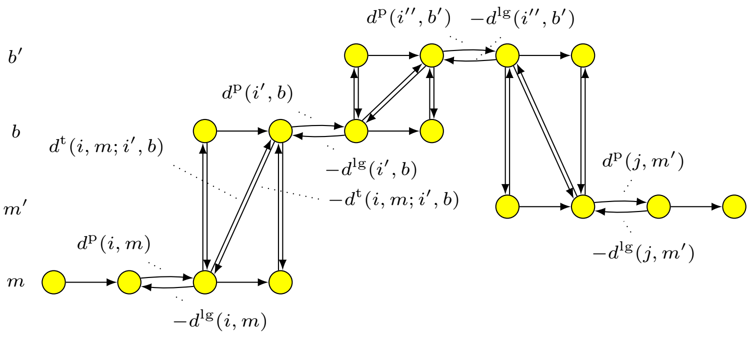

$d^{\mathrm{p}}(i'', b')$
$-d^{\mathrm{lg}}(i'', b')$
b'
$d^{\mathrm{p}}(i', b)$
b
$d^{\mathrm{t}}(i, m; i', b)$
$-d^{\mathrm{lg}}(i', b)$
$-d^{\mathrm{t}}(i, m; i', b)$
$d^{\mathrm{p}}(j, m')$
m'
$d^{\mathrm{p}}(i, m)$
$-d^{\mathrm{lg}}(j, m')$
m
$-d^{\mathrm{lg}}(i, m)$

Figure 2.10.: Two consecutive machining operations i, j of some job and buffer operations i' on b and i' on b'' executed between i and j.

two storage operations are introduced after the operation on m. Figure 2.10 illustrates this case. If the buffers are installed in a parallel fashion, i.e. a job visits exactly one of the two buffers and we can choose which one, then one storage operation is introduced with both buffers as alternative machines. Figure 2.11 illustrates this case.

Transport operations can be integrated in a simple manner by specifying the mobile devices and transport operations accordingly. We refer to the Chapters 6 and 7 where job shop problems with transportation are discussed further.

A no-wait condition between two consecutive operations i and j in some job is integrated by setting $d^{\mathrm{lg}}(i, m) = d^{\mathrm{p}}(i, m)$. Then the hand-over step of operation i starts exactly at the end of its processing step. Moreover, due to the transfer arcs, the job is transferred directly to its next machine, and the no-wait condition is satisfied. This case is illustrated in Figure 2.12.

A final remark is in order. Obviously, depending on the presence or absence of the various features in an instance, a more compact disjunctive graph formulation can be obtained. We point to Chapter 4, where a compact form is provided for the flexible job shop with setup times.

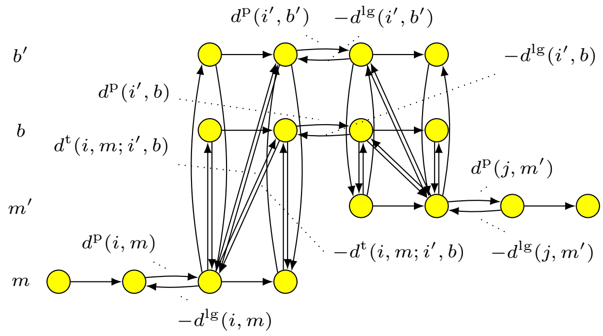

$d^{\mathrm{p}}(i', b')$
$-d^{\mathrm{lg}}(i', b')$
b'
$-d^{\mathrm{lg}}(i', b)$
$d^{\mathrm{p}}(i', b)$
b
$d^{\mathrm{t}}(i, m; i', b)$
$d^{\mathrm{p}}(j, m')$
m'
$d^{\mathrm{p}}(i, m)$
$-d^{\mathrm{t}}(i, m; i', b)$
$-d^{\mathrm{lg}}(j, m')$
m
$-d^{\mathrm{lg}}(i, m)$

Figure 2.11.: Two consecutive machining operations i, j of some job and buffer operation i' executed on b or b' between i and j.

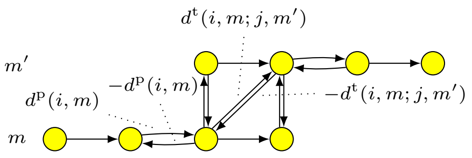


Figure 2.12.: Two consecutive operations i, j with a no-wait condition.

CHAPTER 3 A SOLUTION APPROACH

3.1. Introduction

Chapter 2 introduced the so-called CJS model that includes a broad variety of job shop scheduling problems such as the classical job shop, the blocking job shop, the no-wait job shop and versions of them with setup times, transfer times, time lags and routing flexibility.

CJS problems are clearly difficult problems. Finding a minimum makespan solution is NP-hard in the classical job shop (Lenstra and Rinnooy Kan [65]) as well as in the blocking and no-wait job shop (Hall and Sriskandarajah [47]). Moreover, besides this complexity class membership, the classical job shop has also earned the reputation of being one of the most computationally stubborn combinatorial problem considered to date (Applegate and Cook in [4]). In the current state of the art, exact solution approaches for job shop scheduling are capable of solving smaller problem instances, but fail rapidly when problem size increases to numbers found in practice. It is also notable that the impressive advances made in solving integer linear programs over the last decade do not seem to have benefited proportionally to the solution of scheduling problems.

For these reasons, most methods for job shop scheduling are of a heuristic nature and will likely remain so in the near future, given the current state of the art. This statement applies to the classical job shop: we mention here only the well-known shifting bottleneck procedure of Adams et al. [1] and the tabu search algorithm of Nowicki and Smutnicki [84], the adapted shifting bottleneck approach of Balas et al. [9] in the presence of setups, and the tabu search algorithm of Mastrolilli and Gambardella [79] in the presence of routing flexibility. This statement applies all the more so to CJS problems, such as the blocking job shop, which are inherently more difficult than the classical job shop.

In this chapter, we describe a heuristic solution method for a class of CJS problems based on the disjunctive graph formulation given in Chapter 2. The approach is a local search based on job insertion, which we call the Job Insertion Based Local Search (JIBLS), and which has been introduced by Klinkert [62] and Gröflin and Klinkert [42] for the blocking job shop and extended to the flexible blocking job shop with routing flexibility by Pham [92] and Gröflin, Pham and Bürgy [45], and to the blocking job shop with rail-bound transportation by Bürgy and Gröflin [21]. The JIBLS described here in the more general context of the CJS model unifies its application in the cases mentioned and extends somewhat its use. Its theoretical foundation relies on the "insertion theory" developed in [42].

This chapter is structured as follows. In the next section, the applied local search principle, i.e. job insertion, is introduced in the Example and formalized. Then, some structural properties of job insertion are developed in Section 3.3. These properties are used in Section 3.4 to generate feasible neighbor solutions yielding a neighborhood that is used in a tabu search in Section 3.5.

3.2. The Local Search Principle

Local search methods are based on the exploration of a set of solutions by repeatedly moving from a (current) solution to another solution in the current solution's neighborhood. The aim is to reach an optimal or at least a good solution.

The moves are a key component of the local search. In the JIBLS, these moves are based on job insertion. We illustrate the principle first in the Example and formalize it then via a disjunctive graph.

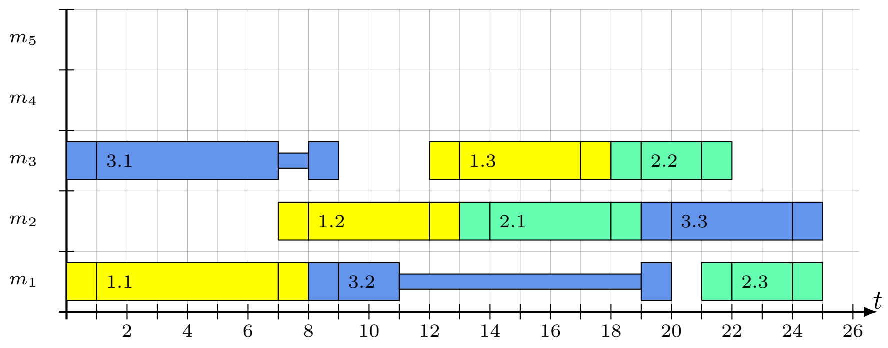
3.2.1. The Local Search Principle in the Example

Consider the Example and let the schedule depicted in Figure 3.1 be the current solution. Its corresponding selection (μ, S) is depicted in graph $G(\mu, S)$ in the lower part of the figure.

The goal is to generate a neighbor of this solution by applying "local" changes, i.e. a small part of the current solution is altered while the other part is kept. In job shop scheduling, these changes are often defined on the level of sequencing decisions and assignments of machines to operations. We apply this idea here and generate a neighbor by choosing some operation and either moving it in the sequence of operations on the machine it was assigned to or assigning it to an alternative machine and inserting it on that machine.

Specifically, some operation $i \in I$ is chosen. Either a selected disjunctive arc $e \in S$ incident to (the nodes of) i is replaced by its mate $\overline{e}$, or operation i is assigned to another machine $m \in M_i - \mu(i)$ and is inserted on m by enforcing corresponding disjunctive arcs incident to i.

Illustrating this in the Example, choose operation 2.2. Operation 2.2 can be assigned to machine m_5 and inserted on m_5. (This insertion is trivial here, as no other operation is on that machine.) Another possibility is to move operation 2.2 before op-



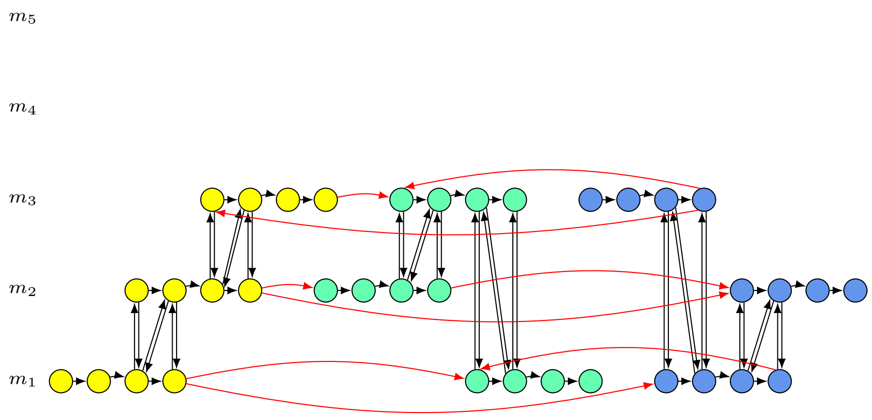


Figure 3.1.: The current schedule in the Example.

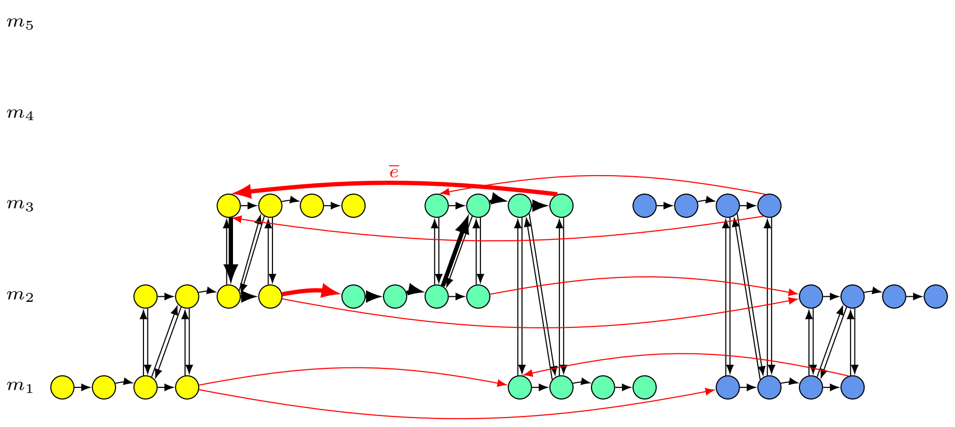


Figure 3.2.: An infeasible neighbor selection $(\mu, S - e \cup \overline{e})$.

eration 1.3 by replacing arc $e = (v^4_{1.3,m_3}, v^1_{2.2,m_3})$ with its mate $\overline{e} = (v^4_{2.2,m_3}, v^1_{1.3,m_3})$, resulting in the neighbor selection $(\mu, S - e \cup \overline{e})$ depicted in Figure 3.2.

Consider the selection $(\mu, S - e \cup \overline{e})$ just obtained. At a certain point in time, job 1 finishes on machine m_2 the processing of operation 1.2 and waits on m_2, thus blocking it, until it can be transferred to m_3 for the processing of 1.3. Operation 2.2 must be executed before 1.3 on m_3. However, job 2 is waiting for machine m_2 in order to execute operation 2.1, a deadlock situation. Indeed, selection $(\mu, S - e \cup \overline{e})$ is not feasible as $G(\mu, S - e \cup \overline{e})$ contains the positive cycle highlighted in Figure 3.2 by the thick arcs. Note that similar feasibility issues arise when assigning an operation to another machine. In fact, it may not even be possible to insert the operation in the sequence of operations on the new machine in a feasible way if no other changes are allowed.

While infeasible solutions are accepted in some local search methods, they are generally avoided in complex job shop scheduling as recovering feasibility while maintaining solution quality is difficult. For this reason, we aim at consistently generating feasible solutions using more complex moves. The main question here is which “changes” to allow in a move and which assignment and sequencing decisions to keep fixed. In job shop scheduling, this choice is mainly driven by the machine and job structure.

One approach is based on resequencing the operations on one machine while keeping the operation sequences on the other machines. This approach is for instance applied in the shifting bottleneck procedure in the JS (Adams et al. [1]) and in the Job Shop with Setup Times (JSS) (Balas et al. [9]). However, preserving feasibility seems to be difficult in more complex job shops (see e.g. Zhang et al. [115] and Khosravi et al. [57]).

Another approach is based on allowing the operations of one job to be moved and keeping the operation sequences of all other jobs. This approach is known as job insertion and has been applied as a mechanism for devising heuristics in several scheduling problems. It was used, for instance, in the JS by Werner and Winkler [113] and Kis [59] (see also Kis and Hertz [61]), and in more complex job shop scheduling

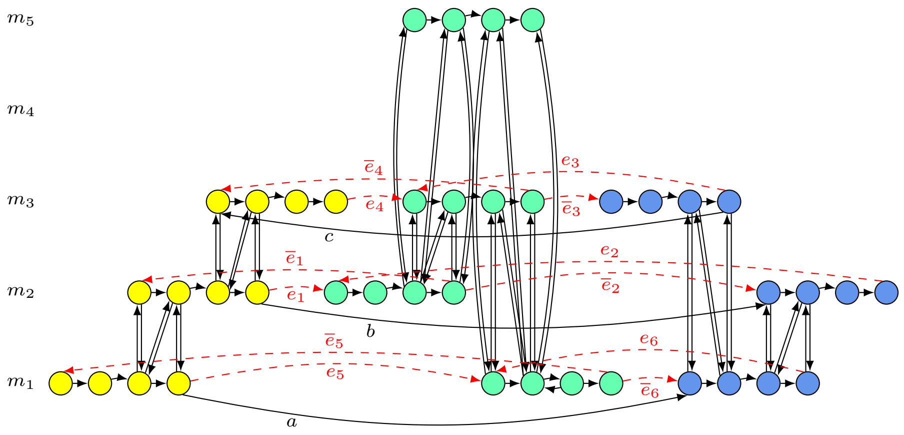


Figure 3.3.: The insertion graph of job 2 with local flexibility at operation 2.2.

problems by van den Broek [111] and Gröflin et al. (cf. Section 3.1) where it proved to be a valuable approach. Thus, it may also be valuable in the CJS model.

Consider again the Example with the selected operation 2.2. We allow operation 2.2 to be assigned to one of its alternative machines and the operations of job 2 to be moved in the operation sequences on their corresponding machines, and keep all other assignment and sequencing decisions fixed. In the disjunctive graph, these decisions are reflected as illustrated in Figure 3.3. Starting from the disjunctive graph G of the Example, we delete all nodes representing other modes, except for operation 2.2, and delete all arcs incident to deleted nodes. As all other jobs are kept fixed, we add all arcs between the other jobs to the conjunctive arc set. In the Example, three arcs are kept fixed, called a, b and c in the figure, and the set of disjunctive arcs consists of six arc pairs named $e_r, \overline{e}_r, r = 1, \dots, 6$. The obtained graph is called the job insertion graph of job 2 with local flexibility at operation 2.2. This graph will serve as the framework in which moves are defined.

Note that the current selection, more appropriately called here the current insertion of job 2, consists of the arc set $\{e_1, \overline{e}_2, e_3, e_4, e_5, e_6\}$.

3.2.2. The Job Insertion Graph with Local Flexibility

We now formalize the job insertion concept described above.

Given is a feasible selection (μ, S) in the disjunctive graph $G = (V, A, E, \mathcal{E}, d)$ of a CJS problem. Select an operation $i \in I$ and let J be the job to which $i \in J$ belongs. Consider the problem of extracting and reinserting job J, allowing operation i to be assigned to any machine $m \in M_i$ while preserving machine assignment $\mu(j)$ for all other operations $j \in I - i$. The disjunctive graph $G_i = (V_i, A_i, E_i, \mathcal{E}_i, d)$ for this problem is obtained as follows.

As the mode of all operations $j \in I - i$ is fixed at $\mu(j)$, delete from V node sets V_{jm} for all $j \in I - i$ and all machines $m \in M_j - \mu(j)$, obtaining V_i. As the sequencing of all other jobs is fixed, add to the set of conjunctive arcs A all arcs of selection (μ, S) that are not incident to job J, obtaining A_i. Finally, delete from E all disjunctive arcs not incident to J, obtaining E_i. Formally,

$$V_i = \cup\{V_{j,\mu(j)} : j \in I - i; V_{im} : m \in M_i; \{\sigma, \tau\}\}, \tag{3.1}$$

and let $V_i^J = \cup\{V_{j,\mu(j)} : j \in J - i; V_{im} : m \in M_i\}$ be the subset of nodes of G_i associated to job J and the set of arcs $R_J = S - \delta(V_i^J)$ the part of selection (μ, S) not incident to job J. The sets of conjunctive arcs, disjunctive arcs and disjunctive arc pairs are

$$\begin{aligned} A_i &= (A \cap \gamma(V_i)) \cup R_J, \\ E_i &= (E \cap \gamma(V_i)) \cap \delta(V_i^J) \text{ and} \\ \mathcal{E}_i &= \{D \in \mathcal{E} : D \subseteq E_i\}. \end{aligned}$$

Disjunctive graph G_i is called *the insertion graph of job J with local flexibility at i*.

As previously in graph G, we define selections in G_i, called insertions, as follows. For any machine $m \in M_i$ of operation i, let $G_i^m = (V_i^m, A_i^m, E_i^m, \mathcal{E}_i^m, d)$ be the subgraph of G_i obtained by deleting node sets $V_{im'}, m' \in M_i - m$, and denote by $V_J^m = V_i^J \cap V_i^m$ the node set associated to J. G_i^m may be called the insertion graph of job J with i on m.

Definition 5 *For any machine $m \in M_i$ and set of disjunctive arcs $T \subseteq E_i^m$, (m, T) is called an insertion in G_i. Insertion (m, T) is complete if $T \cap D \neq \emptyset$ for all $D \in \mathcal{E}_i^m$. Insertion (m, T) is positive acyclic if subgraph $(V_i^m, A_i^m \cup T, d)$ contains no positive cycle, and is* positive cyclic *otherwise. Insertion (m, T) is* feasible *if it is positive acyclic and complete.*

Obviously, any insertion (m, T) in G_i is positive acyclic (complete, feasible) if and only if the corresponding selection $(\mu', T \cup R_J)$ is positive acyclic (complete, feasible) in G where μ' is given by $\mu'(i) = m$ and $\mu'(j) = \mu(j)$ for all $j \in I - i$.

Trivially, there is always a feasible insertion in G_i, namely $(\mu(i), T^S)$ with $T^S = S \cap \delta(V_i^J)$, which corresponds to the selection (μ, S) in G.

3.3. Structural Properties of Job Insertion

As seen in the Example in Section 3.2, generating a neighbor solution simply by replacing an arc e by its mate $\overline{e}$ may lead to an infeasible solution. In this section, we consider structural properties of job insertion enabling us to generate feasible insertions in the job insertion graph. Key ingredients are the short cycle property, conflict graphs and a closure operator.

3.3.1. The Short Cycle Property

Given a job insertion graph $G_i = (V_i, A_i, E_i, \mathcal{E}_i, d)$ of some operation $i \in I$ belonging to some job J, we examine positive cycles in G_i.

We assume that the conjunctive part (V_i, A_i, d) of G_i does not contain any positive cycle (otherwise no feasible insertion exists). Hence, any positive cycle Z has to "visit" job J, i.e. $Z \cap \delta(V_i^J) \neq \emptyset$.

All conjunctive arcs incident to job J, i.e. from $\delta(V_i^J) - E_i$, are of the type (σ, v) or (v, τ). Such arcs do not appear in any cycle. In addition, all disjunctive arcs are incident to job J, i.e. $E_i \subseteq \delta(V_i^J)$. Hence, for any cycle Z, the disjunctive arcs of Z are exactly those arcs of Z that are incident to job J, i.e. $Z \cap E_i = Z \cap \delta(V_i^J)$.

The number of arcs of a cycle Z leaving J, i.e. $|Z \cap \delta^-(V_i^J)|$ and entering J, i.e. $|Z \cap \delta^+(V_i^J)|$, is equal, namely $|Z \cap \delta(V_i^J)| = 2|Z \cap \delta^-(V_i^J)| = 2|Z \cap \delta^+(V_i^J)| = 2k$ for some $k \geq 0$. The number k can be seen as the number of times cycle Z visits the node set V_i^J of job J, or short, visits job J.

An interesting structural property in relation with positive cycles is the so-called Short Cycle Property (SCP), which was introduced in [42] for general disjunctive graphs and is used here for job insertion graphs.

Definition 6 *A job insertion graph G_i has the SCP if for any positive cycle with arc set Z' in $(V_i^m, A_i^m \cup E_i^m, d), m \in M_i$, there exists a "short positive cycle" Z in $(V_i^m, A_i^m \cup E_i^m, d)$ with $Z \cap E_i^m \subseteq Z' \cap E_i^m$ and $|Z \cap E_i^m| \leq 2$.*

We say that a CJS problem has the SCP if for all operations $i \in I$, G_i has the SCP.

Introduce the following bipartition of the set of disjunctive arcs $E_i = E_i^- \cup E_i^+$ where set $E_i^- = E_i \cap \delta^-(V_i^J)$ and $E_i^+ = E_i \cap \delta^+(V_i^J)$ correspond to the disjunctive arcs in G_i entering and leaving job J, respectively.

Consider the positive acyclic insertions in G_i. They form an independence system, i.e. every subset of a positive acyclic insertion is also positive acyclic, and the empty set is a positive acyclic insertion. The "circuits" of this independence system are the setwise minimal positive cyclic insertions where an insertion (m, T) is (setwise) minimal positive cyclic if (m, T) is positive cyclic and any (m, T') with $T' \subset T$ is positive acyclic. Let $\mathcal{C}$ be the collection of all minimal positive cyclic insertions.

Proposition 7 *Given a job insertion graph $G_i = (V_i, A_i, E_i, \mathcal{E}_i, d)$, the following statements are equivalent:*

(i) $|C \cap E_i^-| = 1$ and $|C \cap E_i^+| = 1$ for all $(m, C) \in \mathcal{C}, m \in M_i$.

(ii) G_i has the SCP.

Proof. [The proof is similar to the proof of Proposition 3 in [42] and is given here for completeness.]

(i)⇒(ii): Let Z' be a positive cycle in $(V_i^m, A_i^m \cup E_i^m, d)$ for some $m \in M_i$. The insertion $(m, Z' \cap E_i^m)$ is clearly positive cyclic, so there exists a minimal positive cyclic insertion $(m, C) \in \mathcal{C}$ such that $C \subseteq Z' \cap E_i^m$ and $|C \cap E_i^-| = 1$ and $|C \cap E_i^+| = 1$. Since insertion (m, C) is positive cyclic, there exists a positive cycle Z in $(V_i^m, A_i^m \cup C, d)$ with $Z \cap E_i^m = C \subseteq Z' \cap E_i^m$ and $|Z \cap E_i^m| = |C| = 2$, proving (ii).

(ii)⇒(i): Suppose that G_i has the SCP. For any positive cyclic insertion (m, T), there exists a short positive cycle Z in $(V_i^m, A_i^m \cup T, d)$ with $|Z \cap T| \leq 2$. Since Z has to enter and leave job J at least once using both times one disjunctive arc,

$|Z \cap T| = 2$, and $|(Z \cap T) \cap E_i^-| = 1$ and $|(Z \cap T) \cap E_i^+| = 1$. Insertion $(m, Z \cap T)$ is itself positive cyclic and contained in insertion (m, T). Hence, any positive cyclic insertion is or contains a positive cyclic insertion consisting of exactly two disjunctive arcs, one entering job J and one leaving job J, implying (i). ■

Not all CJS problems have the SCP. However, we now show that the class of CJS problems *without time lags* possesses the property. Formally, a CJS problem is called a CJS problem without time lags if for all operations $i \in I$ and $m \in M_i$, $d^{\text{lg}}(i, m) = \infty$.

To show that CJS problems without time lags have the SCP, we slightly adapt the concept of through-connectedness introduced in [42]. The following notation will be needed.

In $G_i^m = (V_i^m, A_i^m, E_i^m, \mathcal{E}_i^m, d), m \in M_i$, let $N^- = \{v \in V_J^m : v = h(e) \text{ for some } e \in E_i^m\}$ and $N^+ = \{v \in V_J^m : v = t(e) \text{ for some } e \in E_i^m\}$ be the "entry" and "exit" nodes of the arcs going into and out of (V_J^m of) job J.

Consider the conjunctive part (V_i^m, A_i^m, d) of job insertion graph G_i^m. For any two nodes $v, w \in V_i^m$, let $v \not\rightarrow w$, $v \xrightarrow{0+} w$ and $v \xrightarrow{+} w$ if no path, a path of non-negative length and a path of positive length, respectively, exists from node v to w in (V_i^m, A_i^m, d).

Definition 8 *G_i^m is a through-connected job insertion graph if the following holds:*

a) for any disjunctive arcs $e, e' \in E_i^m$, $h(e) \not\rightarrow t(e')$ or $h(e) \xrightarrow{0+} t(e')$.

b) For any distinct $v_1, v_2 \in N^-$ and distinct $w_1, w_2 \in N^+$: if $v_1 \xrightarrow{0+} w_1$ and $v_2 \xrightarrow{0+} w_2$, then $v_1 \xrightarrow{+} w_2$ or $v_2 \xrightarrow{+} w_1$.

Job insertion graph G_i is then called through-connected if G_i^m is through-connected for all $m \in M_i$.

Note that through-connectedness is defined in [42] for graphs with non-negative arc weights. This condition is replaced here by condition a).

Lemma 9 *If G_i is a through-connected job insertion graph, then G_i has the SCP.*

Proof. [The proof is similar to the proof of Lemma 8 in [42].]

We claim that for any positive cycle Z' in $G_i^m, m \in M_i$, visiting job J $k \geq 1$ times, there exists a positive cycle Z with $Z \cap E_i^m \subseteq Z' \cap E_i^m$ visiting job J exactly once.

We prove the claim by induction on the number of visits k. It trivially holds for $k = 1$. Let Z' be a positive cycle in $(V_i^m, A_i^m \cup E_i^m, d)$ visiting J $k > 1$ times and assume that the claim holds for positive cycles visiting J less than k times.

Rewrite Z' as the concatenation $Z' = (e_1, P_1, e_1', Q_1, e_2, P_2, e_2', Q_2, \ldots, e_k, P_k, e_k', Q_k)$ where e_i's and e_i''s are arcs entering and leaving J, respectively, P_i's are paths joining node $h(e_i) \in N^-$ and node $t(e_i') \in N^+$ through arcs in $\gamma(V_J^m)$, i.e arcs between nodes of job J, $i = 1, \ldots, k$, and the Q_i's are paths joining $h(e_i')$ to $t(e_{i+1})$ through arcs not in $\gamma(V_J^m)$, $i = 1, \ldots, k, (k + 1 \equiv 1)$.

Clearly, $h(e_1) \neq h(e_2), t(e'_1) \neq t(e'_2)$, and by Definition 8 a) $h(e_1) \xrightarrow{0+} t(e'_1)$, $h(e_2) \xrightarrow{0+} t(e'_2)$. Moreover, by Definition 8 b), i) $h(e_1) \xrightarrow{+} t(e'_2)$ or ii) $h(e_2) \xrightarrow{+} t(e'_1)$ holds.

Consider case i) and let P_{vw} be a positive path joining $v = h(e_1)$ to $w = t(e'_2)$. Clearly, path P_{vw} consists of arcs in $\gamma(V_J^m)$.

The paths P_i and $Q_i, i = 1, \ldots, k$ start at a head node $h(e)$ of an arc in $e \in E_i^m$ and end at tail node $t(e)$ of an arc in $e \in E_i^m$. Hence by Definition 8 a), there exists a non-negative length path joining $h(e)$ to $t(e)$. Let P'_i and Q'_i be these non-negative length paths corresponding to P_i and Q_i. Clearly, P'_i consists of arcs in $\gamma(V_J^m)$ and Q'_i consists of arcs not in $\gamma(V_J^m)$.

Then, the concatenation $W = (e_1, P_{vw}, e'_2, Q'_2, \ldots, e_k, P_k, e'_k, Q'_k)$ is a closed walk visiting job J at most $k-1$ times, and there exists a decomposition of W into cycles where each cycle visits J at most k times (cycle decomposition of an integer circulation). The arcs and paths of concatenation W are all of non-negative length, and P_{vw} is of positive length, hence W is of positive length. Thus, there exists a cycle Z'' of positive length in the decomposition visiting job J at least once and at most $k-1$ times, and $Z'' \cap E_i^m \subseteq Z' \cap E_i^m$. Therefore, by induction, there exists a positive cycle Z with $Z \cap E_i^m \subseteq Z' \cap E_i^m$ visiting J exactly once.

Consider case ii) and let P_{vw} be a positive path joining $v = h(e_2)$ to $w = t(e'_1)$. Clearly, path P_{vw} consists of arcs in $\gamma(V_J^m)$. The closed walk $W = (e'_1, Q'_1, e_2, P_{vw})$ is a positive cycle Z with $Z \cap E_i^m \subseteq Z' \cap E_i^m$ visiting J exactly once. ■

Theorem 10 *Given a job insertion graph $G_i, i \in I$, of a CJS problem without time lags.*

i) G_i has the SCP.

ii) $|C \cap E_i^-| = 1$ and $|C \cap E_i^+| = 1$ for all $(m, C) \in \mathcal{C}, m \in M_i$.

Proof. For any $m \in M_i$ we show that $G_i^m = (V_i^m, A_i^m, E_i^m, \mathcal{E}_i^m, d)$ is through-connected. Then, i) is implied by Lemma 9 and i) implies ii) by Proposition 7.

Note that in G_i^m, the mode is fixed to μ' where $\mu'(j) = \mu(j)$ for all $j \in I - i$ and $\mu(i) = m$. We show that a) and b) of Definition 8 are satisfied.

a) Consider paths from some node v to node w in G_i^m where $v = h(e)$ and $w = t(e')$ for some $e, e' \in E_i^m$. Either no path exists from v to w, or the length of a longest path is non-negative. Indeed, as the time lag arcs are not present in G_i^m, the only arcs of negative weight are the transfer arcs of the type $(v^2_{k\mu'(k)}, v^3_{j\mu'(j)})$, where k is the successor operation of j in some job. By considering the job structure, it is easy to see that such negative weight transfer arcs are not on a longest path from v to w. Hence, there is either no path from v to w or the longest path from v to w is of non-negative length, proving a).

b) Let $v_1, v_2 \in N^-, v_1 \neq v_2$, and $w_1, w_2 \in N^+, w_1 \neq w_2$ such that $v_1 \xrightarrow{0+} w_1$ and $v_2 \xrightarrow{0+} w_2$. Nodes v_1 and $v_2 \in N^-$ are start nodes of some transfer steps of operations belonging to job J, so $v_1 = v^1_{j\mu'(j)}$ or $v_1 = v^3_{j\mu'(j)}$ of some operation $j \in J$ and $v_2 = v^1_{k\mu'(k)}$ or $v_2 = v^3_{k\mu'(k)}$ of some operation k. Without loss of generality, we may assume that operation j is before operation k in job J. By the job structure,

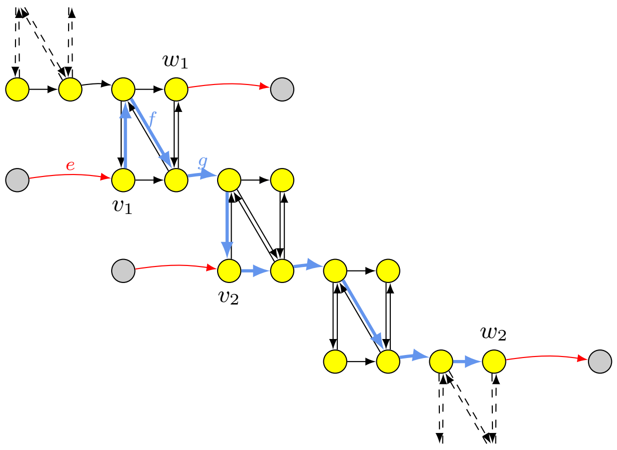


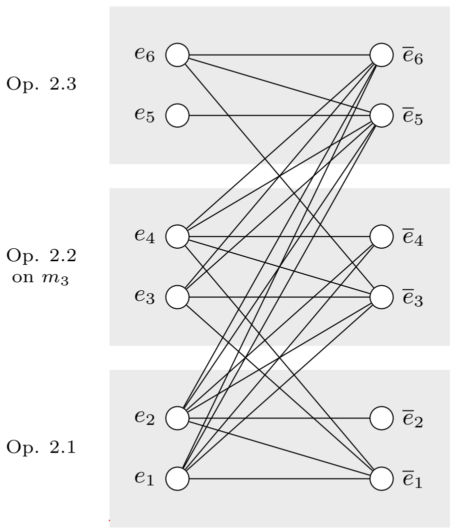
Figure 3.4.: The path from v_1 to v_2 (a part of the arcs in blue) is of positive length as arc f or g is of positive weigth. Therefore, the path from v_1 to w_2 (in blue) is of positive length.

there exists a non-negative path from v_1 to v_2 in (V_i^m, A_i^m, d), so $v_1 \xrightarrow{0+} v_2$. If either $v_1 \xrightarrow{+} v_2$ or $v_2 \xrightarrow{+} w_2$, then there exists a walk from v_1 to w_2 in (V_i^m, A_i^m, d) of positive length. As (V_i^m, A_i^m, d) does not contain any positive cycles, there exists a path from v_1 to w_2 in (V_i^m, A_i^m, d) of positive length, hence $v_1 \xrightarrow{+} w_2$. Otherwise, the longest paths from v_1 to v_2 and from v_2 to w_2 must be of length 0. It remains to show that at least one arc on a longest path from v_1 to v_2 or from v_2 to w_2 is of positive length.

If $v_1 = v^1_{j\mu'(j)}$ represents the start of the take-over step o_j of j on machine $\mu'(j)$ then by $v_1 \in N^-$ some disjunctive arc e is incident to it (see illustration in Figure 3.4). Arc e is either sequencing j with respect to another operation on machine $\mu'(j)$, then the duration of operation j on $\mu'(j)$ must be positive, or e is sequencing the take-over step o_j on $\mu'(j)$ with respect to another transfer step, then the duration of the take-over step must be positive (by the standard assumptions of the CJS model, see Section 2.4). In both cases, the longest path from v_1 to v_2 is of positive length.

If $v_1 = v^3_{j\mu'(j)}$ represents the start of the hand-over $\bar{o}_j$ of j on machine $\mu'(j)$ then by $v_1 \in N^-$ some disjunctive arc e is incident to it. Arc e must be sequencing the hand-over step $\bar{o}_j$ on $\mu'(j)$ with respect to another transfer step. Hence, the duration of the hand-over step $\bar{o}_j$ must be positive. If $k = j + 1$ and $v_2 = v^1_{k\mu'(k)}$, then the hand-over step $\bar{o}_j$ is represented on a path from v_2 to w_2, so that the longest path from v_2 to w_2 is of positive length, and otherwise, $\bar{o}_j$ is represented on a path from v_1 to v_2, so that the longest path from v_1 to v_2 is of positive length. ■

We remark here that there are CJS problems not belonging to the class of CJS problems without time lags that also possess the SCP, among them the (flexible) NWJSS problem. In the NWJSS, the time lag $d^{\text{lg}}(i, m)$ is equal to the processing time $d^{\text{p}}(i, m)$ for all operations $i \in I$ and machines $m \in M_i$. The proof of the SCP



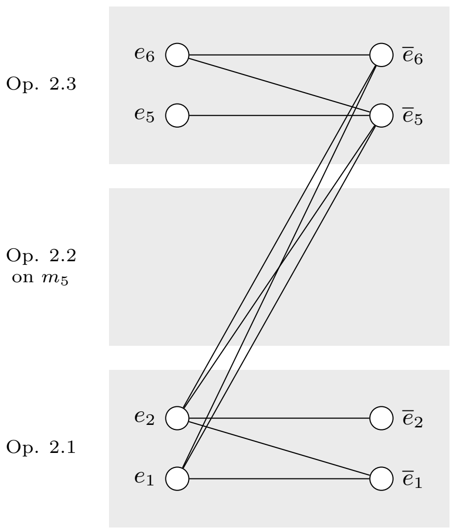


Figure 3.5.: The conflict graphs H^m with $m = m_3$ (left) and $m = m_5$ (right) in the Example.

for the NWJSS is different from the one given above, and we refer the reader to [42] for further details. The NWJSS is not discussed further in this thesis, however we point out the publication [20] on optimal job insertion in the no-wait job shop, where a different job insertion based approach is used to solve the NWJSS.

3.3.2. The Conflict Graph and the Fundamental Theorem

The concept of conflict graphs introduced in [42] for general disjunctive graphs is used here for job insertion graphs $G_i^m = (V_i^m, A_i^m, E_i^m, \mathcal{E}_i^m, d), m \in M_i$.

Denote by $E_i^m = E_i^{m-} \cup E_i^{m+}$ the bipartition of the set of disjunctive arcs in G_i^m where $E_i^{m-} = E_i^m \cap E_i^-$ and $E_i^{m+} = E_i^m \cap E_i^+$.

Definition 11 *The conflict graph of job insertion graph $G_i^m = (V_i^m, A_i^m, E_i^m, \mathcal{E}_i^m, d)$, $m \in M_i$, is the undirected bipartite graph $H^m = (E_i^m, U^m)$ where for any pair of disjunctive arcs $e, f \in E_i^m$, edge $(e, f) \in U$ is present in conflict graph H^m if insertion $(m, \{e, f\})$ is a minimal positive cyclic insertion, i.e. $(m, \{e, f\}) \in \mathcal{C}$ for some $m \in M_i$.*

Take again the Example and consider the job insertion graph of job 2 with local flexibility at operation 2.2 (see Section 3.2). Figure 3.5 depicts its two conflict graphs H^m with operation 2.2 assigned to machine $m = m_3$ (left) and $m = m_5$ (right).

We now establish the fundamental theorem stating that the feasible insertions are precisely the stable sets (of a prescribed cardinality) in the conflict graphs $H^m, m \in M_i$.

#	m	T
1	m_3	$e_1, e_2, e_3, e_4, e_5, e_6$
2	m_3	$e_1, \overline{e}_2, e_3, e_4, e_5, e_6$
3	m_3	$\overline{e}_1, \overline{e}_2, \overline{e}_3, \overline{e}_4, e_5, \overline{e}_6$
4	m_3	$\overline{e}_1, \overline{e}_2, \overline{e}_3, \overline{e}_4, \overline{e}_5, \overline{e}_6$
5	m_5	e_1, e_2, e_5, e_6
6	m_5	$e_1, \overline{e}_2, e_5, e_6$
7	m_5	$\overline{e}_1, \overline{e}_2, e_5, e_6$
8	m_5	$\overline{e}_1, \overline{e}_2, e_5, \overline{e}_6$
9	m_5	$\overline{e}_1, \overline{e}_2, \overline{e}_5, \overline{e}_6$

Table 3.1.: All feasible insertions (m, T) in the Example.

Theorem 12 *Let G_i be a job insertion graph with the SCP. There is a one-to-one correspondence between the feasible insertions in G_i and the stable sets T in conflict graphs $H^m, m \in M_i$, satisfying $T \subseteq E_i^m$ and $|T| = |E_i^m|/2$.*

Proof. [The proof is similar to the proof of Theorem 5 in [42] and given here for completeness.]

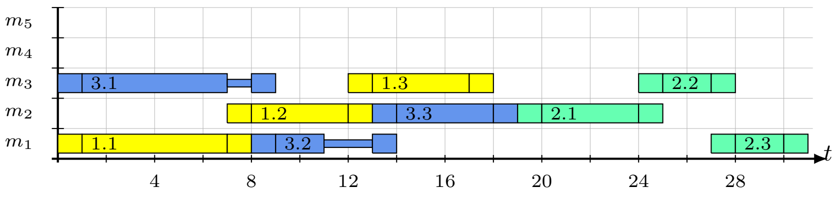
First, we show that any insertion $(m, T), m \in M_i$ is positive acyclic if and only if $T \subseteq E_i^m$ and T is stable in H^m. Observe that $T \subseteq E_i^m$ holds by definition for any insertion (m, T). By Proposition 7, $|C| = 2$ for all $(m', C) \in \mathcal{C}, m' \in M_i$, therefore an insertion (m, T) is positive acyclic if and only if $|T \cap C| \leq 1$ for all $(m, C) \in \mathcal{C}$. These are precisely the conditions for insertion (m, T) to be stable in H^m and satisfying $T \subseteq E_i^m$.

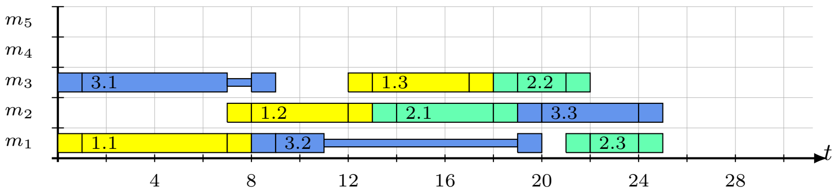
We now show that $|T \cap E_i^m| = |E_i^m|/2$. Since we assume that all disjunctive pairs $D = \{e, \overline{e}\} \in \mathcal{E}_i$ are positive cyclic by (2.24), $|T \cap D| \leq 1$ for all $D \in \mathcal{E}_i^m$, hence $|T| \leq |E_i^m|/2$. If additionally $|T| = |E_i^m|/2$, then $|T \cap D| = 1$ for all $D \in \mathcal{E}_i^m$, so that (m, T) is also complete, hence feasible. Conversely if (m, T) is feasible, (m, T) is positive acyclic and $|T \cap D| = 1$ for all $D \in \mathcal{E}_i^m$, hence $|T| = |E_i^m|/2$. ■

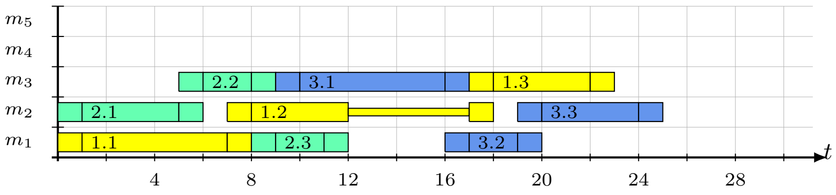
Consider the feasible insertions in the Example, which clearly belongs to the class of CJS problems without time-lags and therefore possesses the SCP. By Theorem 12, we can generate all feasible insertions by computing for each machine $m \in M_i$ all stable sets T of cardinality $|T| = |E_i^m|/2$ in conflict graph H^m. The obtained nine feasible insertions are listed in Table 3.1 using the notation introduced in Figure 3.3. The corresponding schedules are given in Figure 3.6 (with operation 2.2 on machine m_3) and in Figure 3.7 (with operation 2.2 on machine m_5).

3.3.3. A Closure Operator

Consider again the Example and the current feasible insertion of job 2 given in Figure 3.1 (which is the insertion #2 in Table 3.1). The representation of this insertion as a stable set in conflict graph H^{m_3} is depicted by the red nodes in Figure 3.8.

(1) 

(2) 

(3) 

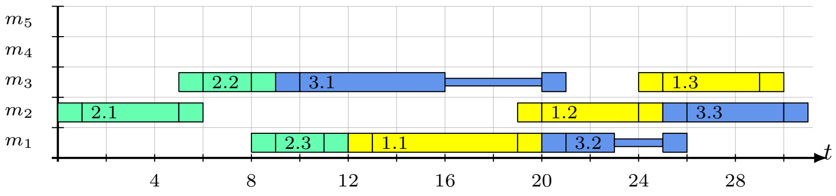
(4) 

Figure 3.6.: The schedules of the feasible job insertions with operation 2.2 on machine m_3. The number in brackets refers to the number assigned in Table 3.1.

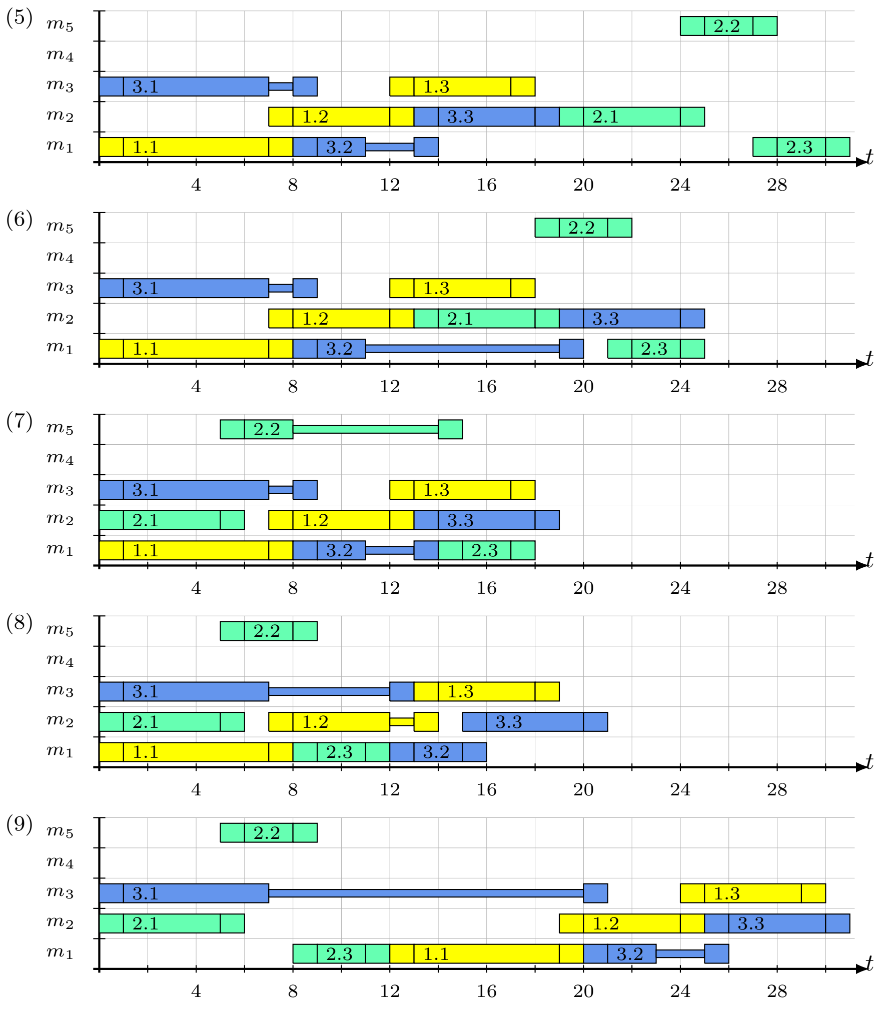


Figure 3.7.: The schedules of the feasible job insertions with operation 2.2 on machine m_5.

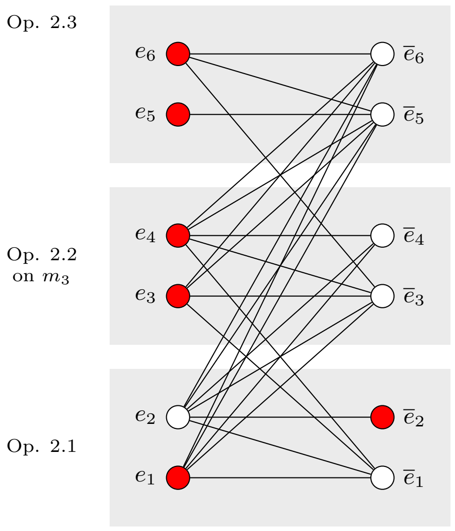


Figure 3.8.: The current insertion (red nodes) in conflict graph H^{m_3}.

We noted in Section 3.2.1 that replacing arc $e_4 = (v^4_{1.3,m_3}, v^1_{2.2,m_3})$ by its mate $\overline{e}_4$ leads to an infeasible solution and stated that other changes are needed to maintain feasibility. This can now be illustrated in the conflict graph H^{m_3} of the Example. By Theorem 12, any feasible insertion containing $\overline{e}_4$ corresponds to a stable set T in H^m with $\overline{e}_4 \in T$ and $|T| = |E^m_i|/2 = 6$. Let us construct such a neighbor insertion step by step.

Clearly, T must contain either e_i or $\overline{e}_i, i = 1, \dots, 6$, as $T = |E^m_i|/2$ and $(e_i, \overline{e}_i) \in U^m$ for all $i = 1, \dots, 6$. Since $(\overline{e}_4, e_1) \in U^m$, we cannot choose e_1, hence $\overline{e}_1 \in T$, and similarly, $\overline{e}_2 \in T$ must be part of our feasible insertion T. $\overline{e}_1$ and $\overline{e}_2$ are said to be implied by $\overline{e}_4$. Then, $\overline{e}_1$ implies another disjunctive arc, namely $\overline{e}_3$, and similarly $\overline{e}_3$ implies $\overline{e}_6$. By these implications, T must contain $\{\overline{e}_1, \overline{e}_2, \overline{e}_3, \overline{e}_4, \overline{e}_6\}$.

The "remaining" pair is $\{e_5, \overline{e}_5\}$ and in order to be "close" to the current insertion, we choose the arc e_5 present in the current insertion, obtaining the feasible neighbor insertion $T = \{\overline{e}_1, \overline{e}_2, \overline{e}_3, \overline{e}_4, e_5, \overline{e}_6\}$, which is the insertion #3 in Table 3.1.

Formally, the implications sketched above can be described as follows. For any $e, f \in E^m_i$ in conflict graph $H^m = (E^m_i, U^m)$ let

$$e \to f \Leftrightarrow (e, \overline{f}) \in U^m.$$

Definition 13 *A sequence $P = (e_0, e_1, \dots, e_n), n \geq 0,$ of distinct nodes in $H^m = (E^m_i, U^m)$ is an alternating path from e_0 to e_n if $e_i \to e_{i+1}$ for all $0 \leq i < n$. For any two nodes $e, f \in E^m_i$, we write $e \rightsquigarrow f$ if there exists an alternating path from e to f in H^m.*

The alternating paths capture the concept of implied disjunctive arcs. Indeed, if some e is to be part of a feasible insertion, then all $f \in E^m_i$ with $e \rightsquigarrow f$ must be part of that insertion. Similarly, if a set Q of disjunctive arcs is to be part of a feasible insertion, all $f \in E_i^m$ that are reachable by an alternating path starting at some $e \in Q$ are implied. This leads to the following definition.

Definition 14 *For any $Q \in E_i^m$, the closure of Q is the set*

$$\Phi(Q) = \{f \in E_i^m : e \rightsquigarrow f \text{ for some } e \in Q\}. \tag{3.2}$$

Also $Q \subseteq E_i^m$ is said to be closed if $Q = \Phi(Q)$.

Observe that $e \rightsquigarrow e$ holds for any $e \in E_i^m$. Moveover, $\Phi(Q)$ can be rewritten as $\Phi(Q) = \bigcup_{e \in Q} \Phi(e)$. Then, it is easy to see that Φ is a well-defined topological closure operator fulfilling the following properties: i) $\Phi(\emptyset) = \emptyset$ (preservation of nullary unions), ii) $Q \subseteq \Phi(Q)$ (inflationary), iii) $Q \subseteq R \Rightarrow \Phi(Q) \subseteq \Phi(R)$ (monotone), iv) $\Phi(\Phi(Q)) = \Phi(Q)$ (idempotence) and v) $\Phi(Q \cup R) = \Phi(Q) \cup \Phi(R)$ (join homomorphism).

Obviously, depending on the choice of the set Q, there exists a feasible insertion (m, T) containing Q or not. The following questions arise naturally: For which sets Q do feasible insertions containing Q exist, and if they exist, how are they constructed. These questions are addressed now.

The following definitions are needed. For any subset of nodes $T \in E_i^m$, the *span* $[T]$ of T contains all nodes of T and all mates $\overline{e}$ of $e \in T$. Formally,

$$[T] = \{e \in E_i^m : \{e, \overline{e}\} \cap T \neq \emptyset\}.$$

Also, for any set $Q \subseteq E_i^m$, let $H^Q = (E^Q, U^Q)$ be the bipartite subgraph of $H^m = (E_i^m, U^m)$ obtained by deleting node set $[Q]$, i.e. all nodes of the span of Q are deleted, and denote by E^{Q-}, E^{Q+} its node bipartition. Observe that a pair $\{e, \overline{e}\} \in E^Q$ is either present in H^Q or not, so that $|E^{Q-}| = |E^{Q+}|$.

Theorem 15 *For any $Q \subseteq E_i^m$, (m, T) is a feasible insertion with $Q \subseteq T$ if and only if $T = \Phi(Q) \cup T'$ where $\Phi(Q)$ is stable in H^m and T' is stable in $H^{\Phi(Q)}$ with $T' = |E^{\Phi(Q)}|/2$.*

Proof. [The proof is similar to the proof of Theorem 9 in [44].]

Let (m, T) be a feasible insertion with $Q \subseteq T$. By Theorem 12, T is stable in H^m and $|T| = |E_i^m|/2$. We show first that $\Phi(Q) \subseteq T$, i.e. if $e \in Q$ and $e \rightsquigarrow f$, then $f \in T$. Indeed let $P = (e = e_0, e_1, \ldots, e_n = f)$ be an alternating path from e to f. Assume $e_{k-1} \in T$ for some $k > 0$. Since T is stable in H^m and $(e_{k-1}, \overline{e}_k) \in U^m, \overline{e}_k \notin T$, and since (m, T) is complete, $e_k \in T$.

Since $\Phi(Q) \subseteq T, \Phi(Q)$ is stable in H^m and, choosing $T' = T \cap E^{\Phi(Q)}$, T' is stable in $H^{\Phi(Q)}$ with $|T'| = |E^{\Phi(Q)}|/2$, proving necessity.

Conversely, assume $\Phi(Q)$ stable in H^m and $T' \subseteq E^{\Phi(Q)}$ stable in $H^{\Phi(Q)}$ with $|T'| = |E^{\Phi(Q)}|/2$. Let $T = \Phi(Q) \cup T'$. Clearly $\Phi(Q) \cap T' = \emptyset$ and $|T| = |\Phi(Q)| + |T'| = |[\Phi(Q)]|/2 + |E^{\Phi(Q)}|/2 = |E_i^m|/2$, hence (m, T) is complete. T is also stable in H^m. Indeed, suppose there exists $(f, g) \in U^m$ with $f \in \Phi(Q), g \in T'$. By construction of

$H^{\Phi(Q)}$, $g \neq \overline{f}$. Also, $e \rightsquigarrow f$ for some $e \in Q$, but then $(f, g) \in U^m$ and $g \neq \overline{f}$ imply that $e \rightsquigarrow \overline{g}$, and therefore $\overline{g} \in \Phi(Q)$, a contradiction to $g \in T' \subseteq E^{\Phi(Q)}$. Therefore, $T = \Phi(Q) \cup T'$ is stable in H^m with $|T| = |E_i^m|/2$, hence is a feasible insertion by Theorem 12, proving sufficiency. ■

Corollary 16 *For any $Q \subseteq E_i^m$, there exists a feasible insertion (m, T) with $Q \subseteq T$ if and only if $\Phi(Q)$ is stable in H^m.*

Proof. There always exists $T' \subseteq E^{\Phi(Q)}$ stable in $H^{\Phi(Q)}$ with $|T'| = |E^{\Phi(Q)}|/2$, namely $T' = E^{\Phi(Q)-}$ or $T' = E^{\Phi(Q)+}$. Therefore $\Phi(Q)$ is stable in H^m is a sufficient condition for the existence of a feasible insertion (m, T) with $Q \subseteq T$. ■

3.4. Neighbor Generation

In this section, we use the results of the previous sections to derive feasible neighbors in CJS problems possessing the SCP. We first recall the local search principle introduced in Section 3.2 using the notation and terms introduced so far.

Given is a feasible selection (μ, S) of some CJS problem with the SCP. Select some operation $i \in I$ of some job J and consider the job insertion graph G_i of job J with local flexibility at i. Let $(\mu(i), T^S)$ with $T^S = S \cap \delta(V_i^J)$ be the insertion of job J that corresponds to the current selection and $R_J = S - T^S$ the part of S not incident to J. A feasible neighbor selection can be constructed by building a feasible neighbor insertion (m, T) of $(\mu(i), T^S)$ in job insertion graph G_i, and letting neighbor selection be (μ', S') where $\mu'(i) = m$, $\mu'(j) = \mu(j)$ for all $j \in I - i$ and $S' = T \cup R_J$.

We build a neighbor insertion (m, T) by keeping the machine assignment for operation i and forcing some disjunctive arc $f \notin T^S$ to be in the insertion, building a so-called "non-flexible" neighbor (m, T_f) with $m = \mu(i)$, or by assigning operation i to another machine and inserting it on this machine forcing some arc set F to be in the insertion, building a so-called "flexible" neighbor (m, T_F) with $m \neq \mu(i)$. Arcs F will be chosen to "position" operation i on machine m with respect to the other operations on m and with respect to take-over and hand-over steps in conflict with steps of i.

We show first how to build a non-flexible neighbor (m, T_f) and then a flexible neighbor (m, T_F).

3.4.1. Non-Flexible Neighbors

Non-flexible neighbors (m, T_f) with $m = \mu(i)$ are constructed by the following two successive steps: i) Take f and all arcs implied by f, forming $\Phi(f)$, ii) keep T^S on the remaining part. Specifically,

$$T_f = \Phi(f) \cup T^S - [\Phi(f)]. \tag{3.3}$$

Theorem 17 *Insertion (m, T_f) with $m = \mu(i)$ given by (3.3) is a feasible neighbor insertion of $(\mu(i), T^S)$.*

Proof. It is sufficient to show that i) $Q = \Phi(f)$ is stable in conflict graph H^m and ii) $T^S - [Q]$ is stable in H^Q with $|T^S - [Q]| = |E^{\Phi(Q)}|/2$, and to apply Theorem 15.

i) Clearly, if $f \in E_i^{m-}$ then $\Phi(f) \subseteq E_i^{m-}$ and if $f \in E_i^{m+}$ then $\Phi(f) \subseteq E_i^{m+}$, hence $\Phi(f)$ is stable in H^m.

ii) Since $(\mu(i), T^S)$ is a feasible insertion and $m = \mu(i)$, T^S is stable in H^m, hence $T^S - [Q]$ is stable in subgraph H^Q. Also, $|T^S - [Q]| = |E^Q|/2$ since $|T^S \cap \{g, \overline{g}\}| = 1$ for all $\{g, \overline{g}\} \subseteq E^Q$. ■

3.4.2. Flexible Neighbors

Flexible neighbors (m, T_F) with $m \neq \mu(i)$ are constructed as follows. In order to generate a "close" neighbor insertion, the choice of arc set F is made such that i is likely to be scheduled at a time not too far from its time in the current schedule. For this reason, we consider the starting times of the operations in the current schedule. If the take-over of an operation j on m starts not earlier than the hand-over of i, we choose to sequence i before j. Moreover, if some transfer step o' of operation j on $\mu(j)$, j belonging to some job $K \neq J$, is in conflict with a transfer step $o = o_i$ or $o = \overline{o}_i$ of i on m, i.e. $\{(o, m), (o', \mu(j))\} \in \mathcal{V}$ and o' starts not earlier than o' in the current schedule, then we choose to sequence o before o'.

Specifically, F is built as follows. Let $\alpha(v)$, $v \in V^\mu$, be the earliest starting times computed in graph $G(\mu, S)$ of the current selection (μ, S). Let $F = F_1 \cup F_2 \subseteq E_i^{m+}$ where $F_1 = \{e \in \delta^+(v_{im}^2) : \alpha(h(e)) \geq \alpha(v_{i,\mu(i)}^1)\}$ and $F_2 = \{e \in \delta^+(v_{im}^4) : \alpha(h(e)) \geq \alpha(v_{i,\mu(i)}^3)\}$.

Let $T^m = T^S \cap E_i^m$. Neighbor (m, T_F) is then constructed by the following three successive steps: i) Take F and all arcs implied, forming $\Phi(F)$, ii) place before i all operations on m that have not already been sequenced with respect to i in step i), and similarly, place before the transfer steps of i all transfer steps in conflict with these steps that have not already been sequenced in step i), forming $\Phi(E_i^{m-} - [\Phi(F)])$, iii) keep T^m on the remaining part. Specifically,

$$T_F = Q \cup (T^m - [Q]) \text{ where} \tag{3.4}$$

$$Q = \Phi(F) \cup \Phi(E_i^{m-} - [\Phi(F)]) \tag{3.5}$$

Theorem 18 *Insertion (m, T_F) given by (3.4) and (3.5) is a feasible neighbor insertion of $(\mu(i), T^S)$.*

Proof. [The proof is similar to the proof of Theorem 11 in [44].]

We prove that (m, T_F) is a feasible insertion in G_i^m with conflict graph H^m by showing i) $Q = \Phi(Q)$, ii) Q is stable in conflict graph H^m and iii) $T^m - [Q]$ is stable in H^Q with $|T^m - [Q]| = |E^{\Phi(Q)}|/2$, and applying Theorem 15.

i) Let $P = \Phi(F)$. Since both sets P and $\Phi(E' - [\Phi(F)])$ are closed, their union Q is also closed, i.e. $Q = \Phi(Q)$ (join homomorphism of the closure Φ).

ii) First, $F \subseteq E_i^{m+}$, implies $P = \Phi(F) \subseteq E_i^{m+}$, and similarly, $E' \subseteq E_i^{m-}$ implies $\Phi(E' - [P]) \subseteq E_i^{m-}$, hence both sets $\Phi(F)$ and $\Phi(E' - [P])$ are stable in H^m. Next, observe that

$$a \notin [P] \text{ and } \bar{b} \in P \Rightarrow (a, \bar{b}) \notin U^m \tag{3.6}$$

since $\bar{b} \in P, a \neq b, (a, \bar{b}) \in U^m$ implies $\bar{a} \in P$. Furthermore,

$$[P] \cap \Phi(E' - [P]) = \emptyset \tag{3.7}$$

Indeed, suppose $g \in [P] \cap \Phi(E' - [P])$ Then there exists $f \in E' - [P]$ such that $f \rightsquigarrow g$. On a corresponding alternating path form f to g, there is an edge $(a, \bar{b}) \in U^m$ with $a \in E' - [P]$ and $b \in P$, contradicting (3.6). As a result, by (3.6) and (3.7), Q is stable in H^m.

iii) Since $(\mu(i), T^S)$ is a feasible insertion, $T^m = T^S \cap E_i^m$ is stable in H^m, hence $T^m - [Q]$ is stable in subgraph H^Q. Also, $T^m - [Q] = |E^Q|/2$, since $|T' \cap \{f, \bar{f}\}| = 1$ for all $\{f, \bar{f}\} \subseteq E^Q$. ■

3.4.3. A Neighborhood

In principle, the generation schemes described in Section 3.4 allow to define a large set of neighbors given the possible choices of operation i, machine m and disjunctive arc f. In order to generate neighbors that are potentially better than the current solution, we may restrict the choices, having the following idea in mind.

The makespan of a selection (μ, S) is determined by a longest path from σ to τ in graph $G(\mu, S) = (V^\mu, A^\mu \cup S, d)$. Let L be the arc set of such a longest path. Any selection (μ', S') containing in graph $G(\mu', S')$ arc set L, i.e. $L \subseteq A^{\mu'} \cup S'$ cannot have a smaller makespan than (μ, S). In order to improve (μ, S), we must "destroy" path L by replacing some disjunctive arc on L.

The selected disjunctive arcs on L, i.e. $S \cap L$, are usually called *critical arcs*. For any arc $e \in S$, let i be the *tail operation* of e if $t(e) \in V_{i\mu(i)}$ and the *head operation* of e if $h(e) \in V_{i\mu(i)}$. Operations that are tail or head operations of critical arcs are called *critical operations*.

The following neighbor insertions are considered. For each critical arc e and incident operation i (i.e. the tail and head operation of e), we build a neighbor insertion (m, T_f) in G_i based on replacing arc e by its mate $\bar{e}$ according to (3.3) with $m = \mu(i)$ and $f = \bar{e}$. Additionally, for each critical operation i and machine $m \in M_i - \mu(i)$, we build neighbor insertion (m, T_F) in G_i according to (3.4) and (3.5) with i assigned to m.

The neighborhood consisting of all described neighbors will be referred to as $\mathcal{N}$, and let $\mathcal{N}(\mu, S)$ be the set of neighbors of selection (μ, S). The size of neighborhood $\mathcal{N}$ depends on the number of critical arcs, critical operations and the degree of the routing flexibility. Two neighbors are built for each critical arc e and one for each critical operation i and machine $m \in M_i - \mu(i)$.

3.5. The Job Insertion Based Local Search (JIBLS)

In principle, neighborhood $\mathcal{N}$ can be used in any local search scheme such as hill climbing and descent methods, tabu search and simulated annealing. In the JIBLS, a tabu search with some general features that proved useful in local search for scheduling problems is applied.

In this section, we first describe the general principles of local and tabu search, and discuss then the tabu search we used in the JIBLS, providing also a pseudo-code implementation.

3.5.1. From Local Search to Tabu Search

Local search methods are based on the exploration of a set of solutions by repeatedly moving from the current solution to another solution located in the current solution's neighborhood, aiming to reach an optimal or at least a good solution. As input, local search uses an initial solution s, a neighborhood function defining for any solution s a set of neighbor solutions $N(s)$, an objective function assigning an objective value $f(s)$ to any solution s, and a stopping criterion. Local search acts then as follows. As long as the stopping criterion is not met, the neighborhood $N(s)$ of the current solution s is evaluated and the best neighbor s^* is set as current solution, i.e. $s := s^*$.

Assume that we have a minimization problem, i.e. we are seeking a solution s minimizing $f(s)$. Then, the "best" neighbor s^* in the neighborhood of s may be in the simplest case the neighbor with the minimum objective value, so s^* is such that $f(s^*) = \min\{f(s') : s' \in N(s)\}$, and we may stop the search if $f(s^*) \geq f(s)$, i.e. there is no better solution in the neighborhood of s, so that s is a local optimal solution with respect to neighborhood N. This local search technique is well-known as steepest descent.

If such a descent method is applied, it reaches at best, i.e. if the search terminates, a local optimal solution. It is a well suited method if all local optimal solutions are global optimal or at least of an acceptable solution quality. In the other case, however, the descent method may be trapped in a local optimal solution of poor quality.

In most scheduling problems tackled with local search procedures, the quality of local optima cannot be guaranteed. In fact, local optima with poor objective value are quite common. Therefore, procedures have been devised that can escape from local optimal solutions. For this purpose, moves from the current solution s to a solution s^* with $f(s^*) \geq f(s)$, i.e. non-improving moves, are being allowed to some extent. Accepting non-improving moves has however one severe drawback; cycling may occur making the local search to visit repeatedly the same solution.

In order to avoid being trapped in such cycles, the tabu search approach uses a memory structure called tabu list. In the tabu list, moves that could lead to a recently visited solution are penalized or forbidden. Such moves are said to be tabu. The "best" neighbor is then determined with respect to the objective function and the tabu list (see e.g. Glover and Laguna [38]). For any solution s and tabu list L, let function $g(s, L)$ assign a penalization value. Then, a general form of the tabu search may look as follows.

Given some initial solution s and let tabu list $L := \emptyset$ and $\widehat{s} := s$ is the best solution found so far.
While the stopping criterion is not satisfied do:

(i) Generate neighborhood $N := N(s)$;
(ii) Determine the best neighbor $s^* \in N$ with respect to the objective function f and penalization function g;
(iii) If $f(s^*) < f(\widehat{s})$ then update $\widehat{s} := s^*$;
(iv) Set $s := s^*$;
(v) Update tabu list L;

In its simplest form, the tabu list L is a list of bounded size, say $|L| \leq maxL$, containing the last $maxL$ visited solutions. Then, a solution s' is called tabu if $s' \in L$. The best neighbor may be determined as follows. Let $g(s', L) = B$ if neighbor $s' \in L$ where B is a large number and $g(s', L) = 0$ if $s' \notin L$. Then, the "corrected" objective value $h(s', L)$ of a neighbor s' is $h(s', L) = f(s') + g(s', L)$, and the best neighbor s^* is such that $h(s^*, L) = \min\{h(s', L) : s' \in N\}$. Updating the tabu list consists in adding the new solution s^* to L and removing the oldest entry if $|L| > maxL$.

It may be remarked that various tabu search versions exist. For example, the best neighbor may be determined in another fashion, some subset of the neighbors are generated instead of the whole neighborhood, or the tabu list may contain some attributes of forbidden solutions instead of the entire solutions. For more details, we refer the reader to the seminal articles of Glover [37, 39, 38].

3.5.2. The Tabu Search in the JIBLS

We now present the tabu search used in the JIBLS. It is based on neighborhood $\mathcal{N}$ developed in Section 3.4.3 and uses some general features that proved to be appropriate in local search methods for scheduling problems. The JIBLS can be used for all CJS problems possessing the SCP.

A tabu list L is used that stores entries of the last $maxL$ iterations. An entry consists of some attributes of a solution, namely either a disjunctive arc or a machine assignment of an operation. Initially, the list is empty. In an iteration, i.e. after moving from the current selection (μ, S) to a neighbor selection (μ', S'), the tabu list L is obtained by dropping the oldest entry if $|L| = maxL$ and adding at first position arc $e = \overline{f}$ if the neighbor is generated with an insertion (m, T_f) with $m = \mu(i)$ according to (3.3), or the entry $(i, \mu(i))$ if operation i is moved from machine $\mu(i)$ to $m \neq \mu(i)$ with insertion (m, T_F) according to (3.4) - (3.5). Hence, the tabu list L contains arcs that were forced to be replaced and machine assignments that were changed. A neighbor (μ', S') is called tabu if $S' \cap L \neq \emptyset$ or $\mu'(i) = m$ for some $(i, m) \in L$.

Based on the tabu list, the following penalization function g is used. If (μ', S') is not tabu, $g(\mu', S'; L) = 0$, otherwise let k be the position of the first entry in the tabu list giving the move a tabu status, i.e. $L[k] \in S'$ or $\mu'(i) = m$ for $(i, m) = L[k]$ where $L[k]$ is the k's entry in the list, then $g(\mu', S'; L) = (maxL + 1 - k)B$. Note that number B is chosen such that it is larger than the makespan of any feasible selection.

In order to evaluate the neighborhood, we assign to each neighbor (μ', S') an objective value $h(\mu', S'; L; z)$ based on its makespan $\omega(\mu', S')$, the best makespan found so far z and the penalization function $g(\mu', S'; L)$ as follows. If $\omega(\mu', S') < z$, then $h(\mu', S'; L; z) = \omega(\mu', S')$, otherwise $h(\mu', S'; L; z) = \omega(\mu', S') + g(\mu', S'; L)$. Hence, the tabu status of a neighbor is not considered if the neighbor improves the best makespan found so far. Such an "overrule" of the tabu status is used as the tabu list may penalize attractive moves to new best solutions. The best neighbor (μ^*, S^*) of (μ, S) is chosen such that $h(\mu^*, S^*; L; z) = \min\{h(\mu', S'; L; z) : (\mu', S') \in \mathcal{N}(\mu, S)\}$.

The following two additional long term memory structures also used by Nowicki and Smutnicki [84] in the JS are implemented to improve the performance of the tabu search.

As the tabu search does not prevent being trapped in cycles that are longer than $maxL$, we use a list C that keeps track of the sequence of makespans encountered during the search. Cycles are detected by scanning list C for repeated subsequences. Specifically, the search is said to be cycling at iteration k if there exists a period $\delta, maxL < \delta < maxIter$ such that $C[k] = C[k - a * \delta]$ for $a = 1, \ldots, maxC$, where $C[j]$ is the makespan in iteration j, and $maxC$ and $maxIter$ are input parameters.

A list E of bounded length $maxE$ of so-called elite solutions is maintained to diversify the search. Initially, list E is empty and a new solution (μ, S) is added to E if its makespan $\omega(\mu, S)$ is lower than the best makespan found so far. If the tabu search runs for a given number $maxIter$ of iterations without improving the best makespan, or if a solution has no neighbors, or if a cycle is detected, then the current search path is terminated and the search is resumed from the last elite solution in list E. For this purpose, an elite solution is stored together with its tabu list and all neighbors that have not been directly visited from this solution. An elite solution is deleted from E if its set of neighbors is empty.

As stopping criterion, we use the computation time of the tabu search and limit this time to $maxT$.

The sketched tabu search is described in pseudocode in Listing 3.1. Note that $time()$ is a method returning the runtime of the algorithm and method $isCycle(C, iter)$ is the aforementioned cycle check.

Listing 3.1: The tabu search in the JIBLS.

```
iter := 0;
computeNeighbor := true;
saveElite := false;
let $(\mu, S)$ be the initial solution;
initialize the best solution $(\widehat{\mu}, \widehat{S}) := (\mu, S)$;
tabu list $L$, elite solutions $E$ and makespan list $C$ are empty;

while (time() < $maxT$) {
  iter := iter + 1;
  if(computeNeighbor = true) {$N := \mathcal{N}(\mu, S)$;}
  generate all neighbors $(\mu', S') \in N$;
  determine best neighbor $(\mu^*, S^*)$ of $N$;
  if(saveElite = true and $|N| > 1$) {
    remove $(\mu^*, S^*)$ from $N$;
    add $(\mu, S)$ with neighbors $N$ and tabu list $L$ to $E$;
    if(|E| > $maxE$){remove oldest entry in $E$;}
    saveElite := false;
  }
  set $(\mu, S) := (\mu^*, S^*)$;
  update tabu list $T$;
  set C[iter] := $\omega(\mu^*, S^*)$;
  if($\omega(\mu^*, S^*) < \omega(\widehat{\mu}, \widehat{S})$) {
    $(\widehat{\mu}, \widehat{S}) := (\mu^*, S^*)$;
    iter := 0;
    saveElite := true;
  } else if(isCycle($C$, iter) = true or iter > $maxIter$) {
    if($|E| = 0$) {stop the search;}
    take the last entry in $E$ and remove it from the list;
    set $(\mu, S)$, $N$ and $L$ according to this entry;
    computeN := false;
    iter := 0;
    saveElite := true;
  }
}
```

Part II.

The JIBLS in a Selection of CJS Problems

In this part, the CJS model and the JIBLS solution method developed in Part I are tailored and applied to a selection of complex job shop problems. Some of the selected problems have been studied by other authors and benchmarks are available while the others are new. Among the first are the Flexible Job Shop with Setup Times (FJSS), the Job Shop with Transportation (JS-T) and the Blocking Job Shop (BJS), and among the second are the Flexible Blocking Job Shop with Transfer and Setup Times (FBJSS), the Blocking Job Shop with Transportation (BJS-T) and the Blocking Job Shop with Rail-Bound Transportation (BJS-RT). The JS-T, BJS-T and BJS-RT are versions of the FJSS and FBJSS where the jobs are transported from one machine to the next by robots.

In order to evaluate the performance of the JIBLS, we compare the obtained results to the best methods of the literature for the problems with available benchmarks and to results obtained by a MIP approach for the new problems.

CHAPTER 4 THE FLEXIBLE JOB SHOP WITH SETUP TIMES

4.1. Introduction

The Flexible Job Shop with Setup Times (FJSS) is an extension of the JS characterized by two additional features, namely sequence-dependent setup times and routing flexibility. Setup times occur if a machine has to be somewhat prepared before executing an operation. The setup times may be sequence-dependent, i.e. the setup time depends on the current and on the immediately preceding operation on the machine. Routing flexibility is the option to assign a machine for each operation from an (operation-dependent) set of machines.

Both features are common in practice. Sequence-dependent setup times are for example present when the machines are mobile and have to execute an idle move between two consecutive operations. Routing flexibility is mainly achieved by the availability of multiple identical machines and by a machine's capability of performing different operations. For further information, we point to Section 1.3 where both features are also discussed.

This chapter is organized as follows. A selective literature review is given in the following section. Section 4.3 formally introduces the FJSS. Section 4.4 shows that the FJSS problem is an instance of the CJS model. A more compact disjunctive graph formulation of the FJSS is given in Section 4.5, which is then used in Section 4.6 to discuss some problem specific properties. The chapter is concluded in Section 4.7 with extensive computational results.

4.2. A Literature Review

We give a selective literature review focusing on a survey and some recent publications with current benchmarks.

Literature related to the FJSS is mainly dedicated to the (non-flexible) Job Shop with Setup Times (JSS), or to the Flexible Job Shop (FJS) (without setup times). We first discuss selected papers dealing with the JSS, then we consider articles addressing the FJS, and finally, we point to some papers dealing with the FJSS.

The JSS has received increased attention over the last years as stated by Allahverdi in his survey on setup times [3]. Solution methods for this problem are generally based on methods that proved valuable in the JS. For example, branch and bound methods are applied by Brucker et al. [19] and Artigues and Feillet [6]. Vela et al. [112] develop a local search based on neighborhoods that generalize well-known structures of the JS. Balas et al. [9] adapt their famous shifting bottleneck procedure to incorporate sequence-dependent setup times.

The FJS is an established problem in the scheduling literature and various solution procedures have been developed for it. Brandimarte [12] decomposes the problem in a routing and a JS scheduling subproblem and solves these problems with a tabu search in a hierarchical manner. Hurink et al. [52] model the problem in a disjunctive graph and develop a tabu search where neighbors are generated by assigning a critical operation to another machine or moving an operation of a critical block before or after all other operations of that block. In contrast to Brandimarte, they simultaneously solve the machine assignment and operation sequencing problems. A similar approach is taken by Mastrolilli and Gambardella [79]. They use a neighborhood based on the extraction and re-insertion of an operation, and solve the re-insertion problem in an optimal or near optimal fashion. Gao et al. [34] develop a hybrid genetic algorithm with a variable neighborhood descent method where one or two operations are extracted and re-inserted. Hmida et al. [51] consider a discrepancy search using various neighborhoods that are based on rescheduling an operation. Finally, Kis [60] considers the job shop with a more general routing flexibility. He describes the feasible routings of a job as paths in a directed graph. A tabu search and a genetic algorithm are developed for solving the problem. Both methods are based on the insertion of a set of operations in a partial schedule and on the improvement of a schedule with fixed routings by standard methods of the JS.

Considering both features, sequence-dependent setup times and routing flexibility, appears to be a natural and interesting extension of the JSS and FJS. However, only few papers address the FJSS. Focacci et al. [32] treat a general scheduling problem with setup times and alternative machines, of which the FJSS is a special case. They propose a heuristic based on constraint programming. Rossi and Dini [101] formulate the FJSS problem in a disjunctive graph and solve it with an ant colony based heuristic. The main ingredient is a list scheduling algorithm for the generation of feasible solutions. Oddi et al. [85] develop a constraint programming based method using an iterative flattening search, and Gonzalez et al. [41] construct a neighborhood structure, which is used in a hybrid local search combining a tabu search with a genetic algorithm.

4.3. A Problem Formulation

The FJSS is a version of the JS with sequence-dependent setup times and routing flexibility.

Sequence-dependent setup times occur between two consecutive operations on a machine. If two operations i and j are executed on machine m, and j immediately follows i then a setup of duration $d^{\mathrm{s}}(i,m;j,m)$ occurs between the completion of i and the start of j. For each operation $i \in I$, an initial setup time $d^{\mathrm{s}}(\sigma;i,m)$ and a final setup time $d^{\mathrm{s}}(i,m;\tau)$ might be present. The final setup time is the minimum time elapsing between the completion time of i and the overall finish time (makespan).

The routing flexibility is of the following type. Any operation $i \in I$ needs one machine for its execution. However, this machine is not fixed, but can be chosen from a subset of alternative machines $M_i \subseteq M$. The processing duration $d^{\mathrm{p}}(i,m)$ of operation i can depend on the assigned machine $m \in M_i$.

A few standard assumptions concerning the data are made. All durations are non-negative, processing times $d^{\mathrm{p}}(i,m)$ are positive and the setup times satisfy the weak triangle inequality, i.e. for any operations i,j,k on a common machine m, $d^{\mathrm{s}}(i,m;j,m)+d^{\mathrm{p}}(j,m)+d^{\mathrm{s}}(j,m;k,m) \geq d^{\mathrm{s}}(i,m;k,m)$.

The FJSS can then be stated as follows. A schedule consists of an assignment of a machine and a starting time for each operation so that all constraints described in the JS with the additional features given above are satisfied. The objective is to find a schedule with minimal makespan.

4.4. The FJSS as an Instance of the CJS Model

A FJSS instance can be specified as an instance in the CJS model by applying the following transformations.

In the FJSS it is assumed that an unlimited number of buffers is available, hence there is always a buffer available to store a job after one of its operations is completed.

Buffers need to be specified in the CJS model. A buffer b_J is introduced for each job J, and the set M of machines in the CJS instance consists of the machines specified in the FJSS instance together with the introduced set of buffers $\{b_J : J \in \mathcal{J}\}$. A storage operation of duration 0 executed on buffer b_J is introduced between any two consecutive operations of job J. Thus, a job in the CJS instance consists of the operations in the FJSS instance together with the introduced storage operations.

The processing durations as well as the setup times between two consecutive operations on a machine are directly taken from the FJSS instance, and all setup times on the buffers are set to 0. For all $o \in \{o_i, \overline{o}_i\}, i \in I, m \in M_i$, the initial setup time $d^{\mathrm{s}}(\sigma;o,m) := d^{\mathrm{s}}(\sigma;i,m)$, the final setup time $d^{\mathrm{s}}(o,m;\tau) := d^{\mathrm{s}}(i,m;\tau)$. For all $i,j \in I$ and $m \in M_i, m' \in M_j$, load duration $d^{\mathrm{ld}}(i,m)$, unload duration $d^{\mathrm{ul}}(i,m)$ and transfer duration $d^{\mathrm{t}}(i,m;j,m')$ are 0, and the maximum time lag $d^{\mathrm{lg}}(i,m)$ is ∞ (not present).

We illustrate the disjunctive graph G of the FJSS problem formulated in the CJS model by considering the job shop example introduced in Section 1.3 with the routing flexibility given in Section 1.4.5 where the machines one and three are duplicated.

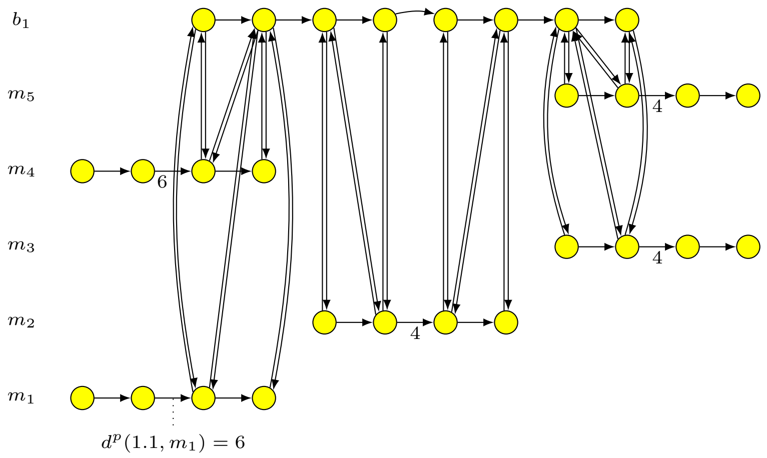


Figure 4.1.: Job 1 of the example. The machining arc weights are given. Nodes σ and τ are omitted for clarity.

The processing durations are assumed to be independent of the used machines, i.e. $d^{\mathrm{p}}(i, m) = d^{\mathrm{p}}(i, m')$. All setup times are set to 0. Figure 4.1 depicts job 1 of the example.

Since the FJSS can be formulated as a CJS problem without time lags, the JIBLS can be applied.

4.5. A Compact Disjunctive Graph Formulation

Clearly, the formulation of the FJSS in the CJS model is not compact. An operation does not need to be detailed with its take-over, processing and hand-over steps as in the CJS model.

A more compact disjunctive graph formulation of the FJSS is readily obtained starting with the disjunctive graph formulation of the JS (see Section 2.2) and using the modes as introduced in the CJS model (see Definition 3).

For each operation and each alternative machine, one node is introduced. Two consecutive operations i, j in some job are linked by a processing arc. Any two operations of the same job on a common machine are linked by a setup arc, and a pair of disjunctive arcs is introduced between any pair of operations from distinct jobs on a same machine.

Specifically, the disjunctive graph $G' = (V', A', E', \mathcal{E}', d')$ for the FJSS is constructed as follows. For each operation i and machine $m \in M_i$, a node v_{im} is introduced. Node set V' of G' consists of the v_{im}'s together with two additional nodes σ and τ representing fictive start and end operations of duration 0 occurring before, respectively after, all other operations, i.e. $V' = \cup\{v_{im} : i \in I, m \in M_i; \{\sigma, \tau\}\}$.

The set of conjunctive arcs A' consists of the following arcs: (i) For each operation $i \in I$ and machine $m \in M_i$, an initial setup arc (σ, v_{im}) of weight $d^{\mathrm{s}}(\sigma; i, m)$ and a final setup arc (v_{im}, τ) of weight $d^{\mathrm{p}}(i,m) + d^{\mathrm{s}}(i,m;\tau)$. (ii) For any two consecutive operations i and j of a job J and machine $m \in M_i$, $m' \in M_j$, a processing arc $(v_{im}, v_{jm'})$ of weight $d^{\mathrm{p}}(i,m)$. (iii) For any operations $i = J_r$ and $j = J_s$ of a job J with $1 \le r < s \le |J|$ and machine $m \in M_i \cap M_j$, a setup arc (v_{im}, v_{jm}) of weight $d^{\mathrm{p}}(i,m) + d^{\mathrm{s}}(i,m;j,m)$.

The set of disjunctive arcs E' consist of the following arcs. For any two operations $i, j \in I$ of distinct jobs and machine $m \in M_i \cap M_j$, two disjunctive arcs $(v_{im}, v_{jm}), (v_{jm}, v_{im})$ with respective weights $d^{\mathrm{p}}(i,m) + d^{\mathrm{s}}(i,m;j,m)$ and $d^{\mathrm{p}}(j,m) + d^{\mathrm{s}}(j,m;i,m)$.

The FJSS can be formulated as follows.

$$\boxed{\text{Among all feasible selections, find a selection } (\mu, S) \text{ minimizing the length of a longest path from } \sigma \text{ to } \tau \text{ in subgraph } G'(\mu,S) = (V'^{\mu}, A'^{\mu} \cup S, d').}$$

Two remarks are in order. First, while G' has less nodes than G, the number of disjunctive arc pairs is the same. In fact, it is easy to see that there is a one-to-one correspondence between the selections in G and G', and positive acyclic, complete and feasible selections in G and G' correspond. Second, any cycle in G' is a positive cycle (assuming positive processing times $d^{\mathrm{p}}(i,m)$ for all $i \in I$ and $m \in M_i$).

4.6. Specifics of the Solution Approach

Some properties specific to the FJSS and which will be taken into account in the JIBLS are now discussed. In order to simplify the proofs, we occasionally use the compact disjunctive graph G' instead of G, using in G' similar notation as in G.

4.6.1. The Closure Operator

We show that the closure operator Φ is simpler in FJSS instances than in other instances of the CJS model, e.g. a FBJSS (see Chapter 5).

Given some FJSS problem as an instance of the CJS model, let $G_i^m = (V_i^m, A_i^m, E_i^m, \mathcal{E}_i^m, d)$ be the job insertion graph of some operation $i \in I$ with i on machine $m \in M_i$ and $H^m = (E_i^m, U^m)$ its corresponding conflict graph.

Proposition 19 *In $H^m = (E_i^m, U^m)$, $e \to f \Leftrightarrow e \rightsquigarrow f$.*

Proof. We show transitivity of the relation $\to$, i.e. $e \to f$ and $f \to g$ implies $e \to g$.

Reasoning in the compact disjunctive graph G', $e \to f$ implies that $\{e, \overline{f}\}$ is positive cyclic, i.e. there is a positive cycle Z_1 in G' with $Z_1 \cap E' = \{e, \overline{f}\}$, and similarly $f \to g$ implies that there exists a positive cycle Z_2 in G' with $Z_2 \cap E' = \{f, \overline{g}\}$. Let P_1 be the path in Z_1 from $h(\overline{f})$ to $t(\overline{f})$ and P_2 the path in Z_2 from $h(f)$ to $t(f)$. P_1 contains e and P_2 contains $\overline{g}$. Since $h(f) = t(\overline{f})$ and $t(f) = h(\overline{f})$, $P_1 \cup P_2$ is a closed walk. Hence it contains a cycle Z which is positive since any cycle of G' is positive, $Z \cap E' = \{e, \overline{g}\}$ and therefore $e \to g$. ■

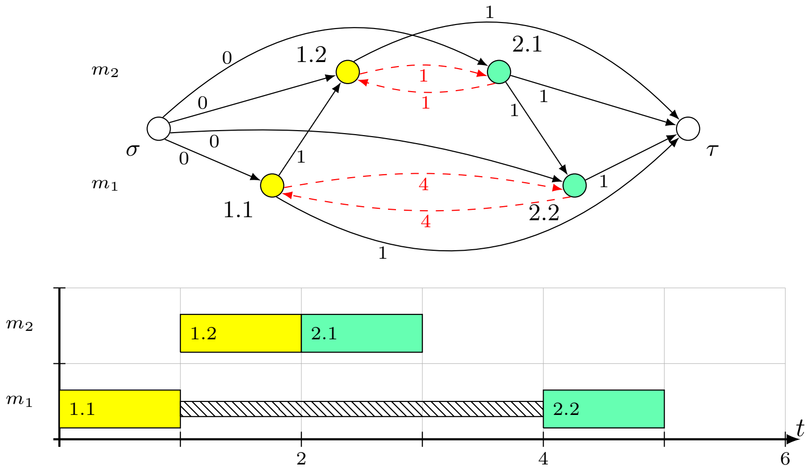


Figure 4.2.: The compact disjunctive graph of a small FJSS example and a schedule obtained by placing operation 1.1 before 2.2 and 1.2 before 2.1.

Using Proposition 19, we can rewrite the closure operator Φ (see Definition 14) in the FJSS as follows, allowing to speed up the computation of the closure.

$$\Phi(Q) = \{f \in E_i^m : e \rightsquigarrow f \text{ for some } e \in Q\} = \{f \in E_i^m : e \to f \text{ for some } e \in Q\}$$

4.6.2. Feasible Neighbors by Single Reversals

In some instances, replacing a critical arc $e \in S$ in selection (μ, S) by $\overline{e}$ according to (3.3) yields a neighbor selection $(\mu, S \cup \overline{e} - e)$ that is feasible. We call this single replacement of e by $\overline{e}$ a single reversal. We analyze cases in which a single reversal yields a feasible selection.

It is well-known that selection $(\mu, S \cup \overline{e} - e)$ obtained by a single reversal of a critical arc e is feasible in FJS instances (FJSS without setup times), see e.g. Balas [8]. However, this is not always the case in FJSS instances with setup times, as one can easily observe in the following small example.

Given are two jobs 1 and 2, both consisting of two operations (job 1: 1.1 and 1.2, job 2: 2.1 and 2.2) and two machines m_1, m_2. All operations have a processing time of 1. There is no routing flexibility, and operations 1.1 and 2.2 are executed on m_1, 1.2 and 2.1 on m_2. Setup times occur only on machine m_1 and are of duration 3 for each pair of operations on m_1. Initial and final setup times are 0. In Figure 4.2, the compact disjunctive graph G' of this problem is depicted together with a schedule where operation 1.1 is executed before 2.2 and 1.2 before 2.1.

Clearly, the disjunctive arc $e = (v_{1.1,m_1}, v_{2.2,m_1})$ is a critical arc in the illustrated solution. However, a single reversal of e forcing operation 2.2 to be executed before 1.1 on m_1 is not feasible. Indeed, a positive cycle of length 7 composed of the four arcs $\overline{e} = (v_{2.2,m_1}, v_{1.1,m_1})$, $(v_{1.1,m_1}, v_{1.2,m_2})$, $(v_{1.2,m_2}, v_{2.1,m_2})$ and $(v_{2.1,m_2}, v_{2.2,m_1})$ is obtained.

The example shows that in the presence of arbitrary setup times, a single reversal might yield an infeasible selection. Nevertheless, for some “structured” setup times, this does not occur.

Consider the following setup times called *setup times based on job pairs*. For all ordered pairs of jobs $(J, K), J, K \in \mathcal{J}$, a setup time s_{JK} is defined. It is assumed that $s_{JJ} = 0$, and the triangle inequality is satisfied, i.e. for any triple of distinct jobs J, K, L, the inequality $s_{JK}+s_{KL} \geq s_{JL}$ holds. Between two consecutive operations i, j on some machine m, i from job J and j from job K, a setup of duration $d^{\mathrm{s}}(i, m; j, m) = s_{JK}$ occurs.

Setup times based on job pairs are often used in benchmark instances for the FJSS (with and without flexibility). For example Brucker et al. [19] generated JSS instances with setup times based on job pairs. Their instances are then used by various authors, e.g. by Balas et al. [9], Vela et al. [112] and Artigues and Feillet [6]. Recently, Oddi et al. [86] generated FJSS instances by taking the setup times of Brucker et al.

Single reversals of critical arcs are always feasible in FJSS instances with setup times based on job pairs. Consider the compact disjunctive graph G' of some FJSS instance with setup times based on job pairs. Let (μ, S) be some feasible selection with makespan ω and $e \in S$ a critical arc from operations i of job J to operation j of job $K \neq J$, i.e. $e = (v_{i,\mu(i)}, v_{j,\mu(j)})$ is on a longest path L from σ to τ of length ω in $G'(\mu, S) = (V'^{\mu}, A'^{\mu} \cup S, d')$.

Proposition 20 *Selection (μ, S') with $S' = S - e \cup \overline{e}$ is feasible in FJSS instances with setup times based on job pairs.*

Proof. Selection (μ, S') is clearly complete. (μ, S') is also positive acyclic. Assume the contrary, i.e. there is a positive cycle Z' in $(V'^{\mu}, A'^{\mu} \cup S', d')$. By the SCP, there exists a positive cycle Z such that $Z \cap S' \subseteq Z' \cap S'$ and Z visits each job at most once. Moreover, since S is feasible, $(V'^{\mu}, A'^{\mu} \cup S - e, d')$ contains no positive cycle, so Z must contain $\overline{e} = (v_{j,\mu(j)}, v_{i,\mu(i)})$.

Let $P \subseteq S - \overline{e}$ be the path in Z from node $v_{i,\mu(i)}$ of operation i to node $v_{j,\mu(j)}$ of operation j. Path P starts at node $v_{i,\mu(i)}$ in job $J^1 = J$, eventually leaves job J, traverses (possibly) intermediary jobs $J^2, \ldots, J^{p-1}$ and enters finally job $J^p = K$, until it reaches node $v_{j,\mu(j)}$. As $e \notin Z$, also $e \notin P$, so P does not go directly from operation i to j, hence it visits after operation i another operation, say $k \neq j$.

The length $d(P)$ of path P clearly satisfies

$$d(P) \geq d^{\mathrm{p}}(i, \mu(i)) + d^{\mathrm{p}}(k, \mu(k)) + \sum_{q=1}^{p-1} s_{J^q, J^{q+1}}.$$

By the setup triangle inequalities and as the processing times are positive, $d(P) > d^{\mathrm{p}}(i, \mu(i)) + s_{JK}$. Then, there exists a walk from σ to τ in $G'(\mu, S)$ obtained from L by substituting arc e by path P, and the length of the walk is $\omega + d(P) - d(e) = \omega + d(P) - (d^{\mathrm{p}}(i, \mu(i)) + s_{JK}) > \omega$. Since (μ, S) is positive acyclic, this walk is or contains a path from σ to τ with length greater than ω, a contradiction.

We remark that a similar proof is possible without invoking the SCP. ■

As a consequence of Proposition 20, non-flexible neighbors can be built by single reversals of critical arcs in FJSS instances with setup times based on job pairs.

4.6.3. Critical Blocks

Consider the compact disjunctive graph G' of some FJSS instance, and let L be a longest path from σ to τ in graph $G'(\mu, S)$ of some selection (μ, S), and consider its critical operations. A (setwise) maximal sequence $i_1, \ldots, i_k$ of at least two consecutive critical operations using the same machine is called *a critical block* (cf. Brucker and Knust [17], p. 245).

It is well-known that swaps of two operations in the inner part of a critical block give no better neighbor if the FJSS instance has no setup times. Specifically, let $i_1, \ldots, i_k$ be a critical block of length $k > 3$. Then, neighbors $(\mu, S - e \cup \overline{e})$ with $e = (v_{i_r,m}, v_{i_{r+1},m}), r \in \{2, \ldots, k-2\}$, (obtained by single reversals) have a makespan that is no shorter than the makespan of the current selection (μ, S) (see e.g. Brucker et al. [17], p. 246).

In FJS instances, we may not build a neighbor by reversing a critical arc, but generate more promising moves, for example by moving an operation that is in the inner part of a critical block either before or after this block. We call these moves *block moves*.

Specifically, for each operation $i_r, 1 < r \leq k$, of a critical block $i_1, \ldots, i_k$ on machine m, construct a neighbor insertion (m, T_f) in job insertion graph $G^m_{i_r}$ according to (3.3) with arc $f = (v_{i_r,m}, v_{i_1,m})$ placing operation i_r before the first operation i_1 of the block, and similarly, for each operation $i_r, 1 \leq r < k$, of a critical block $i_1, \ldots, i_k$, construct a neighbor insertion (m, T_f) in job insertion graph $G^m_{i_r}$ according to (3.3) with arc $f = (v_{i_k,m}, v_{i_r,m})$ placing operation i_r after the last operation i_k of the block. Clearly, insertions (m, T_f) (and thus the corresponding selections) are feasible according to Theorem 17.

Based on the block moves and neighborhood $\mathcal{N}$, we build two new neighborhoods $\mathcal{N}^1$ and $\mathcal{N}^2$. Neighborhood $\mathcal{N}^1$ is obtained by taking the block moves together with the flexible moves of $\mathcal{N}$, and neighborhood $\mathcal{N}^2$ is obtained by combining the block moves with $\mathcal{N}$. So $\mathcal{N}^1 \subseteq \mathcal{N}^2$ and $\mathcal{N}^2 - \mathcal{N}^1$ consists of the non-flexible moves in the inner part of a critical block.

Two remarks are in order. First, the idea of moving an operation to the beginning or end of its critical block has been used by several authors (see Brucker and Knust [17]). In contrast to the job insertion based approach used here that always generates feasible neighbors, they fix the sequences of *all* other operations, resulting in neighbors that are sometimes infeasible. Second, although neighborhoods $\mathcal{N}^1$ and $\mathcal{N}^2$ are inspired by properties of instances without setup times, we tested them also in instances with setup times.

4.7. Computational Results

In this section, we present the results obtained by the JIBLS in FJSS and FJS instances, and compare them to the state of the art, which is to our best knowledge Oddi et al. [85] and Gonzalez et al. [41] in the FJSS and Mastrolilli and Gambardella [79], Gao et al. [34] and Hmida et al. [51] in the FJS. The obtained results turn out to be competitive with the state of the art in the FJSS instances and similar in the simpler FJS instances.

The JIBLS described in Section 3.5 was implemented single-threaded in Java for the FJSS in three versions using neighborhood $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$, respectively. It was run on a PC with 3.1 GHz Intel Core i5-2400 processor and 4 GB memory.

Extensive computational experiments on a set of 59 benchmark instances were performed for the FJSS with and without setup times. We used the instances of Oddi et al. [85] (with setup times), Barnes and Chambers [23] (without setup times) and Dauzère-Pérès and Paulli [29] (without setup times).

The computational settings were set as follows. The initial solution was a permutation schedule obtained by randomly choosing a job permutation and a mode. For each instance and neighborhood, five independent runs with different initial solutions were performed. The computation time of a run was limited to 600 seconds and the tabu search parameters were set as $maxt = 14$, $maxl = 300$ and $maxIter = 6000$.

We first report on the instances with setup times, starting with a comparison of the three neighborhoods $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ in the instances of Oddi et al. Table 4.1 shows the detailed numerical results as follows. The first block (columns 2-4) and second block (columns 5-7) depicts the best and average results over the five runs, respectively. The instances are grouped according to size, e.g. the first group 10×5 consists of instances with 10 jobs and 5 machines. The best values of each block are highlighted in boldface. The following observations can be made.

Comparing the best results (columns 2-4) over the five runs, neighborhoods $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ give the best values in 13, 10 and 19 instances (out of 20), and comparing the average results (columns 5-7), $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ present in 10, 5 and 19 instances the best values. In the instances of size 10×5 and 10×10 the results of the three neighborhoods are quite the same, and in the instances of size 15×5 and 20×5 the respective results of neighborhoods $\mathcal{N}$ and $\mathcal{N}^1$ are on average 0.6% and 2.6% worse than the results of neighborhood $\mathcal{N}^2$. Altogether, these numbers suggest that neighborhood $\mathcal{N}^2$ should give preference over $\mathcal{N}$ and $\mathcal{N}^1$.

We now compare the obtained results with neighborhood $\mathcal{N}^2$ to the best current benchmarks for the FJSS to our knowledge, namely the benchmarks of Oddi et al. [85] and Gonzalez et al. [41]. Table 4.2 displays for each instance the best and average results over the five runs, together with benchmarks of Oddi et al. and Gonzalez et al. We tried to use similar computation times as reported by these authors. Therefore, we divided Table 4.2 into three blocks. The first block (columns 2-3) reports the average results after 20 seconds (*avg-20*) and average results over 10 runs of Gonzalez et al. with their method GA+TS (*GVV1*). In the second block (columns 4-5), the average results after 600 seconds (*avg-600*) are compared with the benchmarks of Oddi et

	best			*avg*		
	$\mathcal{N}$	$\mathcal{N}^1$	$\mathcal{N}^2$	$\mathcal{N}$	$\mathcal{N}^1$	$\mathcal{N}^2$
10×5						
la01	**721**	726	**721**	**721.0**	727.2	**721.0**
la02	**737**	**737**	**737**	**737.0**	**737.0**	**737.0**
la03	**652**	**652**	**652**	**652.0**	**652.0**	**652.0**
la04	**673**	**673**	**673**	**673.0**	**673.0**	**673.0**
la05	**602**	**602**	**602**	602.8	602.8	**602.2**
15×5						
la06	947	954	**943**	951.6	961.0	**947.2**
la07	908	917	**904**	912.6	920.2	**906.6**
la08	940	**937**	940	**940.0**	963.8	**940.0**
la09	**986**	1001	**986**	990.0	1008.2	**986.0**
la10	**952**	970	**952**	**952.2**	977.2	954.4
20×5						
la11	1237	1263	**1228**	1246.8	1269.4	**1233.8**
la12	1079	1108	**1072**	1085.0	1118.6	**1072.8**
la13	**1172**	1202	**1172**	1184.0	1213.2	**1173.2**
la14	1232	1264	**1221**	1244.8	1274.0	**1239.8**
la15	1257	1288	**1256**	1274.4	1294.0	**1260.0**
10×10						
la16	**1007**	**1007**	**1007**	**1007.0**	1007.6	**1007.0**
la17	**851**	**851**	**851**	**851.0**	**851.0**	**851.0**
la18	**985**	**985**	**985**	**988.2**	991.4	**988.2**
la19	**951**	**951**	**951**	953.0	955.2	**952.0**
la20	**997**	**997**	**997**	**997.0**	**997.0**	**997.0**

Table 4.1.: Best and average results over the five runs with neighborhoods $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ in the FJSS instances (bold: best).

	avg-20	*GVV1*	*avg-600*	*ORCS1*	*best-600*	*GVV2*	*ORCS2*
10 × 5							
la01	725.0	**724**	**721.0**	736	**721**	**721**	726
la02	741.8	**737**	**737.0**	749	**737**	**737**	749
la03	653.2	**652**	**652.0**	658	**652**	**652**	**652**
la04	675.8	**675**	**673.0**	686	**673**	**673**	**673**
la05	603.0	**602**	**602.2**	603	**602**	**602**	603
15 × 5							
la06	**954.2**	957	**947.2**	963	**943**	953	950
la07	915.6	**911**	**906.6**	966	**904**	905	916
la08	946.2	**941**	**940.0**	963	**940**	**940**	948
la09	**990.4**	995	**986.0**	1020	**986**	989	1002
la10	959.0	**956**	**954.4**	991	**952**	956	977
20 × 5							
la11	**1246.4**	1254	**1233.8**	1257	**1228**	1244	1256
la12	**1077.4**	1107	**1072.8**	1097	**1072**	1098	1082
la13	**1186.0**	1212	**1173.2**	1240	**1172**	1205	1215
la14	**1258.6**	1263	**1239.8**	1285	**1221**	1257	1285
la15	**1278.0**	1282	**1260.0**	1291	**1256**	1275	1291
10 × 10							
la16	1010.6	**1007**	**1007.0**	1012	**1007**	**1007**	**1007**
la17	**851.0**	**851**	**851.0**	868	**851**	**851**	858
la18	996.2	**992**	**988.2**	1025	**985**	**985**	**985**
la19	955.6	**951**	**952.0**	976	**951**	**951**	956
la20	999.2	**997**	**997.0**	1033	**997**	**997**	**997**

Table 4.2.: The results *avg-20, avg-600, best-600* compared to benchmarks *GVV1, ORCS1, GVV2, ORCS2* in the FJSS instances (bold: best).

al. (*ORCS1*) obtained by their slack-based selection procedure and parameter value $\gamma = 0.3$. The third block (columns 6-8) depicts the best results after 600 seconds (*best-600*) and the best benchmarks reported by Gonzales et al. (*GVV2*) and Oddi et al. (*ORCS2*). The following observations can be made.

Results *avg-20* are similar to *GVV1*. On average *avg-20* is 0.1% better than *GVV1*, and most results are quite similar. Indeed, the relative deviation (*avg-20* − *GVV1*)/*GVV1* is in 18 of 20 instances within a range of +/- 0.7%. The exceptions are instances *la11* and *la12* where *avg-20* is 2.7% and 2.1% better than *GVV1*. Results *avg-600* systematically dominate *ORCS1*. On average *avg-600* is 2.6% better than *ORCS1*. Finally, observing results *best-600*, it improves the best results of *GVV2* and *ORCS2* in 9 instances and matches it in the other 11 instances. Altogether, our approach appears competitive in the FJSS when compared to the best current benchmarks.

For the evaluation of the tabu search, it is also of interest to examine the evolution of attained solution quality during computation. For this purpose, we recorded for each instance and run the current best makespan ω at the beginning (initial solution) of the tabu search and after 10, 20, 60, 120 and 300 seconds of computation time, and calculated its relative deviation from the final makespan $(\omega - \omega_{\text{final}})/\omega_{\text{final}}$. Table 4.3 provides these deviations (in %) in columns 4 to 8 in an aggregated way, reporting average deviations over runs and instances of the same size. Additionally, the average

size	*iter*	*time*	*0 s*	*10 s*	*20 s*	*60 s*	*120 s*	*300 s*
10 x 5	3'535'915	374	226.0%	0.6%	0.4%	0.2%	0.0%	0.0%
15 x 5	2'940'463	600	258.6%	1.0%	0.7%	0.1%	0.1%	0.0%
20 x 5	1'724'258	600	265.3%	1.6%	1.1%	0.4%	0.2%	0.0%
10 x 10	4'317'167	600	359.7%	0.8%	0.4%	0.2%	0.1%	0.1%

Table 4.3.: Average number of tabu search iterations, average runtime and relative deviations of the makespan from the final makespan during runtime in the FJSS instances.

number of tabu search iterations and the runtime are displayed in columns 2-3. The following can be observed.

Initial solutions are far away from the obtained final solutions, with makespans three to five times as large. Also, most of the improvements are found within seconds. Consider for example the 10×10 instances. While the deviation is initially 359.7%, it drops to 0.8% and 0.4% after 10 and 20 seconds. This improvement behavior can be partly attributed to the rather high number of tabu search iterations; they are in the range of 1.5 and 4.5 million. Altogether, these figures suggest good improvement performance and convergence of the tabu search.

We now consider the instances without setup times. First, we compare the three versions $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ with each other in the instances of Dauzère-Pérès and Paulli. Table 4.4 provides detailed results as follows. The first block (columns 2-4) and second block (columns 5-7) depicts the best and average results over the five runs, respectively. The instances are grouped according to size. The best values of each block are highlighted in boldface. The following observations can be made.

Comparing the best results (columns 2-4) over the five runs, neighborhoods $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ give the best values in 0, 17 and 2 instances (out of 20), and comparing the average results (columns 5-7), $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ present in 0, 17 and 1 instances the best values. On average, the respective results of neighborhoods $\mathcal{N}$ and $\mathcal{N}^2$ are 1.6% and 1.0% worse than the results of neighborhood $\mathcal{N}^1$.

While $\mathcal{N}^2$ appears to be the best neighborhood in the instances with setup times, neighborhood $\mathcal{N}^1$ should give preference in instances without setup times. This result is not surprising having in mind that the difference of the two neighborhoods are the swaps of operations in the inner part of a critical block that are present in $\mathcal{N}^2$ but not in $\mathcal{N}^1$. It is well-known (and was discussed before) that these moves result in a neighbor having no better makespan than the current solution in the FJS instances. We observed that the number of cycling situations in the tabu search version with neighborhoods $\mathcal{N}^2$ is much higher than in the version with $\mathcal{N}^1$.

We now compare the results obtained with neighborhood $\mathcal{N}^1$ to the best current benchmarks to our knowledge, namely the benchmarks of Mastrolilli and Gambardella [79], Gao et al. [34] and Hmida et al. [51] in the instances of Barnes and Chambers (denoted by BCdata) and Dauzère-Pérès and Paulli (denoted by DPdata). We tried to use similar computation times as reported by the authors of the benchmarks. Therefore, we compare the average results obtained after 20 seconds in the BCdata

	best			*avg*		
	$\mathcal{N}$	$\mathcal{N}^1$	$\mathcal{N}^2$	$\mathcal{N}$	$\mathcal{N}^1$	$\mathcal{N}^2$
10×5						
01a	2570	**2505**	2554	2628.4	**2516.6**	2606.6
02a	2243	**2234**	2235	2254.6	**2238.4**	2244.0
03a	2233	**2230**	2232	2240.8	**2232.8**	2234.4
04a	2581	**2503**	2529	2656.6	**2506.2**	2589.2
05a	2221	**2218**	2225	2236.0	**2220.0**	2241.8
06a	2217	2206	**2204**	2222.8	**2208.8**	2217.0
15×8						
07a	2390	**2288**	2385	2461.2	**2317.0**	2780.6
08a	2081	**2074**	2081	2098.2	**2081.4**	2090.2
09a	2071	**2068**	2075	2084.8	**2074.0**	2081.8
10a	2374	**2266**	2373	2617.2	**2313.8**	2393.0
11a	2080	**2073**	2075	2083.2	2078.4	**2078.2**
12a	2048	**2039**	**2039**	2064.4	**2043.8**	2052.6
20×10						
13a	2361	**2262**	2305	2431.0	**2288.4**	2327.4
14a	2196	**2173**	2184	2212.8	**2186.0**	2224.4
15a	2194	**2174**	2179	2217.2	**2178.0**	2185.2
16a	2339	**2262**	2313	2405.0	**2276.4**	2335.8
17a	2175	**2148**	2155	2189.4	**2158.6**	2171.4
18a	2167	**2142**	2149	2180.2	**2150.0**	2156.8

Table 4.4.: Best and average results over the five runs with neighborhoods $\mathcal{N}$, $\mathcal{N}^1$ and $\mathcal{N}^2$ in the FJS instances (bold: best).

and after 200 seconds in the DPdata with the benchmark results. We also report results obtained at termination of the tabu search (after 600 seconds). Table 4.5 displays the detailed results as follows. Instances of BCdata and DPdata are reported in the upper and lower part, respectively. For each instance, the average results after 20 seconds (*avg-20* in BCdata), 200 seconds (*avg-200* in DPdata), and 600 seconds (*avg-600*), and best results after 600 seconds (*best*) over the five runs are reported. The average and best results are compared to the average and best results of Mastrolilli and Gambardella (*MG1*, *MG2*), Gao et al. (*GS1*, *GS2*) and Hmida et al. (*HH1*, *HH2*). These authors report average and best results over 4 to 5 runs. The following observations can be made.

In both data sets, the results of the different methods are of similar quality. Indeed, the relative deviation between the results is mostly within a range of +/- 1.0%. Now considering results *avg-20*, they are slightly worse than *MG1*, *GS1* and *HH1*. The relative deviation of *avg-20* to *MG1*, *GS1* and *HH1*, calculated as (*avg-20*− *bench*)/*bench* where *bench* stands for the benchmark value, is 0.37%, 0.35% and 0.40% in BDdata, and 0.61%, 0.65% and 0.65% in DPdata. If we allow 600 seconds of computation time (see column *avg-600*), these deviations are -0.06%, -0.08% and -0.03% in the BCdata, and 0.37%, 0.42% and 0.42% in the DPdata. While the results *avg-600* appear to be slightly better than the others in BCdata, *GS1* seem to be slightly better than the others in DPdata, but all differences are quite small.

We now compare the *best* results with benchmarks *MG2*, *GS2* and *HH2*. Results *best*, *MG2*, *GS2* and *HH2* give best values (among them) in 18, 14, 11, 13 instances

BCdata	*avg-20*	*avg-600*	*MG1*	*GS1*	*HH1*	*best*	*MG2*	*GS2*	*HH2*
mt10c1	933.2	927.4	928.0	**927.2**	928.5	**927**	928	**927**	928
mt10cc	911.6	**909.2**	910.0	910.0	910.8	**908**	910	910	910
mt10x	925.2	918.8	**918.0**	**918.0**	**918.0**	**918**	**918**	**918**	**918**
mt10xx	924.6	**918.0**	**918.0**	**918.0**	**918.0**	**918**	**918**	**918**	**918**
mt10xxx	919.6	**918.0**	**918.0**	**918.0**	**918.0**	**918**	**918**	**918**	**918**
mt10xy	907.6	**905.0**	906.0	**905.0**	906.0	**905**	906	**905**	906
mt10xyz	851.6	**847.8**	850.8	849.0	850.5	**847**	**847**	849	849
setb4c9	923.8	918.6	919.2	**914.0**	919.0	917	919	**914**	919
setb4cc	909.4	**907.0**	911.6	914.0	910.5	**907**	909	914	909
setb4x	926.4	**925.0**	**925.0**	931.0	**925.0**	**925**	**925**	**925**	**925**
setb4xx	926.4	**925.0**	926.4	**925.0**	**925.0**	**925**	**925**	**925**	**925**
setb4xxx	**925.0**	**925.0**	**925.0**	**925.0**	**925.0**	**925**	**925**	**925**	**925**
setb4xy	916.0	**911.8**	916.0	916.0	916.0	**910**	916	916	916
setb4xyz	909.4	**905.0**	908.2	**905.0**	906.5	**905**	**905**	**905**	**905**
seti5c12	1184.8	**1174.2**	**1174.2**	1175.0	1174.5	**1174**	**1174**	1175	**1174**
seti5cc	1142.0	**1136.4**	**1136.4**	1138.0	1137.0	**1136**	**1136**	1138	**1136**
seti5x	1212.0	1204.2	1203.6	1204.0	**1201.5**	1204	**1201**	1204	**1201**
seti5xx	1204.8	1202.6	1200.6	1203.0	**1199.0**	**1198**	1199	1202	1199
seti5xxx	1207.4	1201.2	1198.4	1204.0	**1197.5**	**1197**	**1197**	1204	**1197**
seti5xy	1142.0	**1136.4**	**1136.4**	1136.5	1138.0	**1136**	**1136**	**1136**	**1136**
seti5xyz	1135.8	1130.2	1126.6	1126.0	**1125.3**	1128	**1125**	1126	**1125**

DPdata	*avg-200*	*avg-600*	*MG1*	*GS1*	*HH1*	*best*	*MG2*	*GS2*	*HH2*
01a	2518.6	**2516.6**	2528.0	2518.0	2524.5	**2505**	2518	2518	2518
02a	2241.4	2238.4	2234.0	**2231.0**	2234.5	2234	**2231**	**2231**	**2231**
03a	2233.6	2232.8	2229.6	**2229.3**	2231.8	2230	**2229**	**2229**	**2229**
04a	2511.2	**2506.2**	2516.2	2518.0	2510.0	**2503**	**2503**	2515	**2503**
05a	2221.4	2220.0	2220.0	**2218.0**	**2218.0**	2218	**2216**	2217	**2216**
06a	2211.2	2208.8	2206.4	**2198.0**	2202.5	2206	2203	**2196**	**2196**
07a	2325.4	2317.0	2297.6	2309.8	**2295.5**	2288	**2283**	2307	**2283**
08a	2083.6	2081.4	2071.4	2076.0	**2069.0**	2074	**2069**	2073	**2069**
09a	2076.2	2074.0	2067.4	2067.0	**2066.8**	2068	**2066**	**2066**	**2066**
10a	2322.8	2313.8	2305.6	2315.2	**2302.5**	**2266**	2291	2315	2291
11a	2083.0	2078.4	**2065.6**	2072.0	2072.0	2073	**2063**	2071	**2063**
12a	2048.8	2043.8	2038.0	**2031.6**	2034.0	2039	2034	**2030**	2031
13a	2295.0	2288.4	2266.2	**2260.0**	2260.3	2262	2260	**2257**	**2257**
14a	2187.8	2186.0	2168.0	**2167.6**	2178.8	2173	**2167**	**2167**	**2167**
15a	2180.6	2178.0	2167.2	**2165.4**	2170.3	2174	2167	**2165**	**2165**
16a	2294.8	2276.4	2258.8	2258.0	**2257.8**	2262	**2255**	2256	2256
17a	2170.8	2158.6	2144.0	**2142.0**	2145.5	2148	2141	**2140**	**2140**
18a	2155.2	2150.0	2140.2	**2130.7**	2131.5	2142	2137	**2127**	**2127**

Table 4.5.: The results *avg-20, avg-600, best* compared to benchmarks *MG1, GS1, HH1, MG2, GS2, MG2* (bold: best) in the FJS instances.

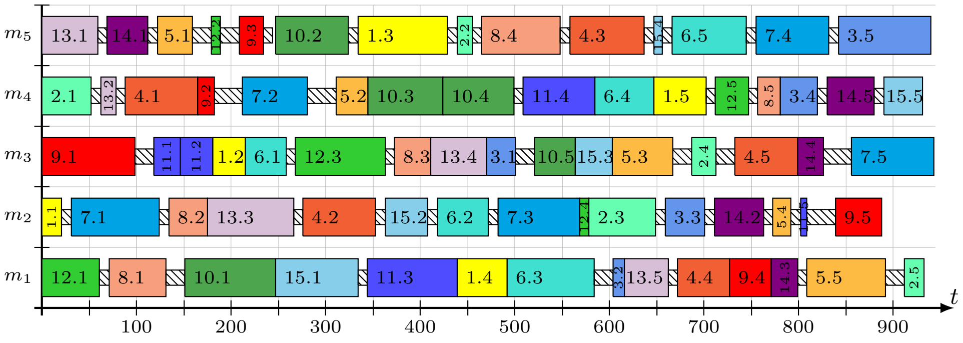


Figure 4.3.: A schedule with makespan 943 of FJSS instance *la06*.

of BCdata, and in 3,10,10,14 instances of DPdata. These numbers support the view that the different methods give quite similar results.

Altogether, the JIBLS appears to find good results compared to state of the art methods also in the instances without setup times.

We conclude the discussion of the computational results with two Gantt charts. Figure 4.3 depicts a schedule of instance *la06* (with setup times) having makespan 943, and Figure 4.4 presents a schedule of instances *mt10cc* (without setup times) having makespan 908. Thick bars represent the processing of the operations while the narrow hatched bars illustrate setup times. The numbers refer to the operations, e.g. 5.1 refers to the first operation of job 5.

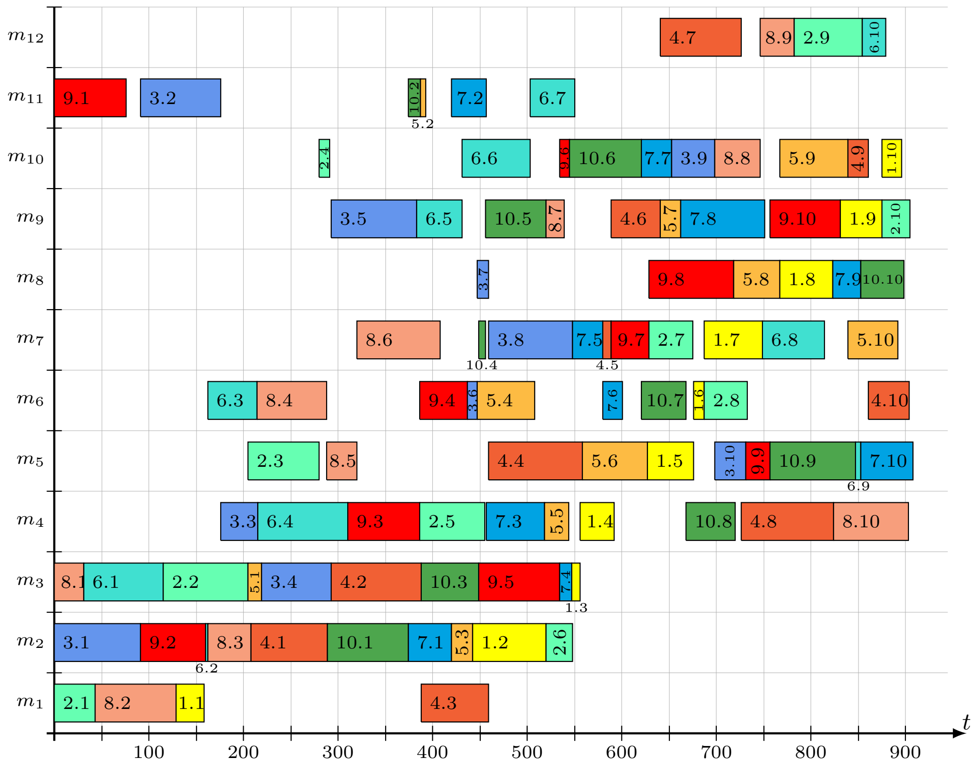


Figure 4.4.: A schedule with makespan 908 of FJS instance *mt10cc*.

CHAPTER 5

THE FLEXIBLE BLOCKING JOB SHOP WITH TRANSFER AND SETUP TIMES

5.1. Introduction

The Flexible Blocking Job Shop with Transfer and Setup Times (FBJSS) is a version of the Flexible Job Shop with Setup Times (FJSS) characterized by the absence of buffers. In practice, many systems have limited or no buffers and considering the buffer restrictions is important. Solutions that do not satisfy these constraints might not be implementable.

An additional feature of the FBJSS will be transfer times for passing a job from a machine to the next. This feature is useful in practice. Indeed, a job after completion of an operation on a machine has to be handed over to its next machine. Such transfer steps have to be taken into account if transfer times are not negligible. Transfers are also important when considering transportation of the jobs, allowing to model the transfer of a job from a mobile device to a machine, or vice versa. They take place at a fixed location and may imply additional constraints if the mobile devices interfere with each other in space (see Chapter 7). For further information, we point to Section 1.3 where the absence of buffers and the presence of transfer times are also discussed.

This chapter is organized as follows. A selective literature review is given in the following section. Section 5.3 gives a formal description of the FBJSS, which is used in Section 5.4 to formulate the problem in the CJS model. Extensive computational results are given in Section 5.5. We conclude this chapter in Section 5.6 with remarks on how limited buffers can be taken into account.

We remark that a good part of the material of this chapter is published [45].

5.2. A Literature Review

We give a selective literature review focusing on a survey and some recent publications with current benchmarks.

Literature related to the FBJSS is mainly dedicated to the non-flexible Blocking Job Shop (BJS) without transfer and setup times and to the Blocking Job Shop with Transfer and Setup Times (BJSS). Indeed, we are not aware of previous literature on the blocking job shop with flexibility (except our publication [45]). Blocking constraints together with flexibility have been addressed in the simpler flow shop version of the job shop by Thornton and Hunsucker [110].

The BJS has found increasing attention over the last years. Mascis et al. [76, 77] formulate several scheduling problems, among them the BJS, with the help of alternative graphs and solve them with dispatching heuristics. Meloni et al. [80] develop a "rollout" metaheuristic which they apply among other problems also to the BJS. Brizuela et al. [13] propose a genetic algorithm for the BJS. Brucker et al. [16] present tabu search algorithms for cyclic scheduling in the BJS. Van den Broek [111] proposes a heuristic for the BJS that successively inserts optimally a job. He formulates each job insertion as a mixed integer linear programming problem and solves it with CPLEX. He also develops an exact approach based on branch and bound, using his heuristic solution as an initial upper bound. Oddi et al. [87] develop two iterative improvement algorithms for the BJS based on flattening search and on the constraint programming solver of IBM ILOG. Pranzo and Pacciarelli [99] recently proposed a method based on an iterated greedy approach, which iteratively builds a partial solution starting from a feasible solution and reconstructs a solution in a greedy fashion.

The BJSS has received less attention than the BJS. Klinkert [62] studies the scheduling of pallet moves in automated high-density warehouses, proposes a generalized disjunctive graph framework similar to the alternative graphs and devises a local search heuristic with a feasible neighborhood for the BJSS. Gröflin and Klinkert [42] study a general insertion problem with, among others, an application to job insertion in the BJSS. They also present in [43] a tabu search for the BJSS.

Job shop scheduling problems with limited buffer capacity (extending in a sense the BJS where buffer capacity is zero) are studied by Brucker et al. [15] and Heitmann [49]. Various ways in which buffers occur are analyzed and tabu search approaches for flow shop and job shop problems are proposed for specific buffer configurations, including the BJS.

5.3. A Problem Formulation

The FBJSS is a version of the FJSS without buffers and with transfer times. The additional features can be described as follows.

The assumption that there are no buffers between machines has the following consequence. A job, after having finished an operation on some machine m, might have to wait on m, thus blocking m, until the machine for its next operation becomes available.

Specifically, consider operation $j = J_r$ of a job J and assume that j is processed on $m \in M_j$, its job predecessor operation $i = J_{r-1}$ is executed on $p \in M_i$ and its job successor $k = J_{r+1}$ is processed on $q \in M_k$. Operation j consists of the following four successive steps: (i) a take-over step of duration $d^{\mathrm{t}}(i,p;j,m)$ where, after the completion of operation i, the job is taken over from machine p to machine m; (ii) a processing step on machine m of duration $d^{\mathrm{p}}(i,m)$, (iii) a possible waiting time of the job on machine m of unknown duration; and (iv) a hand-over step of duration $d^{\mathrm{t}}(j,m;k,q)$ where job J is handed over from machine m to machine q for its next operation k.

The transfer times as well as the setup times can have value 0, allowing also for the case where so-called swapping is permitted. Swapping occurs when two or more jobs swap their machines.

Note that setups between two consecutive operations on a machine may occur between the end of an operation's hand-over step and the start of the next operation's take-over step.

The FBJSS problem can be stated as follows. A schedule consists of an assignment of a machine and a starting time of the hand-over, processing and take-over step for each operation so that the constraints described in the FJSS and given above are satisfied. The objective is to find a schedule with minimal makespan.

5.4. The FBJSS as an Instance of the CJS Model

The formulation of the FBJSS served as a basis for the development of the CJS model. Therefore, a FBJSS instance can be specified in a straightforward manner as an instance in the CJS model, the only remarks given here concern time lags and initial and final setup times.

Clearly, the maximum time lags $d^{\mathrm{lg}}(i,m)$ are set to ∞ (not present). An initial and a final setup time are given for each operation in the FBJSS while they can be specified for each transfer step in the CJS. We simply set $d^{\mathrm{s}}(\sigma;o_i,m)$ and $d^{\mathrm{s}}(\sigma;\bar{o}_i,m)$ to $d^{\mathrm{s}}(\sigma;i,m)$, and similarly, $d^{\mathrm{s}}(o_i,m;\tau)$ and $d^{\mathrm{s}}(\bar{o}_i,m;\tau)$ are set to $d^{\mathrm{s}}(i,m;\tau)$ for each operation $i \in I$ and $m \in M_i$.

Note that the Example introduced in Section 2.4.4 is a FBJSS instance formulated in the CJS model.

Clearly, the FBJSS belongs to the class of CJS problems without time lags, hence the JIBLS can be applied.

5.5. Computational Results

In this section, we present computational results obtained by the JIBLS and compare them to the state of the art, which is to our best knowledge Oddi et al. and Pranzo and Pacciarelli in the BJS, and our results published in [45] in the FBJSS. The obtained results turn out to be substantially better than the state of the art.

The JIBLS (with neighborhood $\mathcal{N}$) was implemented single-threaded in Java for the FBJSS. It was run on a PC with 3.1 GHz Intel Core i5-2400 processor and 4 GB memory.

We conducted extensive computational experiments on a set of FBJSS instances that were introduced in [45]. These instances were generated as follows. We started with the 40 instances la01 to la40 of Lawrence [64]. For each lapq, two groups of 3 instances were generated, the first group without transfer and setup times (referred to as Flexible Blocking Job Shop (FBJS) instances), the second group with transfer and setup times (referred to as FBJSS instances). Within a group, the 3 instances are generated by introducing increasing flexibility: in the first instance, 1 machine is available for each operation (i.e. the non-flexible instance), while in the 2nd and 3rd, 2 and 3 machines, respectively, are available for each operation. This is achieved by adding randomly and successively one machine, creating for each operation three machine sets of size 1, 2 and 3 that are nested. Such a choice will allow to evaluate the impact of increasing flexibility. Transfer and setup times were generated randomly as described in [43].

The computational settings were similar to those described in the FJSS (Chapter 4). The initial solution was a permutation schedule obtained by randomly choosing a job permutation and a mode. For each instance, five independent runs with different initial solutions were performed. The computation time of a run was limited to 1800 seconds and the tabu search parameters were set as $maxt = 10$, $maxl = 300$ and $maxIter = 6000$.

We compared the obtained results with the best current benchmarks to our knowledge, namely the results of Oddi et al. [87], Pranzo and Pacciarelli [99] (for non-flexible FBJS instances), and the results published in [45].

Table 5.1 displays detailed results for the non-flexible FBJS instances. We tried to use similar computation times as in the benchmarks. Therefore, we divided the table into three blocks. The first block (columns 2-3) displays the average results after 600 seconds (*avg-600*) and the benchmarks of Pranzo and Pacciarelli with configuration IG.RW. The seconds block (columns 4-6) reports the average results after 1800 seconds (*avg-1800*), the average benchmarks reported in [45] (*GPB*), the benchmarks of Oddi et al. obtained by their IFS method with the slack-based selection procedure and parameter $\gamma = 0.5$ (*OR1*) and by their method CP-OPT (*OR2*). In the third block (columns 7-9), the best results after 1800 seconds over the five runs (*best-1800*) are compared to the best benchmarks of Oddi et al. (*OR3*) over 32 runs of their IFS method and the best values of Pranzo and Pacciarelli over 12 runs of their iterated greedy approach. In every block, the best values are highlighted in boldface. The instances are grouped according to size, e.g. the first group 10×5 consists of instances with 10 jobs and 5 machines. The following observations can be made.

Results *avg-600* are competitive with *PP1*. Indeed, *avg-600* gives lower, equal and higher values than *PP1* in 33, 5 and 2 instances, respectively. Moreover, the relative deviation of *avg-600* to *PP1* (i.e. $(avg\text{-}600 - PP1)/PP1$) is $-4,2\%$ averaged over all

	avg-600	*PP1*	*avg-1800*	*GPB*	*OR1*	*OR2*	*best-1800*	*OR3*	*PP2*
10×5									
la01	**793.0**	**793**	**793.0**	820	**793**	**793**	**793**	**793**	**793**
la02	**793.0**	**793**	**793.0**	817	**793**	815	**793**	**793**	**793**
la03	**715.0**	**715**	**715.0**	740	740	790	**715**	**715**	**715**
la04	**743.0**	**743**	**743.0**	764	776	784	**743**	**743**	**743**
la05	**664.0**	**664**	**664.0**	666	**664**	**664**	**664**	**664**	**664**
15×5									
la06	**1083.8**	1102	**1077.8**	1180	1112	1131	1076	**1064**	1102
la07	**1042.4**	1062	**1033.4**	1084	1081	1106	**1016**	1038	1020
la08	**1058.6**	1089	**1053.8**	1162	1135	1129	**1040**	1062	1071
la09	**1154.4**	1192	**1148.2**	1258	1257	1267	**1141**	1185	1162
la10	**1117.2**	1140	**1116.0**	1208	1158	1168	**1096**	1110	1134
20×5									
la11	**1460.0**	1550	**1460.0**	1591	1501	1520	**1442**	1466	1498
la12	**1255.2**	1342	**1255.2**	1398	1321	1308	**1240**	1272	1271
la13	**1402.2**	1531	**1399.8**	1541	1471	1528	**1373**	1465	1482
la14	**1483.8**	1538	**1479.0**	1638	1567	1506	**1465**	1548	1513
la15	**1506.8**	1593	**1493.8**	1630	1547	1571	**1465**	1527	1517
10×10									
la16	1064.0	**1060**	**1060.0**	1143	1086	1150	**1060**	1084	**1060**
la17	**929.0**	930	**929.0**	977	1000	996	**929**	930	**929**
la18	**1029.4**	1040	**1025.4**	1098	1120	1135	**1025**	1026	**1025**
la19	1051.0	**1043**	**1045.0**	1102	1077	1108	**1043**	**1043**	**1043**
la20	**1062.8**	1080	**1062.8**	1162	1166	1119	**1060**	1074	**1060**
15×10									
la21	**1436.0**	1514	**1436.0**	1576	1521	1579	**1422**	1521	1490
la22	**1319.0**	1368	**1311.2**	1467	1490	1379	**1292**	1425	1339
la23	**1439.6**	1445	**1428.6**	1570	1538	1497	**1393**	1531	1445
la24	**1394.8**	1434	**1384.8**	1546	1498	1523	**1372**	1498	1434
la25	**1361.6**	1422	**1345.2**	1523	1424	1561	**1311**	1424	1392
20×10									
la26	**1887.2**	2013	**1854.8**	2125	2179	2035	**1795**	2045	1989
la27	**1939.8**	2044	**1923.8**	2201	2172	2155	**1892**	2104	2017
la28	**1911.0**	2039	**1871.4**	2167	2132	2062	**1836**	2027	2039
la29	**1780.8**	1928	**1754.2**	1990	1963	1898	**1733**	1963	1846
la30	**1898.2**	2137	**1895.4**	2097	2125	2147	**1868**	2095	2049
30×10									
la31	**2686.6**	3095	**2673.0**	3137	3771	2921	**2628**	3078	3018
la32	**2978.0**	3415	**2945.8**	3316	3852	3237	**2911**	3336	3338
la33	**2668.2**	2970	**2652.6**	3061	3741	2844	**2640**	3147	2909
la34	**2717.4**	3016	**2713.0**	3146	3796	2848	**2686**	3125	3016
la35	**2763.4**	3193	**2722.8**	3171	3818	2923	**2673**	3148	3133
15×15									
la36	**1706.0**	1755	**1699.4**	1919	1891	1952	**1643**	1793	1755
la37	**1853.8**	1870	**1812.2**	2029	1983	1952	**1774**	1983	1870
la38	**1625.6**	1728	**1624.4**	1828	1708	1880	**1616**	1708	1720
la39	**1699.6**	1731	**1682.0**	1882	1848	1813	**1651**	1783	1731
la40	**1693.4**	1743	**1662.4**	1925	1831	1928	**1629**	1777	1743

Table 5.1.: The results *avg-600*, *avg-1800* and *best-1800* compared to the benchmarks *PP1, PP2, GPB, OR1, OR2, OR3* in the FBJS instances with *flex*= 1 (bold: best).

instances, and -11.9% averaged over instances of size 30×10, showing that *avg-600* is particularly at an advantage in large instances. Results *avg-1800* dominate *GPB*, *OR1* and *OR2*. Indeed, in all 40 instances *avg-1800* provides the best values. Moreover, the relative deviation of *avg-1800* to the benchmarks is on average $-9.1\%, -9.1\%$ and -7.2 for *GPB*, *OR1* and *OR2*, respectively. These deviations are lower in small instances, e.g. $-2.5\%, -1.5\%$ and -3.5% in 10×5 instances, and higher in large instances, e.g. $-13.4\%, -27.8\%$ and -7.2% in 30×10 instances for *GPB*, *OR1* and *OR2*, respectively. Also, results *best-1800* are on average 6.0% and 4.6% better than *OR3* and *PP2*, respectively, and *best-1800* gives lower, equal and higher values than the best of *OR3* and *PP2* in 29, 10 and 1 instances, respectively.

These numbers suggest two findings. First, the methods of Pranzo and Pacciarelli, and Oddi et al. provide a reasonable solution quality for small instances, but are far away from the obtained results in all other instances. Second, the obtained results are much better than the benchmarks *GPB*, especially in large instances.

We now consider all other instances, i.e. the flexible FBJS instances and the FBJSS instances. Table 5.2 provides the detailed results for these instances as follows. The first line splits the results into the two groups FBJS and FBJSS. The second line refers to the degree of flexibility (*flex*$= 1, 2$ or 3), i.e. the number of alternative machines per operation, and the third line to the type of results (average (*avg*) or *best* result over the 5 runs per instance). The instances are grouped according to size. Note that results FBJS with *flex*$= 1$ are not present as they are listed in Table 5.1.

We now discuss the results of Table 5.2 by comparing them with the benchmarks *GPB* published in [45] (taking the results with neighborhood $\mathcal{N}_c^1$). The comparison of the obtained results with *GPB* is done in the following way. For each instance, we computed the relative deviation $(avg - GPB)/GPB$ of the average results *avg* to the average benchmarks *GPB*. Table 5.3 shows these deviations in an aggregated way, reporting average deviations over runs and instances of the same size. The table is split into two blocks. The FBJS instances are treated in the first block (columns 2-4), and the FBJSS are presented in the second block (columns 5-7). Each block consists of three columns, one for each degree of flexibility (*flex*$= 1, 2, 3$). In the first column the size of the instances are given. Note that the last line indicates average deviations over all instance sizes, and the deviations in the non-flexible FBJS instances are also provided in column 2. The following observations can be made.

Results *avg* are substantially better than *GPB*, particularly in large non-flexible instances. E.g. they are on average 13.4% and 7.8% lower in 30×10 instances of FBJS and FBJSS, respectively. But also in instances with flexibility, better results are found, e.g. with *flex*$= 2$ results *avg* are on average 9.6% and 3.7% lower than *GPB* in 30×10 instances of FBJS and FBJSS, respectively.

Although the results *GPB* are obtained by a similar approach as used here, namely a tabu search with a job insertion based neighborhood, several improvements have been implemented since then, two of them worth being mentioned. First, we consider the tail and head operation of critical arcs to build neighbors (see Section 3.4.3), while the publication [45] considers just head operations of critical arcs. Second, we substantially improved the time efficiency of the tabu search. Particularly, the efficiency of the closure computation and makespan calculation were refined. As a

group	*FBJS*				*FBJSS*					
flex	*2*		*3*		*1*		*2*		*3*	
	avg	*best*	*avg*	*best*	*avg*	*best*	*avg*	*best*	*avg*	*best*
10 × 5										
la01	642.6	641	609.0	607	1441.0	1441	1005.0	1005	849.2	849
la02	595.4	590	574.0	574	1515.0	1515	942.0	942	796.8	794
la03	541.0	541	507.8	506	1425.0	1423	925.6	923	737.6	733
la04	562.6	561	522.8	522	1391.8	1387	900.4	898	743.2	739
la05	549.0	549	549.0	549	1324.8	1304	842.0	842	733.2	731
15 × 5										
la06	900.4	895	841.2	837	2036.8	2024	1370.8	1346	1145.2	1129
la07	834.6	830	790.4	787	1999.8	1986	1339.2	1312	1075.0	1057
la08	858.4	851	804.2	795	2040.6	2017	1382.0	1352	1109.4	1092
la09	968.2	958	901.0	897	2124.0	2116	1521.2	1497	1171.0	1149
la10	914.4	895	851.8	844	2147.6	2130	1495.6	1477	1155.8	1144
20 × 5										
la11	1214.8	1204	1135.4	1122	2765.2	2740	1941.2	1888	1537.0	1509
la12	1058.2	1038	989.4	985	2577.8	2566	1771.2	1755	1406.8	1366
la13	1170.0	1152	1094.4	1089	2600.0	2570	1877.6	1849	1480.2	1450
la14	1210.8	1192	1131.8	1128	2699.4	2636	1917.0	1905	1524.2	1504
la15	1215.2	1197	1151.4	1132	2690.0	2656	1905.0	1855	1561.2	1548
10 × 10										
la16	769.8	763	717.0	717	1865.0	1865	1268.4	1247	1119.6	1102
la17	657.2	655	646.0	646	1718.0	1718	1202.8	1192	1034.2	1027
la18	734.2	725	664.2	663	1831.0	1831	1253.8	1249	1132.8	1114
la19	737.6	723	665.8	655	1765.0	1765	1274.8	1261	1148.8	1115
la20	779.2	775	756.0	756	1901.0	1901	1328.6	1297	1175.8	1139
15 × 10										
la21	1111.6	1085	1026.0	1018	2665.8	2640	1833.8	1796	1572.6	1555
la22	1005.2	989	945.8	930	2445.4	2420	1728.2	1698	1471.0	1449
la23	1125.2	1108	1056.4	1050	2574.4	2511	1830.0	1769	1535.4	1509
la24	1065.0	1059	989.8	978	2551.0	2532	1819.0	1781	1535.4	1507
la25	1036.6	1027	949.0	937	2511.4	2482	1760.6	1734	1483.0	1461
20 × 10										
la26	1417.4	1397	1278.2	1269	3391.6	3330	2385.4	2344	1994.4	1966
la27	1464.4	1446	1327.0	1316	3536.0	3484	2512.2	2450	2110.2	2057
la28	1447.0	1413	1325.8	1305	3465.0	3429	2439.0	2385	2057.4	2026
la29	1348.0	1320	1203.8	1175	3456.8	3416	2355.4	2281	2002.8	1972
la30	1416.4	1402	1296.6	1278	3449.8	3362	2472.6	2436	2076.8	2068
30 × 10										
la31	2064.0	2049	1886.0	1859	5120.6	4928	3540.0	3497	2942.6	2917
la32	2276.6	2231	2054.0	2023	5362.2	5330	3766.4	3750	3196.4	3152
la33	2061.2	2030	1852.4	1826	5096.0	4964	3606.6	3500	2971.8	2944
la34	2091.4	2077	1912.2	1885	5109.8	4959	3593.6	3515	3008.4	2968
la35	2111.0	2086	1929.8	1896	5235.0	5104	3603.4	3499	3011.8	2954
15 × 15										
la36	1212.4	1197	1077.0	1063	3028.8	2954	2056.2	2013	1849.6	1836
la37	1331.8	1311	1164.0	1139	3143.4	3051	2146.4	2116	1905.8	1887
la38	1148.0	1119	1020.4	1008	2965.0	2912	1998.8	1958	1773.6	1745
la39	1203.6	1151	1052.2	1032	2975.4	2919	2020.6	1985	1781.2	1752
la40	1227.8	1192	1056.4	1040	2999.6	2941	2048.0	2009	1789.0	1778

Table 5.2.: Detailed numerical results for various degrees of flexibility in the FBJS and FBJSS instances.

group	*FBJS*			*FBJSS*		
flex	*1*	*2*	*3*	*1*	*2*	*3*
10 × 5	-2.5%	-3.1%	-1.2%	-0.9%	-1.2%	-1.6%
15 × 5	-7.8%	-3.5%	-2.3%	-3.6%	-2.6%	-3.3%
20 × 5	-9.1%	-4.4%	-2.2%	-4.1%	-3.0%	-2.9%
10 × 10	-6.5%	-6.8%	-2.2%	-3.2%	-2.6%	-3.6%
15 × 10	-10.1%	-4.2%	-3.5%	-5.6%	-2.9%	-2.8%
20 × 10	-12.1%	-7.2%	-8.0%	-6.1%	-2.0%	-3.4%
30 × 10	-13.4%	-9.6%	-9.3%	-7.8%	-3.7%	-4.9%
15 × 15	-11.5%	-9.8%	-13.6%	-6.2%	-2.6%	-2.5%
All	-9.1%	-6.1%	-5.3%	-4.7%	-2.6%	-3.1%

Table 5.3.: Relative deviations of the results *avg* to the benchmarks *GPB* in the FBJS and FBJSS instances.

group	*FBJS*			*FBJSS*		
flex	*1*	*2*	*3*	*1*	*2*	*3*
10 × 5	1'687'000	4'616'000	3'892'000	1'319'000	4'791'000	3'764'000
15 × 5	3'040'000	2'621'000	2'184'000	2'447'000	3'115'000	2'443'000
20 × 5	2'403'000	1'704'000	1'404'000	2'960'000	2'100'000	1'651'000
10 × 10	3'000'000	1'318'000	780'000	3'480'000	2'643'000	1'455'000
15 × 10	1'539'000	922'000	699'000	2'108'000	1'318'000	783'000
20 × 10	877'000	398'000	218'000	1'375'000	849'000	522'000
30 × 10	482'000	197'000	122'000	799'000	435'000	251'000
15 × 15	913'000	214'000	105'000	1'444'000	992'000	567'000

Table 5.4.: Number of tabu search iterations per run (rounded to nearest 1000) in the FBJS and FBJSS instances.

result, far more iterations can be done in the same time. In the non-flexible FBJS instances of size 30 × 10, for example, about 4500 iterations are performed in [45] while about 482'000 iterations are now executed (on a similar PC with the same time limit). The reader may also check the number of iterations in FBJSS instances with *flex* = 2 by considering column 6 of Table 5.4 and column 2 of Table 3 in [45]. The number of iterations increased here by a factor of 7 to 20.

Another indicator for the quality of the tabu search is its improvement performance, i.e. the evolution of attained solution quality over computation time. To evaluate this feature, we recorded the best makespan ω at the beginning (initial solution) and during the execution of the search for each instance and run, and computed its relative deviation from the final solution $(\omega - \omega_{\text{final}})/\omega_{\text{final}}$. Figure 5.1 depicts these deviations for FBJS instances with flexibility 2 averaged over instances of the same size, e.g. the red line indicates the deviations for the 10 × 5 instances. Deviations from time 0 to 10 seconds are omitted for clarity. However, the deviations at the start ($t = 0$) are indicated in brackets in the legend of the chart. Furthermore, Table 5.4 depicts the number of tabu search iterations averaged over instances of the same size. The following observations can be made.

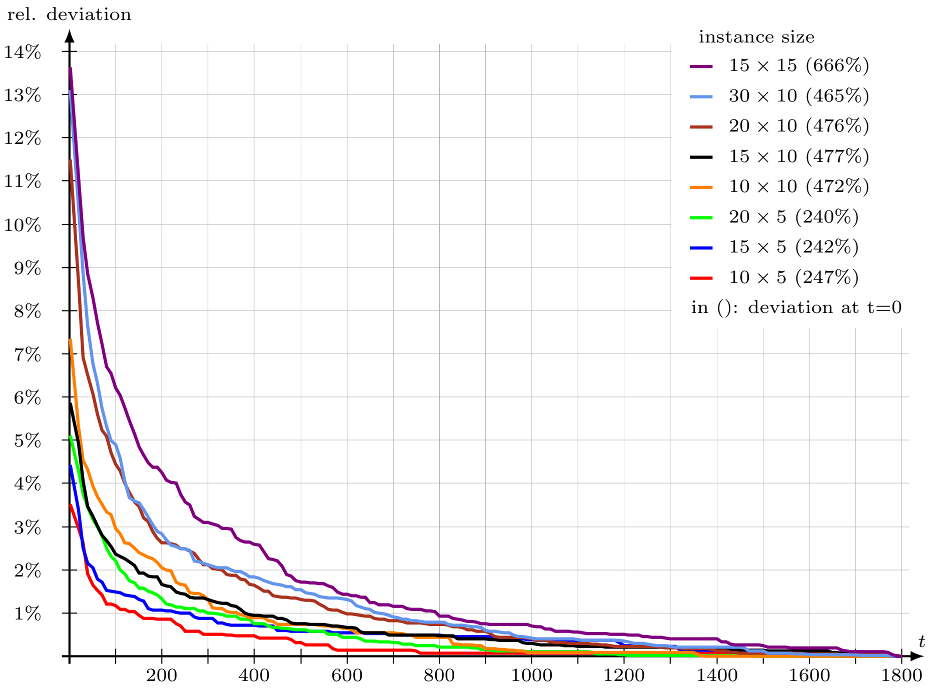


Figure 5.1.: Relative deviations of the makespan from the final makespan during runtime in FBJS instances with flexibility 2.

group	*FBJS*		*FBJSS*	
flex	*1 to 2*	*2 to 3*	*1 to 2*	*2 to 3*
10×5	-21.0%	-4.9%	-34.6%	-16.5%
15×5	-16.6%	-5.4%	-30.4%	-19.8%
20×5	-15.6%	-5.7%	-28.4%	-19.2%
10×10	-26.5%	-5.3%	-29.8%	-10.9%
15×10	-22.1%	-5.6%	-28.8%	-14.3%
20×10	-22.2%	-6.2%	-29.5%	-13.6%
30×10	-21.7%	-5.5%	-29.2%	-14.5%
15×15	-26.3%	-6.8%	-31.5%	-11.7%
All	-21.5%	-5.7%	-30.3%	-15.1%

Table 5.5.: Relative changes in the makespan when adding one machine per operation in the FBJS and FBJSS instances.

Initial solutions are far away from the obtained final solutions, with makespans three to seven times as large. Also, most of the improvements are found within minutes. Consider for example the 10×10 instances. While the deviation is initially 472%, it drops to 7.3% and 2.6% after 10 and 120 seconds, respectively. It is also notable that the solution quality is (roughly) at the same level as the current state of the art after about one minute of computation time. Similar results have been achieved in the other instances, suggesting good improvement performance and convergence of the tabu search. This can be partly attributed to the rather high number of tabu search iterations; they are in the range of 100'000 and 5'000'000. Note that in the smallest non-flexible instances, the tabu search stopped before reaching the time limit.

It is also of interest to examine the extent to which increasing flexibility decreases makespan. Information of this type may be of interest at the design stage, when the value of more flexible machines or additional machines is asserted. For each instance, the average makespan ω_k (over the five runs) with $k \in \{1, 2, 3\}$ denoting the degree of flexibility *flex* is compared to the average makespan ω_{k-1} of the corresponding instance with one alternative machine less for each operation. Table 5.5 depicts these changes in an aggregated way (over instances of the same size). The following observations can be made.

First, flexibility offers more potential for makespan reduction when transfer and setup times are present. For example in instances of size 20×5, when increasing the degree of flexibility from 1 to 2 the makespan is reduced by 15.6% in the absence of transfer and setup times (group FBJS with *flex 1 to 2*) and by 30.4.% otherwise (group FBJSS with *flex 1 to 2*). This can be attributed to the opportunity for two consecutive operations in a job to be performed on the same machine if their machine sets intersect, in which case transfer and setup times are saved. This effect is quite visible when schedules are displayed in Gantt charts (see Figure 5.3). Second, going from 1 to 2 machines per operation reduces the makespan significantly, the average decrease being 21.5% and 30.3% in the group FBJS and FBJSS, respectively. When adding a third machine, these numbers go down to 5.7% and 15.1%. While these figures are only estimates, they give an interesting indication on the benefit of flexibility and also of the diminishing return of adding flexibility.

We conclude the discussion of the computational results with two Gantt charts. Figure 5.2 depicts a schedule of FBJS instance *la17* without flexibility, and Figure 5.3 presents a schedule of FBJSS instance *la03* with flexibility 2. The thick bars represent the take-over, processing and hand-over steps of the operations while the narrow bars represent waiting times (filled) and setup times (hatched). The numbers refer to the operations, e.g. 5.1 refers to the first operation of job 5.

5.6. From No-Buffers to Limited Buffer Capacity

There are job shops in practice that do not comprise any buffers at all. However, in many cases a limited number of buffers is available. Hence, extending the blocking job shop to the job shop with limited buffer capacity appears valuable. This research direction is taken by several groups, among them Brucker et al. [15, 17] who consider various buffer configurations (general buffers, job-dependent buffers, pairwise buffers, machine output buffer, machine input buffers) and propose several solution representation schemes.

Buffer capacities can be captured (to some extent) in the FBJSS by modeling each buffer capacity unit as a machine. We assigned Marc Wiedmer in his Bachelor thesis [114] the task of developing a modeling tool allowing to specify complex job shop problems with buffers in a simple manner. The developed tool not only allows to specify various buffer configurations, among them input buffers, output buffers and buffers that are placed between pairs of machines (called intermediate or pairwise buffers), but also displays the disjunctive graph of the problems and generates input files which can be used further, e.g. with the JIBLS. A screenshot of the tool is given in Figure 5.4

We conclude this chapter by an experiment with a BJS instance and the ad hoc introduction of output buffers. Such experiments can be helpful at the design stage of a production system to assess the value of additional buffers.

The following two settings were analyzed in the experiment. Starting with the BJS instance *la17* (without transfer times and setup times), we attached one output buffer to each machine, and introduced after each operation on some machine m, a storage operation of duration 0 executed on the buffer that is attached to m. Based on the so obtained instance with one output buffer per machine, we created an instance with two parallel output buffers per machine by attaching a second output buffer to each machine and giving the choice to execute any storage operation on one of the two corresponding buffers.

Solutions of the two created instances are displayed in Figures 5.5 and 5.6. A line is added for each buffer above its corresponding machine indicated by the letter b. The storage operations with a positive duration are illustrated by narrow bars in the color of the corresponding job.

We compare the two illustrated solutions to the solution of the instance we started with displayed in Figure 5.2. While the makespan is 923 in the instance without buffers, it goes down to 803 with one buffer per machine and to 784 with two buffers per machine, which is a respective decrease of 13% and 15%, showing that a sub-

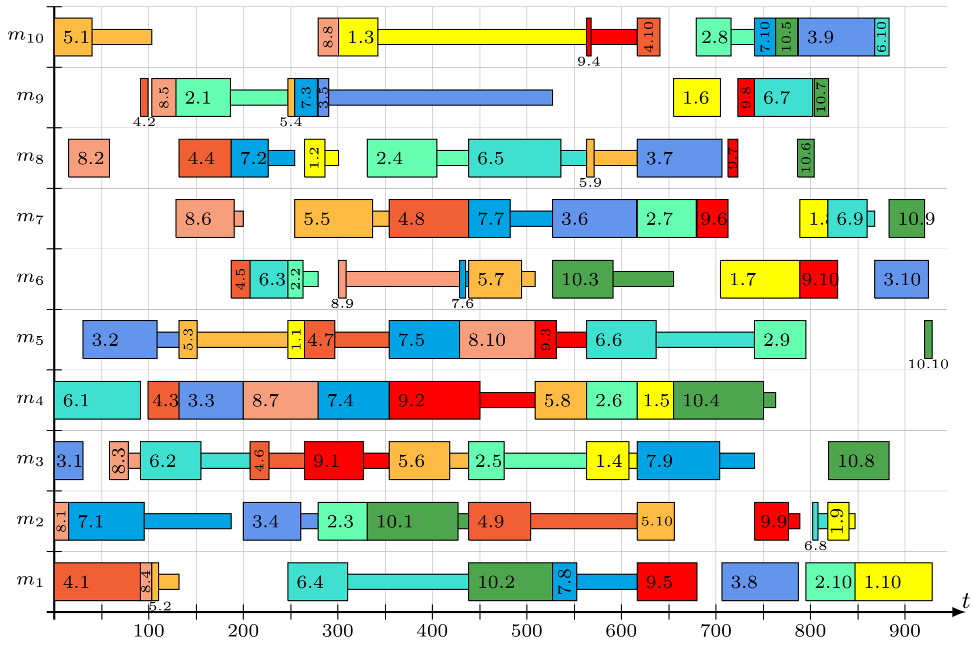


Figure 5.2.: A schedule with makespan 923 of FBJS instance *la17* without flexibility.

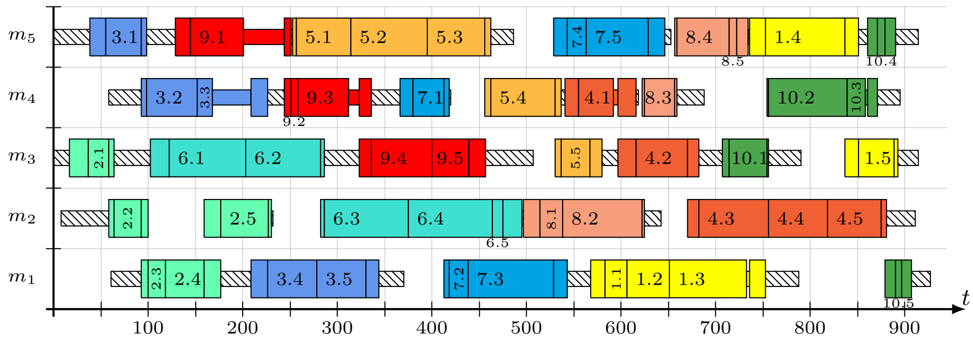
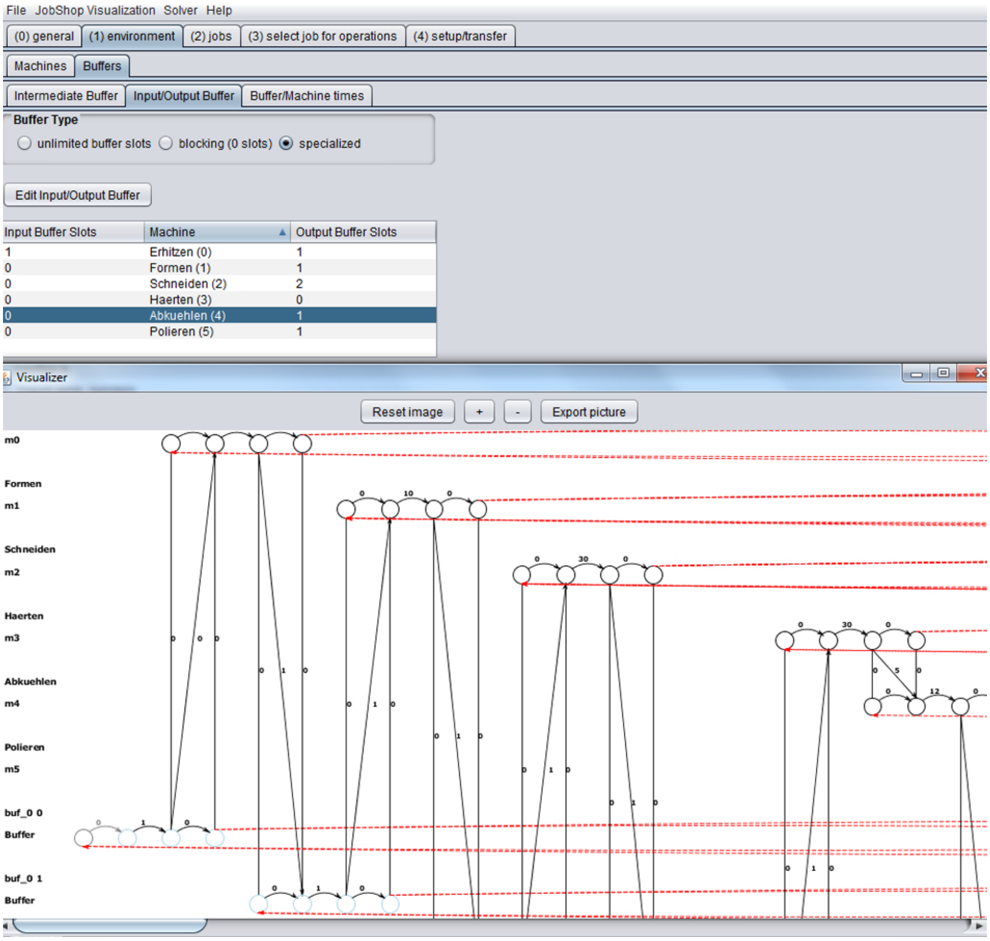


Figure 5.3.: A schedule with makespan 927 of FBJSS instance *la03* with flexibility 2.



Buffer Type

unlimited buffer slots, blocking (0 slots), [✓ specialized]

Input Buffer Slots	Machine	Output Buffer Slots
1	Erhitzen (0)	1
0	Formen (1)	1
0	Schneiden (2)	2
0	Haerten (3)	0
0	Abkuehlen (4)	1
0	Polieren (5)	1

Figure 5.4.: A screenshot of Marc Wiedmer's complex job shop modeling tool.

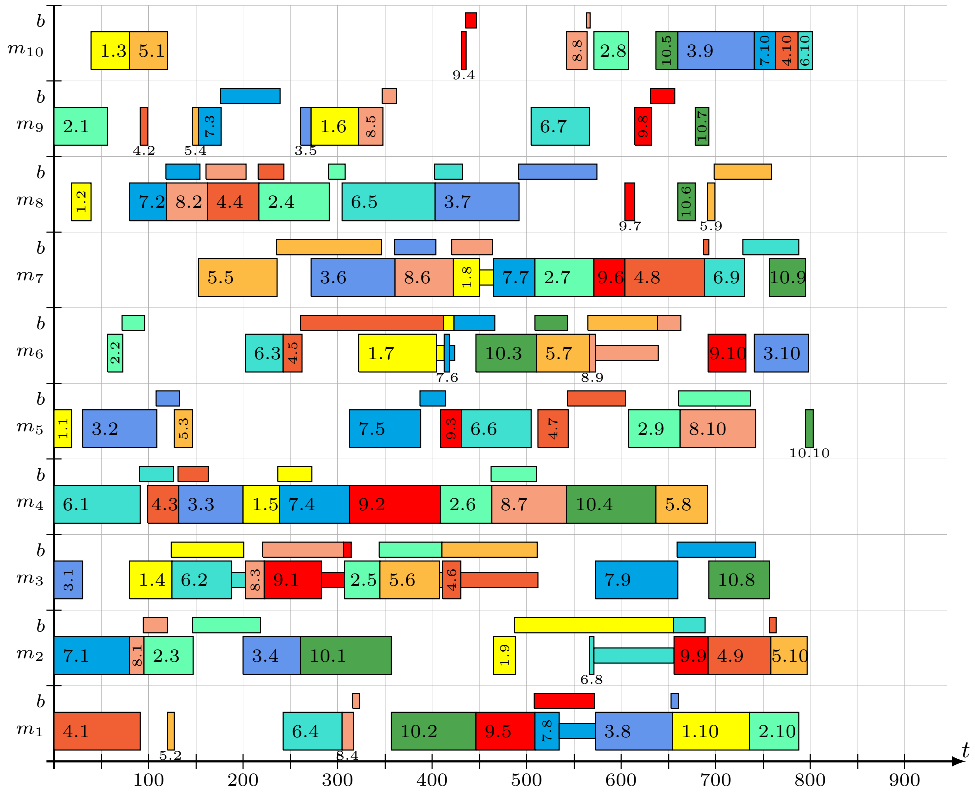


Figure 5.5.: A schedule with makespan 803 of instance *la17* with one output buffer per machine.

stantial amount of time can be saved in this instance by installing buffer capacities.

A remark on the quality of the two solutions is in order. It is known that an optimal solution of instance *la17* in the JS has makespan 784 (see e.g. Pezzella and Merelli [91]). Clearly, makespan 784 is then a lower bound for the optimal makespan in the versions with a limited number of buffers. Hence, the solution with two buffers and makespan 784 (Figure 5.6) is optimal.

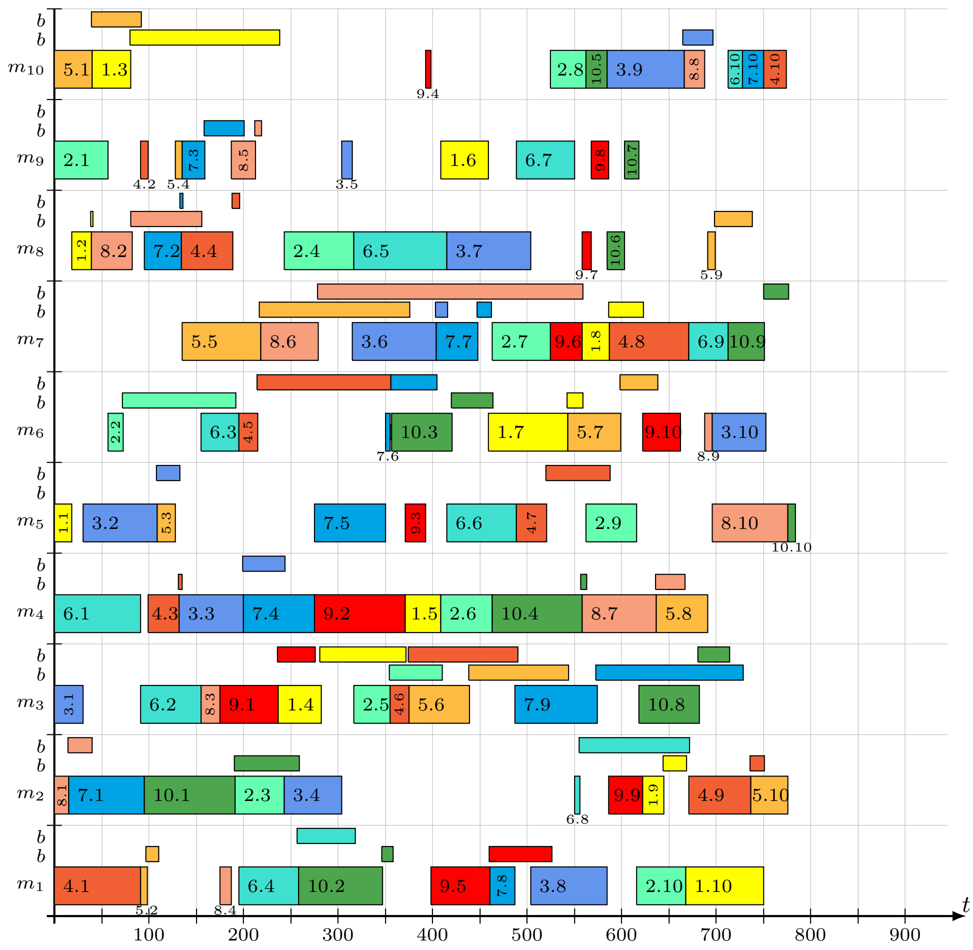


Figure 5.6.: A schedule with makespan 784 of instance *la17* with two parallel output buffers per machine.

CHAPTER 6

TRANSPORTATION IN COMPLEX JOB SHOPS

6.1. Introduction

In modern manufacturing systems, (semi-) automated mobile devices transport the jobs from one machine to the next. The mobile devices carry various names, they are called, for example, automated guided vehicles (AGVs) in flexible manufacturing systems (FMS), robots in robotic cells, cranes in container terminals and hoists in electroplating plants.

In such systems, it is important to schedule not only the machining operations but also the transport operations. Moreover, the scheduling problem should not be partitioned into a machine scheduling problem and a transportation problem as these two problems are often strongly linked with each other. For example Smith et al. [105] mention on this issue: "Due to the difficulties involved in job-shop scheduling, researchers have tended to simplify the scope of the problems in order to make them more tractable. The most common of these assumptions involves disregarding material handling activities under the premise that they are negligible and can be "added-on" to complete the schedule. However, the procedure for "adding-on" the material handling activities does not appear to be well-known." In their review on cyclic scheduling in robotic flow shops, Crama et al. [27] stress the following: "Classical scheduling models however, as they have been developed until the late seventies, appear to be unsuitable to incorporate the most important characteristics of flexible manufacturing cells, such as the interaction between the material handling system and the machines."

Thus, it appears valuable to integrate transportation also in the job shop models, as is done in this and the next chapter. The problems we consider can informally be described as follows. Given are jobs, ordinary machines and robots. As usual, a machine and a robot can handle at most one job at any time. A job is processed on machines in a sequence of machining operations and is transported from one machine to the next by a robot in transport operations. Thus a job can be seen as a sequence of alternating machining and transport operations. Moreover, there is some routing flexibility: while a machining operation is executed on a preassigned machine, a transport operation can be executed by any robot. Sequence-dependent setup times between transport operations will be a necessary feature, and we allow such setup times also between machining operations.

We consider three different versions of the problem with increasing difficulty. In the first version, we assume that the robots do not interfere with each other in their movements and an unlimited number of buffers is available, and call this version the JS-T. In the second version, we change the assumption concerning the buffers and state that there are no buffers available, and call this version the BJS-T. And finally, we consider a version of the BJS-T where the robots interfere with each other in their movements. This version is called the BJS-RT and is discussed in the next chapter.

It is well-known that the robots can be treated as "normal" machines with sequence-dependent setup times. Then, the JS-T can be seen as a special FJSS and similarly, the BJS-T can be seen as a special FBJSS, allowing us to apply the JIBLS also to the JS-T and the BJS-T. Although not tailored to these problems, we show in this chapter that the JIBLS is competitive when compared to the state of the art.

The chapter is structured as follows. The following section gives a selective literature review on job shop scheduling problems with transportation. The JS-T and the BJS-T are then discussed in Sections 6.3 and 6.4.

6.2. A Literature Review

Job shop scheduling problems that include transportation have been studied by several authors. Hurink and Knust [53] consider a single transportation robot in a JS. Bilge and Ulusoy [10], Brucker and Strotmann [18], Khayat et al. [56], Deroussi et al. [31] and Lacomme et al. [63] tackle a similar problem with multiple identical robots (with no spatial interferences between them). Only few papers address versions of the blocking job shop with transportation. Poppenborg et al. [97] consider a BJS with transportation and total weighted tardiness objective. They propose a MIP formulation and a tabu search. Brucker et al. [14] recently studied the cyclic BJS with one transportation robot.

To our knowledge, no paper (except our contribution [21]) considers a job shop setting (JS or BJS) with multiple robots interfering with each other, although the value of incorporating interferences between robots and limited buffer capacity has been recognized by several authors. For instance Khayat et al. [56] suggested future research to "include testing conflict avoidance and limited buffer capacities in a job shop setting".

Numerous papers deal with application-specific problems occurring in the context of hoist scheduling, robotic cell scheduling, scheduling of cranes in container terminals and factory crane scheduling. We briefly discuss a selection of representative articles.

In electroplating facilities, panels are covered with a coat of metal by immersing them sequentially in tanks, and hoists move the panels from tank to tank. Scheduling the coating operations as well as the movements of the hoists is commonly addressed as the hoist scheduling problem, cf. Manier and Bloch [73]. Many papers address versions with a single hoist, for example by Phillips and Unger [94], Shapiro and Nuttle [104] and Che and Chu [25]. Versions with multiple hoists are studied for example by Manier et al. [74] and Leung et al. [67, 68]. In these applications, empty hoists can move out of the way in order to avoid collisions, whereas loaded hoists have to move directly from tank to tank.

A typical robotic cell consists of an input device, a set of machines, an output device and one or multiple robots that transport the parts within the cell. Usually, there are no buffers available and the jobs have to visit the machines in the same order, i.e. in a flow shop manner, cf. Crama and Kats [27] and Dawande et al. [30]. Robotic cell scheduling problems with one robot have been considered for example by Sethi et al. [103], Crama and van de Klundert [28], Hertz et al. [50] and Hall et al. [48]. Versions with multiple robots have received less attention. Geismar et al. [35], Galante and Passannanti [33], Geismar et al. [36] consider multiple robots. Typically, collisions are avoided by assigning the robots non-overlapping working areas or, if the working areas overlap, restricting the access to an area to at most one robot at any time. Simple policies are established to control the access.

Crane scheduling in container terminals addresses the problem of scheduling transport operations (storage, retrieval and relocation) of containers executed by yard cranes. In many papers, systems with a single crane have been tackled, for example by Kim and Kim [58], Narasimhan and Palekar [81], Ng and Mak [83]. Multiple cranes have been considered e.g. by Ng [82] who partitions the yard into non-overlapping areas, one for each crane, to eliminate the occurrence of collisions, and by Li et al. [69] who use a time-discretized MIP formulation to enforce a minimum distance between cranes at any time period.

Factories frequently comprise track-mounted overhead cranes for moving parts between different locations. Typically, the lifting component of a crane, called hoist, is mounted on a crossbar on which it moves laterally and the crossbar itself moves longitudinally along a track, which may be shared by multiple cranes, cf. Peterson et al. [90]. Factory crane scheduling consists in scheduling the transportation of the parts in order to meet a given manufacturing schedule. Factory crane scheduling problems are addressed, for example, by Liebermann and Turksen [70], Tang et al. [109], Aron et al. [5] and Peterson et al. [90].

Several authors emphasize that the presence of multiple robots that interfere with each other increases complexity, e.g. Leung et al. [68] write: “the scheduling problem for multi-hoist lines is significantly more difficult than for single-hoist lines because of the additional problem of hoist collision avoidance.”

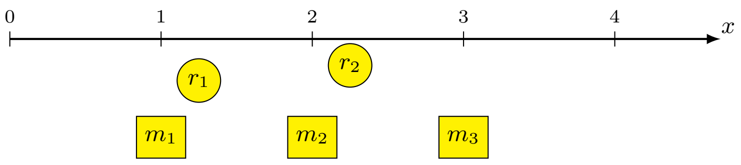


Figure 6.1.: A layout in the JS-T example.

6.3. The Job Shop with Transportation (JS-T)

6.3.1. A Problem Formulation

The Job Shop with Transportation (JS-T) can be seen as a FJSS instance with transportation, characterized by an unlimited number of buffers and no interferences between the robots. The transportation features can be described as follows.

The set of machines $M = M' \cup R$ is partitioned into a set of ordinary machines M' and a set of robots R. Similarly the set of operations $I = I^M \cup I^R$ is partitioned into set I^M of machining operations and set I^R of transport operations. As usual, all machines $m \in M$ can handle at most one job at any time.

There is some routing flexibility: While each machining operation $i \in I^M$ is executed on a preassigned machine $m_i \in M'$, each transport operation $i \in I^R$ can be executed by any robot $r \in R$.

The operations of a job $J \in \mathcal{J}$ are alternately machining and transport operations, and we assume that $|J|$ is odd, $J_q \in I^M$ for q odd and $J_q \in I^R$ for q even, $1 \leq q \leq |J|$. Note that typically the first operation J_1 will consist in loading job J at some storage place or device, and the last operation $J_{|J|}$ will represent the unloading of completed job J.

The robots might not be identical, e.g. they may move at different (maximum) speeds. Indeed, the duration $d^{\mathrm{p}}(i, r)$ of a transport operation i depends on the assigned robot $r \in R$ and might be different for various $r \in R$. Moreover, also the setup times $d^{\mathrm{s}}(i, r; j, r)$ occurring between two transport operations on a same robot r depend on $r \in R$.

A schedule consists of an assignment of a robot for each transport operation and a starting time for each (machining and transport) operation so that all constraints described in the FJSS and given above are satisfied. The objective is to find a schedule with minimal makespan.

An illustration of the JS-T is given in an example, which is based on the job shop example introduced in Figure 1.2 (Section 1.3), assuming that the machines are located on a line, machine m_1, m_2 and m_3 being at location $1, 2$ and 3, respectively, measured on the x-axis, and there are two robots r_1 and r_2 available for the transportation of the jobs between the machines. Figure 6.1 depicts this layout. The robots have a maximum speed of 1, e.g. a move between machines m_1 and m_3 needs two time units. The location of r_1 and r_2 at the beginning and at the end is the same, namely 0 and 1 for r_1 and r_2, respectively.

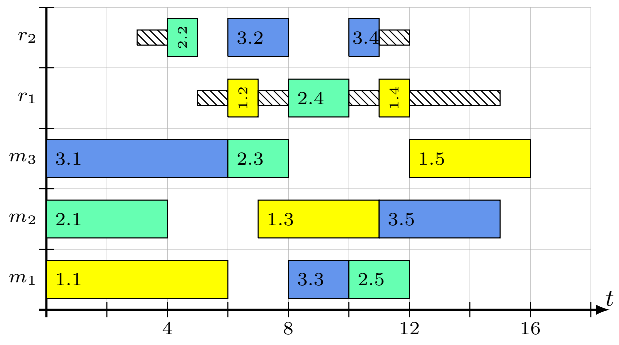


Figure 6.2.: A solution of the JS-T example.

In Figure 6.2 a solution of the example is depicted in a Gantt chart. The numbering of the operations is changed compared to Figure 1.2 due the added transport operations. Note that the dashed lines stand for idles moves, i.e. for the setups, of the robots.

6.3.2. Computational Results

We examine if the JIBLS also finds good solutions in the JS-T instances. For this purpose, we use the standard JS-T instances of Hurink and Knust [53] (HK) and compare the obtained results to best benchmarks of these instances. As the number of robots is one in all the instances of Hurink and Knust, we also created for each HK instance two instances with multiple robots by introducing a second and third identical robot.

Note that we also considered the instances of Bilge and Ulusoy [10] (BU), which are often used in the literature. The benchmark set BU contains 40 instances with 5 to 8 jobs, 3 to 4 machines and 2 robots. Numerical experiments have shown that the instances are quite simple to solve. In fact, using a MIP with the solver Gurobi 5.5 [46], near-optimal or optimal solutions are found in these instances within seconds. Very similar results are obtained by the JIBLS and by the state of the art methods of Deroussi et al. [31] and Lacomme et al. [63]. Hence, a comparison of the different methods is somewhat superfluous.

The computational settings were similar to those described in the FJSS (Chapter 4). The JIBLS was run on a PC with 3.1 GHz Intel Core i5-2400 processor and 4 GB memory. The initial solution was a permutation schedule obtained by randomly choosing a job permutation and a mode. For each instance, five independent runs with different initial solutions were performed. The computation time of a run was limited to 600 seconds and the tabu search parameters were set as $maxt = 10$, $maxl = 300$ and $maxIter = 6000$.

We experimented with the two neighborhood versions $\mathcal{N}^1$ and $\mathcal{N}^2$ (see Section 4.6.3). Neighborhood $\mathcal{N}^1$ was slightly better than $\mathcal{N}^2$ in the HK instances.

instance	*1 robot*		*2 robots*		*3 robots*	
	best	*avg*	*best*	*avg*	*best*	*avg*
6×6						
P01.dat.D1_d1	87	87.0	63	63.0	62	62.0
P01.dat.D1_t1	81	81.0	62	62.0	62	62.0
P01.dat.D2_d1	151	152.8	84	84.2	73	73.0
P01.dat.D3_d1	217	218.4	113	113.8	88	88.0
P01.dat.T2_t1	74	74.6	62	62.0	61	61.0
P01.dat.T3_t0	92	92.0	65	65.0	65	65.0
P01.dat.tikl.1	137	138.2	78	78.4	74	74.0
P01.dat.tikl.2	132	133.0	77	77.0	75	75.0
P01.dat.tikl.3	148	148.0	80	80.0	72	72.0
P01.dat.tkl.1	139	140.0	77	77.4	71	71.0
10×10						
P02.dat.D1_d1	984	1000.2	954	976.6	955	959.6
P02.dat.D1_t0	988	1006.2	954	967.8	955	966.0
P02.dat.D1_t1	966	988.0	954	975.0	961	971.8
P02.dat.D2_d1	1014	1041.6	973	999.4	971	981.2
P02.dat.D3_d1	1047	1069.4	988	1005.2	988	1005.2
P02.dat.D5_t2	1323	1331.0	1019	1039.2	1020	1029.2
P02.dat.T1_t1	950	990.6	934	961.8	934	948.4
P02.dat.T2_t1	973	999.8	941	961.6	941	956.8
P02.dat.T5_t2	1006	1034.8	970	988.6	970	980.8
P02.dat.mult0.5_D1_d1	544	561.0	522	530.6	519	530.4
P02.dat.mult0.5_D1_t1	537	548.8	522	533.0	522	525.2
P02.dat.mult0.5_D2_d1	636	644.8	542	556.8	538	541.2
P02.dat.mult0.5_D2_t0	561	579.6	541	548.8	538	544.2
P02.dat.mult0.5_D2_t1	584	605.2	542	555.0	538	546.0
P02.dat.tikl.1	973	1018.8	966	987.4	957	960.6
P02.dat.tikl.2	983	1011.4	957	976.2	962	968.2
P02.dat.tikl.3	990	1012.8	960	972.8	954	969.8
P02.dat.tikl.4	972	1009.0	965	986.4	964	965.6
P02.dat.tkl.1	985	1021.6	956	978.8	956	974.2
P02.dat.tkl.2	991	1030.0	958	976.2	958	970.2

Table 6.1.: Detailed numerical results in the JS-T instances.

Table 6.1 displays the obtained results with $\mathcal{N}^1$ as follows. The table is divided into three blocks. Block 1 (columns 2-3), 2 (columns 4-5) and 3 (columns 5-6) displays the *best* and average (*avg*) results over the five runs of the instances with 1, 2 and 3 robots, respectively. In the first column the name of the HK instance is given. The instances are grouped according to size, e.g. the first group 6×6 consists of instances with 6 jobs and 6 machines.

The results of the instances with one robot are compared in Table 6.2 with the best current benchmarks to our knowledge, namely the benchmarks of Hurink and Knust [53] and Lacomme et al. [63]. Column 1 gives the name of the instance and column 2 presents the *best* results over the five runs. Benchmarks *HK1*, *HK2* and *HK3*, listed in columns 3-5, are the best results of Hurink and Knust obtained in 6 runs of their “short one-stage” approach (*HK1*), 12 runs of their “short combined approach” (*HK2*), and 12 runs of their “long combined” approach (*HK3*). Benchmarks *LA1*, listed in column 6, are the best values obtained by Lacomme et al. in 5 runs. The

instance	*best*	*HK1*	*HK2*	*HK3*	*LA1*
6 × 6					
P01.dat.D1_d1	**87**	**87**	88	-	**87**
P01.dat.D1_t1	**81**	**81**	83	-	**81**
P01.dat.D2_d1	151	**148**	153	-	**148**
P01.dat.D3_d1	217	217	216	-	**213**
P01.dat.T2_t1	**74**	**74**	**74**	-	**74**
P01.dat.T3_t0	**92**	**92**	93	-	**92**
P01.dat.tikl.1	137	**134**	137	-	-
P01.dat.tikl.2	132	**129**	134	-	-
P01.dat.tikl.3	148	**144**	**144**	-	-
P01.dat.tkl.1	139	137	141	-	**136**
10 × 10					
P02.dat.D1_d1	**984**	1044	1013	990	1012
P02.dat.D1_t0	**988**	1042	989	989	1017
P02.dat.D1_t1	**966**	1016	995	989	983
P02.dat.D2_d1	1014	1070	1004	**993**	1045
P02.dat.D3_d1	**1047**	1070	1078	1072	1100
P02.dat.D5_t2	**1323**	1325	1383	1371	1361
P02.dat.T1_t1	**950**	1006	1022	1018	978
P02.dat.T2_t1	**973**	1015	1053	1030	993
P02.dat.T5_t2	**1006**	1102	1090	1020	1022
P02.dat.mult0.5_D1_d1	**544**	555	562	558	581
P02.dat.mult0.5_D1_t1	**537**	544	551	542	546
P02.dat.mult0.5_D2_d1	636	**633**	674	666	673
P02.dat.mult0.5_D2_t0	**561**	578	595	595	584
P02.dat.mult0.5_D2_t1	**584**	613	621	620	620
P02.dat.tikl.1	**973**	1082	1089	1027	1009
P02.dat.tikl.2	**983**	1035	1087	1033	1002
P02.dat.tikl.3	990	1039	1081	**989**	-
P02.dat.tikl.4	**972**	1045	1084	997	-
P02.dat.tkl.1	**985**	1086	1061	1018	-
P02.dat.tkl.2	**991**	1028	1058	1014	-

Table 6.2.: The results *best* compared to benchmarks (*HK1*, *HK2*, *HK3*, *LA1*) in the JS-T instances with one robot (bold: best).

runtime limit in *best*, *HK1*, *HK2* and *LA1* are the same, namely 600 seconds while *HK3* uses up to 3600 seconds per run. The following can be observed.

The results *best* are on average 2.7%, 3.9%, 2.9% and 2.0% better than benchmarks *HK1*, *HK2*, *HK3* and *LA1*, respectively. Moreover, *best* gives the lowest values in 21 instances (out of 30) while *HK1*, *HK2*, *HK3* and *LA1* present in 9, 2, 2 and 7 instances, respectively, the lowest values (see numbers in bold). Altogether, the JIBLS appears also competitive in the JS-T when compared to the best current benchmarks.

Since the HK instances with multiple robots have not been addressed in the literature, a comparison of the obtained results with benchmarks is not possible. For this reason, we tried to assess the quality of the solutions with results obtained via a MIP model that we derived in a straightforward manner from the disjunctive graph formulation. The model was implemented in the mathematical modeling language LPL [54], and all instances (including the instances with one robot) were solved using the solver Gurobi 5.5 [46] with a time limit of 3600 seconds. Table 6.3 shows

instance	*1 robot*	*2 robots*	*3 robots*
6×6			
P01.dat.D1_d1	87	63	62
P01.dat.D1_t1	80	62	62
P01.dat.D2_d1	(104;151)	(76;84)	73
P01.dat.D3_d1	(135;213)	(93;115)	(84;88)
P01.dat.T2_t1	74	62	61
P01.dat.T3_t0	92	65	65
P01.dat.tikl.1	(79;138)	(74;79)	74
P01.dat.tikl.2	132	(75;77)	75
P01.dat.tikl.3	(89;146)	(70;81)	72
P01.dat.tkl.1	(86;142)	(71;78)	71
10×10			
P02.dat.D1_d1	957	(771;964)	(831;955)
P02.dat.D1_t0	954	(795;961)	(861;954)
P02.dat.D1_t1	(861;956)	954	954
P02.dat.D2_d1	(850;998)	972	(818;973)
P02.dat.D3_d1	(873;1076)	988	988
P02.dat.D5_t2	(847;1369)	(1014;1019)	1019
P02.dat.T1_t1	934	(765;953)	(772;960)
P02.dat.T2_t1	(867;942)	941	(775;952)
P02.dat.T5_t2	974	970	(787;980)
P02.dat.mult0.5_D1_d1	(424;544)	518	518
P02.dat.mult0.5_D1_t1	(474;522)	(439;522)	(446;526)
P02.dat.mult0.5_D2_d1	(444;678)	539	(477;538)
P02.dat.mult0.5_D2_t0	(476;573)	538	538
P02.dat.mult0.5_D2_t1	(442;617)	(471;546)	538
P02.dat.tikl.1	957	(846;964)	(830;957)
P02.dat.tikl.2	959	(815;962)	957
P02.dat.tikl.3	954	954	954
P02.dat.tikl.4	961	959	(786;971)
P02.dat.tkl.1	965	956	956
P02.dat.tkl.2	963	958	958

Table 6.3.: MIP results in the JS-T instances with one to three robots.

detailed results by providing the optimal values or upper and lower bounds (ub;lb) if optimality could not be established. The following can be observed.

Optimality is established in about half of the instances. In fact, in 14, 16, and 19 instances with 1,2 and 3 robots, respectively, optimal solutions are found. In all other instances, a feasible solution is found, which is not the case for instances of the same size in more complex job shop problems with transportation (cf. the BJS-T in Table 6.5 and the BJS-RT in Table 7.2).

Now comparing the MIP results with the tabu search results *best* of Table 6.1. In the instances where the MIP established optimality, the average relative deviations $(best - MIP)/MIP$ are 1.8%, 0.2% and 0.1% for instances with 1, 2 and 3 robots, and in the other instances they are -0.7%, -0.7% and -0.6%, respectively. These numbers suggest that the tabu search finds good results within a short time.

It is also of interest to examine the impact on the makespan when increasing the number of robots. This information may be of value when the gain of using more

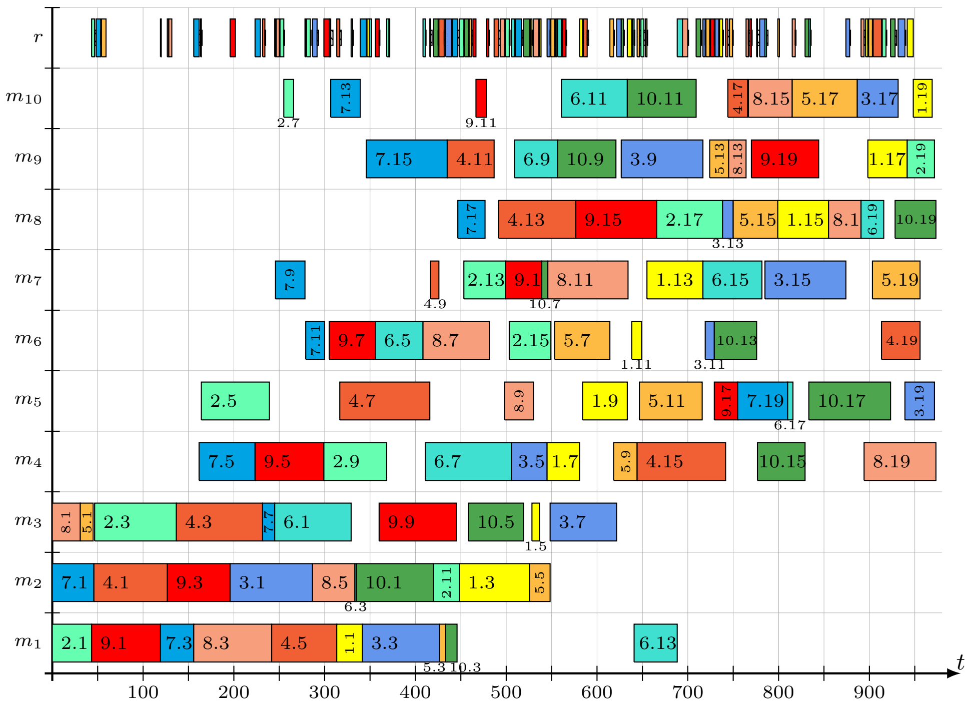


Figure 6.3.: A schedule of the JS-T instance *P02.dat.tikl.1* with one robot having makespan 973.

robots needs to be asserted. For each instance, the average makespan ω_k (over the five runs of the tabu search) with $k \in \{1, 2, 3\}$ denoting the number of robots is compared to the average makespan ω_{k-1} of the corresponding instance with one robot less. The following observations can be made. When adding a second robot, the makespan goes down on average by 15.8%. In some instances the decrease is quite low, e.g. 1.3% in instance *P02.dat.D1_t1*, while in other instances the decrease is quite large, e.g. 47.9% in instance *P01.dat.D3_d1*. Adding a third robot offers less potential. Indeed, the makespan goes down by another 2.9% on average. Nevertheless, in some instances the additional decrease is substantial, e.g. 22.7% in *P01.dat.D3_d1* and 10.0% in *P01.dat.tikl.3*.

We conclude this section with Figure 6.3 depicting in a Gantt chart a schedule of instance *P02.dat.tikl.1* with one robot having makespan 973. For clarity, the numbers referring to the transport operations are omitted (see the bars on the line of the robot r).

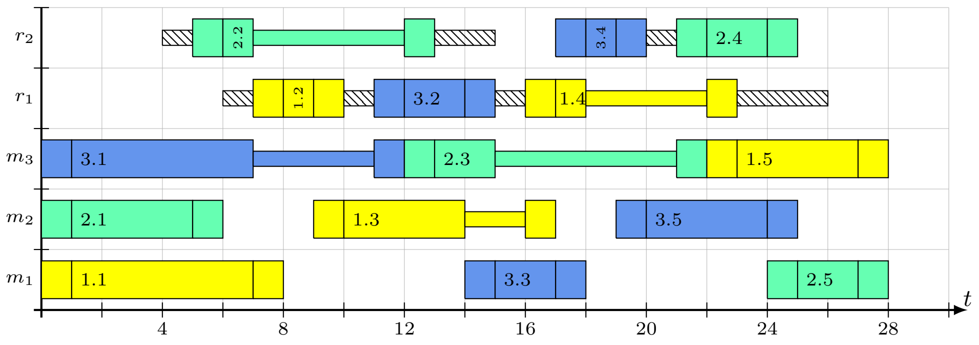


Figure 6.4.: A solution of the BJS-T example.

6.4. The Blocking Job Shop with Transportation (BJS-T)

In practical applications of the job shop with transportation such as in robotic cells, in electroplating plants and in container terminals, the number of buffers is often limited or there are no buffers at all. In this section, we consider such a problem, namely the JS-T without buffers, called the Blocking Job Shop with Transportation (BJS-T).

6.4.1. A Problem Formulation

The Blocking Job Shop with Transportation (BJS-T) can be seen as a FBJSS instance with transportation, characterized by the absence of buffers and no interferences between the robots. The transportation features are those introduced for the JS-T in Section 6.3. The Blocking Job Shop with Transportation (BJS-T) can also be seen as a JS-T instance without buffers and with transfer times.

An illustration is given in the example of Section 6.3, assuming that no buffers are present and each transfer (take-over and hand-over) has duration 1. Figure 6.4 depicts a solution of this example.

6.4.2. Computational Results

We investigate the performance of the JIBLS in the BJS-T. Since the problem does not appear in the literature, we created a test set of 120 instances, starting from the JS-T instances of Hurink and Knust (HK) with one, two and three robots and setting all transfer times (including loading and unloading times) to 1. The computational settings were the same as in the FBJSS (Chapter 5).

Detailed results are provided in Table 6.4. The table is divided into three blocks. Block 1 (columns 2-3), 2 (columns 4-5) and 3 (columns 5-6) displays the *best* and average (*avg*) results over the five runs of the instances with 1, 2 and 3 robots, respectively. In the first column the name of the (basic) HK instance is given. The instances are grouped according to size. We now discuss the results, evaluating the

instance	*1 robot*		*2 robots*		*3 robots*	
	best	*avg*	*best*	*avg*	*best*	*avg*
6×6						
P01.dat.D1_d1	191	191.0	115	115.0	89	89.8
P01.dat.D1_t1	161	161.0	100	100.0	85	85.0
P01.dat.D2_d1	255	255.0	141	141.0	105	105.2
P01.dat.D3_d1	321	321.0	172	172.6	125	126.0
P01.dat.T2_t1	156	156.0	98	98.8	84	84.0
P01.dat.T3_t0	157	157.0	98	98.0	86	86.0
P01.dat.tikl.1	222	222.0	123	123.4	96	96.4
P01.dat.tikl.2	214	214.0	123	123.4	98	98.0
P01.dat.tikl.3	229	229.0	130	130.8	99	99.0
P01.dat.tkl.1	240	240.0	133	133.0	101	101.0
10×10						
P02.dat.D1_d1	1396	1417.2	1087	1107.4	1024	1042.2
P02.dat.D1_t0	1295	1307.4	1058	1098.4	1005	1014.0
P02.dat.D1_t1	1307	1313.2	1103	1111.0	1012	1020.0
P02.dat.D2_d1	1450	1504.6	1138	1167.2	1054	1062.4
P02.dat.D3_d1	1681	1720.0	1188	1214.4	1078	1082.6
P02.dat.D5_t2	1889	2029.4	1279	1318.8	1120	1160.0
P02.dat.T1_t1	1259	1272.0	1077	1089.8	998	1003.8
P02.dat.T2_t1	1289	1315.6	1086	1101.2	1005	1016.6
P02.dat.T5_t2	1447	1479.0	1123	1147.6	1041	1046.2
P02.dat.mult0.5_D1_d1	935	940.6	653	694.4	593	620.8
P02.dat.mult0.5_D1_t1	819	833.0	634	649.2	577	601.6
P02.dat.mult0.5_D2_d1	1141	1190.2	717	771.6	622	651.2
P02.dat.mult0.5_D2_t0	927	940.0	677	691.2	608	615.4
P02.dat.mult0.5_D2_t1	994	1011.8	678	701.2	603	624.6
P02.dat.tikl.1	1368	1397.6	1108	1131.2	1023	1031.8
P02.dat.tikl.2	1344	1393.0	1112	1129.6	1015	1037.6
P02.dat.tikl.3	1340	1363.4	1116	1118.2	1023	1031.2
P02.dat.tikl.4	1370	1386.6	1127	1135.8	1018	1040.4
P02.dat.tkl.1	1387	1419.2	1112	1131.0	1019	1035.2
P02.dat.tkl.2	1393	1438.4	1125	1140.2	1030	1052.8

Table 6.4.: Detailed numerical results in the BJS-T instances.

solution quality, the impact of the absence of buffers and the impact of the number of robots.

Since no benchmarks are available in the literature, we tried to assess the quality of the solutions with results obtained via a MIP model that we derived in a straightforward manner from the disjunctive graph formulation. The model was implemented in LPL [54], and the instances were solved using the solver Gurobi 5.5 [46] with a time limit of 3600 seconds. Table 6.5 shows the results by providing the optimal values or upper and lower bounds (ub;lb). The following observations can be made.

First, all instances with 10 machines and 10 jobs remained unsolved as the solver could not find a feasible solution. For this reason, only results for the 6×6 instances appear in the table. Second, even in these instances, an optimal solution could not always be found. Instances with multiple robots seem to be of a particular difficulty. Third, comparing the results of Table 6.4 with those achieved by the MIP solver, in

instance	*1 robot*	*2 robots*	*3 robots*
P01.dat.D1_d1	191	(83;117)	(79;90)
P01.dat.D1_t1	161	(76;100)	(83;85)
P01.dat.D2_d1	255	(99;145)	(89;108)
P01.dat.D3_d1	(274;321)	(126;174)	(100;129)
P01.dat.T2_t1	156	(88;99)	84
P01.dat.T3_t0	157	(93;98)	(85;86)
P01.dat.tikl.1	222	(94;129)	(88;96)
P01.dat.tikl.2	214	(94;127)	(89;98)
P01.dat.tikl.3	229	(93;134)	(89;99)
P01.dat.tkl.1	240	(91;135)	(87;103)

Table 6.5.: MIP results for the BJS-T instances of size 6×6.

all instances the best of the five tabu search runs reached the MIP optimum or upper bound, and the results of all five runs are as good or very close to the MIP results. Although limited, these results suggest that the JIBLS performs well in the BJS-T.

Furthermore, we investigate the impact of the absence of buffers. For this purpose, we compare for each instance the average makespan with the average makespan of the corresponding instance in the JS-T (with unlimited buffers). The following observations can be made. On average, the makespan is 76.0%, 62.6% and 38.0% higher in the BJS-T instances of size 6×6 with 1, 2 and 3 robots, respectively, and 46.2%, 19.3% and 9.4% higher in the instances of size 10×10 with 1, 2 and 3 robots, respectively. These numbers suggest that buffers have a significant impact on the makespan and the robots act as buffers. This effect can also be seen in the examples depicted in Figures 6.4 and 6.5.

It is also of interest to examine the impact on the makespan when increasing the number of robots. For this purpose, we compare the average makespan ω_k of each instance with $k \in \{1, 2, 3\}$ robots to the average makespan ω_{k-1} of the corresponding instance with one robot less. The following observations can be made. Additional robots reduce the makespan significantly. Indeed, when adding a second and third robot, the makespan goes down by 40.7% and 20.7% on average in instances of size 6×6 and by 22.4% and 9.1% in instances of size 10×10.

This section is concluded by showing a schedule of instance *P02.dat.mult0.5_D2_t1* with two robot in Figure 6.5. For clarity, the numbers referring to the transport operations are omitted.

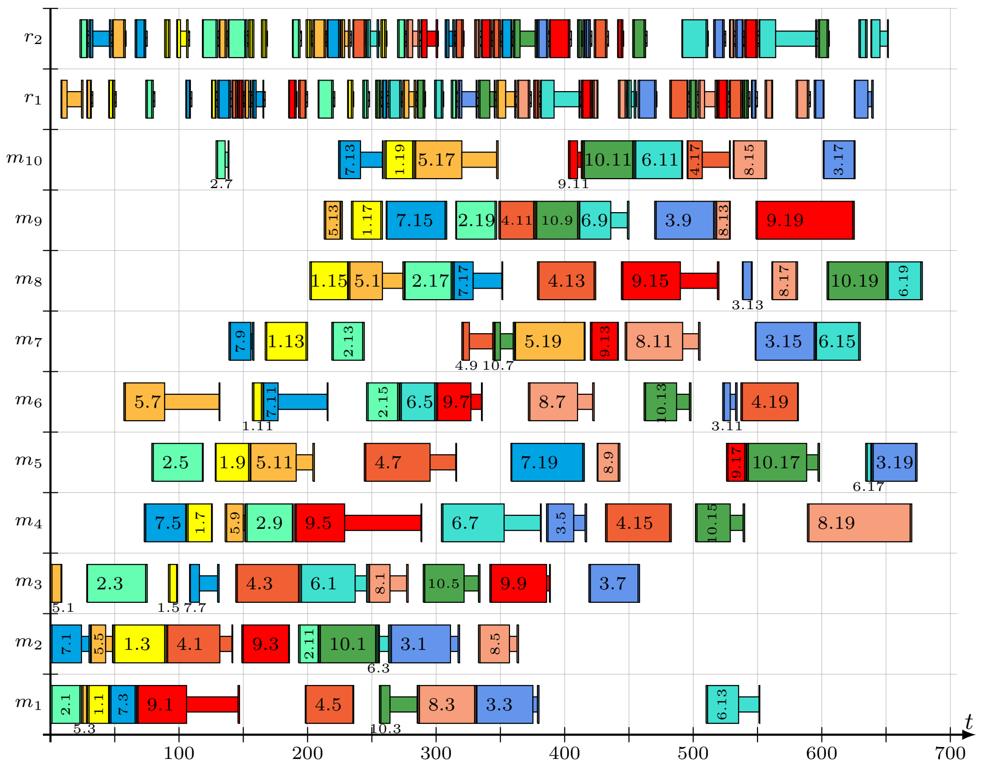


Figure 6.5.: A schedule of the BJS-T instance *P02.dat.mult0.5_D2_t1* with two robot and makespan 678.

CHAPTER 7

THE BLOCKING JOB SHOP WITH RAIL-BOUND TRANSPORTATION

7.1. Introduction

In practice, not only the number of buffers is limited but often the transportation devices interfere with each other. Typically, interferences arise when these devices move in a common transportation network. For example, hoists in electroplating plans, robots in robotic cells and cranes in container terminals occasionally move on a single rail line. As mentioned in the literature review in Section 6.2, these interferences substantially increase the complexity of the scheduling problem due to the additional problem of preventing robot collisions.

In this chapter, we consider a version of the BJS-T where the robots move on a single rail line, and call this problem the Blocking Job Shop with Rail-Bound Transportation (BJS-RT). The material of this chapter is to a main part taken from the publication [21].

The transportation system considered here consists of multiple robots moving on a common rail line along which the machines are located. The robots cannot pass each other and must maintain a minimum distance from each other, but can move "out of the way". Also, a robot can move at a speed up to a limit which can be robot dependent.

The objective is to determine the starting time of each transport and machining operation, the assigned robot of each transport operation, and the trajectory, i.e. the location at any time, of each robot, in order to minimize the makespan.

An illustration of the BJS-RT is given in an example, which is based on the BJS-T example described in Section 6.4.1. Now assume that the robots move on a single rail line along which the machines are located and the minimum distance between the robots is 1 (see Figure 7.1).

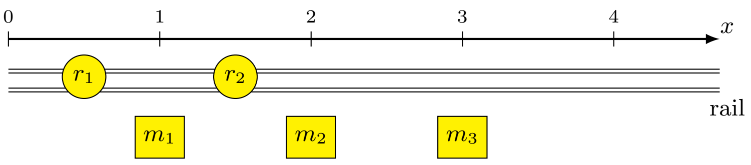


Figure 7.1.: The rail layout in the example.

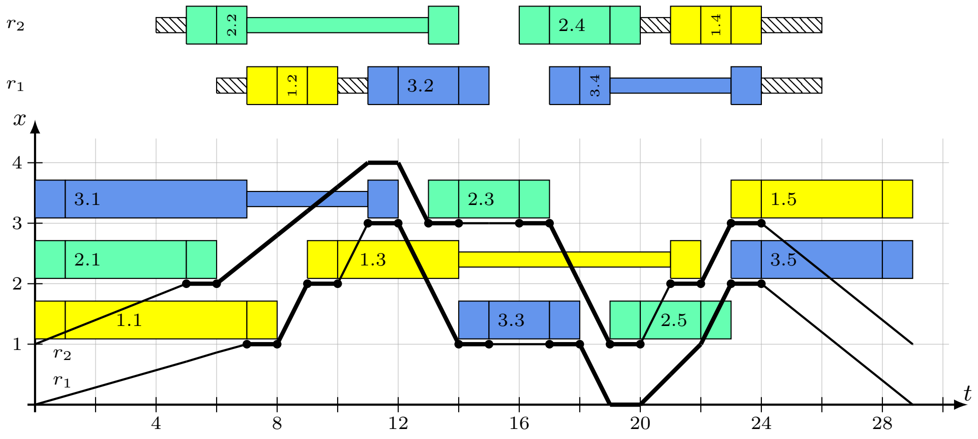


Figure 7.2.: A solution of the BJS-RT example.

Figure 7.2 depicts a solution of this problem in a Gantt chart like time-location diagram. The horizontal and vertical axes stand for the time and the location on the rail. The location of the machining operation bars represents the location of the corresponding machine. The transport operations are depicted above the x-axis as their location is not fixed. Feasible trajectories of the robots are displayed by two black lines. Thick line sections indicate that a robot is loaded with a job while thin sections stand for idle moves. The black dots on the lines indicate the start or completion of a transfer step.

7.2. Notation and Data

We slightly extend the notation of the BJS-T to capture transports on a rail.

The locations of the machines and robots along and on the rail are measured on an x-axis. For each machine $m \in M'$, let a_m be its fixed location. Also, for each robot $r \in R$, let $x(r,t)$ denote the (variable) location of r at time t, and $a_{\sigma r}$ and $a_{\tau r}$ be prescribed initial and end location, i.e. $x(r,0) = a_{\sigma r}$ and $x(r,\omega) = a_{\tau r}$ must hold at makespan ω. Furthermore, for each robot $r \in R$, let $v_r > 0$ be its maximum speed, and $\delta > 0$ be the minimum distance to be maintained between two consecutive robots on the rail. Finally, L is the usable rail length: $0 \le x(r,t) \le L$ for all $r \in R$ and all t.

For each operation $i \in I$, let a_{o_i} and $a_{\overline{o}_i}$ be the locations of its hand-over and take-over steps. Note that these locations are determined by the machine locations: if $i \in I^M$, $a_{o_i} = a_{\overline{o}_i} = a_{m_i}$; if $i \in I^R$, $i \in J$ and $j, k \in J$ are the (machining) operations preceding and following i, then $a_{o_i} = a_{\overline{o}_j} = a_{m_j}$ and $a_{\overline{o}_i} = a_{o_k} = a_{m_k}$.

For each transport operation $i \in I^R$ and each robot $r \in R$, let processing duration $d^{\mathrm{p}}(i, r) = |a_{o_i} - a_{\overline{o}_i}|/v_r$ be the minimum duration of its transport step on r, namely the time needed by r to cover the transport distance at maximum speed.

For any two distinct operations i, $j \in I^R$ and $r \in R$, if both i and j are executed on robot r and j immediately follows i on r, a "setup" of duration $d^{\mathrm{s}}(i, r; j, r)$ occurs on robot r, corresponding to the minimum duration of the idle move of r from the location of the hand-over step $\overline{o}_i$ of i to the location of the take-over step o_j of j, i.e. $d^{\mathrm{s}}(i, r; j, r) = |a_{\overline{o}_i} - a_{o_j}|/v_r$.

Finally, for each $i \in I^R$ and $r \in R$, the initial and final setup times are defined as $d^{\mathrm{s}}(\sigma; o_i, m) = |a_{\sigma r} - a_{o_i}|/v_r$, $d^{\mathrm{s}}(\sigma; \overline{o}_i, m) = |a_{\sigma r} - a_{\overline{o}_i}|/v_r$, and similarly, $d^{\mathrm{s}}(o_i, m; \tau) = |a_{o_i} - a_{\tau r}|/v_r$, $d^{\mathrm{s}}(\overline{o}_i, m; \tau) = |a_{\overline{o}_i} - a_{\tau r}|/v_r$, the minimum time needed by r to cover the distance from its initial location to the location of the take-over o_i and hand-over $\overline{o}_i$, respectively from the location of the take-over o_i and hand-over $\overline{o}_i$ to its end location.

A few standard assumptions concerning the data are made. Maximum robot speeds are positive, and, since we allow a transport operation to be executed by *any* robot, enough machine-free space on the rail left and right should be available: $(|R| - 1)\delta \leq \min\{a_m : m \in M\}$ and $\max\{a_m : m \in M\} \leq L-$ $(|R| - 1)\delta$ must hold.

7.3. A First Problem Formulation

In this section, we formulate the BJS-RT by using our problem formulation of the BJS-T in Section 6.4 and extending it to take into account the interferences between robots.

7.3.1. The Flexible Blocking Job Shop Relaxation

We temporarily ignore the interferences between the robots on the rail. This relaxed BJS-RT is then an instance of the BJS-T, which is discussed in Section 6.4.

Figure 7.3 depicts the disjunctive graph G of the example. For clarity however, nodes σ and τ, as well as all disjunctive arcs, except two pairs denoted by $e, \overline{e}$ and $e', \overline{e}'$, have been omitted. A pair of synchronization arcs is represented by an undirected edge.

Now consider feasible selections in the disjunctive graph G of the relaxed BJS-RT. A feasible selection (μ, S) specifies with μ the assignment of robots to the transport operations and with S the sequencing of the operations on the machines and robots. Note that each operation $i \in I^M$ is executed on a preassigned machine $m_i \in M'$, hence in any mode $\mu \in \mathcal{M}, \mu(i) = m_i$ for all $i \in I^M$.

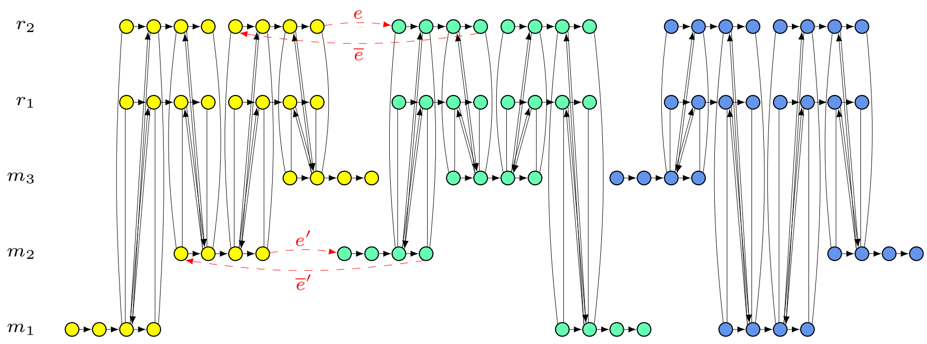


Figure 7.3.: Disjunctive graph of the example.

For any mode μ, let $\mathcal{F}^\mu = \{S \subseteq E^\mu : (\mu, S) \text{ is feasible}\}$. Given $\mu \in \mathcal{M}$ and $S \in \mathcal{F}^\mu$, any $\alpha = (\alpha_v : v \in V^\mu)$ satisfying:

$$\alpha_w - \alpha_v \geq d(v, w) \text{ for all arcs } (v, w) \in A^\mu \cup S \tag{7.1}$$

$$\alpha_\sigma = 0 \tag{7.2}$$

specifies starting times for the events corresponding to the nodes of V^μ. α_τ is the makespan and for any operation $i \in I$ and $v = v^1_{i,\mu(i)}, v^2_{i,\mu(i)}, v^3_{i,\mu(i)}, v^4_{i,\mu(i)}$, α_v is the starting and completion time of its take-over step o_i and the starting and completion time of its hand-over step $\bar{o}_i$. Note that the lag between the starting time and the completion time of all these transfer steps is exactly the transfer time. Let

$$\Omega(\mu, S) = \{\alpha \in \mathbb{R}^{V^\mu} : \alpha \text{ satisfies } (7.1) - (7.2)\}$$
$$\Omega(\mu) = \cup_{S \in \mathcal{F}^\mu} \Omega(\mu, S).$$

Definition 21 *The solution space of the relaxed BJS-RT is* $\Omega = \{(\mu, \alpha) : \mu \in \mathcal{M}, \alpha \in \Omega(\mu)\}$. *Any* $(\mu, \alpha) \in \Omega$ *is called a schedule.*

The relaxed BJS-RT is the problem of finding a schedule $(\mu, \alpha) \in \Omega$ minimizing α_τ. Note that, given $\mu \in \mathcal{M}$ and $S \in \mathcal{F}^\mu$, finding a schedule α minimizing the makespan is finding $\alpha \in \Omega(\mu, S)$ minimizing α_τ. As is well-known, this is easily done by longest path computation in $(V^\mu, A^\mu \cup S, d)$ and letting α_v be the length of a longest path from σ to v for all $v \in V^\mu$ (see also Section 2.3). The relaxed BJS-RT can therefore also be formulated as: Among all feasible selections, find a selection (μ, S) minimizing the length of a longest path from σ to τ in $(V^\mu, A^\mu \cup S, d)$.

7.3.2. Schedules with Trajectories

Not every schedule $(\mu, \alpha) \in \Omega$ is feasible in the BJS-RT. Indeed, due to the interference of the robots with each other, there might not exist feasible trajectories $x(r, .)$, $r \in R$, that "meet" the schedule. Given $(\mu, \alpha) \in \Omega$, we now examine which constraints the trajectories must satisfy in order to be feasible.

First, since the robots $r \in R$ cannot pass each other on the rail, it is convenient to index them $r_1, r_2, \ldots, r_K, K = |R|$, according to their natural ordering on the rail, with their locations at any time t satisfying $x(r_1, t) < x(r_2, t) < \ldots < x(r_K, t)$. From now on, for ease of notation, reference to robot r_k will be made simply through its index k, e.g. $x(r_k, t)$ is denoted by $x(k, t)$ and the maximum speed v_{r_k} by v_k.

Second, main input data for the trajectories are the locations, starting times and durations of the transfer steps (take-over or hand-over steps) of all transport operations. For $k = 1, \ldots, K$, let $O_k = \{o_i, \bar{o}_i : i \in I^R \text{ with } \mu(i) = k\}$ be the set of transfer steps executed by robot k. The location of a transfer step $o \in O_k$ is denoted by a_o, its starting time by $\alpha(o)$ and its duration by $d(o)$. Note that these data are all determined by the schedule (μ, α), e.g. if $o = o_i \in O_k$ for some $i \in I^R$, then $a_o = a_{o_i}$, $\alpha(o) = \alpha_{v^1_{ik}}$ and $d(o) = d^{\mathrm{t}}(j, m_j; i, k)$, where i is in some job J and j is the machining operation preceding i in J. It is convenient to add to O_k a fictive initial and final transfer step σ_k and τ_k, both of duration 0, and respective locations the prescribed initial and final locations $a_{\sigma k}$ and $a_{\tau k}$, and starting times 0 and α_τ. Denote again by O_k the so extended set and let $O = \bigcup_k O_k$.

Feasible trajectories $x(k, .)$, $k = 1, \ldots, K$, must satisfy the following constraints:

$$|x(k, t') - x(k, t)| \leq (t' - t)v_k \text{ for all } k = 1, \ldots, K \text{ and } t' > t \geq 0 \tag{7.3}$$

$$x(k, t) = a_o \text{ for all } k = 1, .., K,\ o \in O_k \text{ and } t \text{ with } \alpha(o) \leq t \leq \alpha(o) + d(o) \tag{7.4}$$

$$x(k, t) + \delta \leq x(k + 1, t) \text{ for all } k = 1, \ldots, K - 1 \text{ and } t \geq 0 \tag{7.5}$$

$$0 \leq x(1, t) \text{ and } x(K, t) \leq L \text{ for all } t \geq 0 \tag{7.6}$$

(7.3) expresses that a robot cannot cover a greater distance than allowed by its maximum speed. (7.4) enforces that a robot is at a_o while it executes the transfer step o. (7.5) maintains a minimum distance δ between two adjacent robots, while (7.6) restricts the moves of the robots to the interval $[0, L]$.

Given a schedule $(\mu, \alpha) \in \Omega$, let

$$X\,(\mu, \alpha) = \{\mathbf{x} = (x(k, .), k = 1, \ldots, K) : \mathbf{x} \text{ satisfies (7.3) to (7.6)}\}$$

Definition 22 *The solution space of the BJS-RT is* $\Gamma = \{(\mu, \alpha, \mathbf{x}) : (\mu, \alpha) \in \Omega$ *and* $\mathbf{x} \in X\,(\mu, \alpha)\}$. *Any* $(\mu, \alpha, \mathbf{x}) \in \Gamma$ *is called a* schedule with trajectories. *The BJS-RT is the problem of finding a schedule with trajectories minimizing* α_τ.

7.4. A Compact Problem Formulation

The objective in this section is to transform the BJS-RT into a “pure” scheduling problem, i.e. we derive a formulation whose decision variables involve only starting times and robot assignments and whose constraints ensure the existence of feasible trajectories.

Since the objective function in the BJS-RT depends only on α, a more compact formulation is obtained in principle by projecting Γ onto the space of the schedules. Letting

$$\begin{aligned}\Gamma_{\text{proj}} &= \{(\mu,\alpha) : \exists\ (\mu,\alpha,\mathbf{x}) \in \Gamma\}\\ &= \{(\mu,\alpha) : (\mu,\alpha) \in \Omega \text{ and } X\,(\mu,\alpha) \neq \emptyset\},\end{aligned}$$

the BJS-RT is then the problem of finding a schedule $(\mu,\alpha) \in \Gamma_{\text{proj}}$ minimizing α_τ.

The usefulness of this formulation depends on how the condition $X\,(\mu,\alpha) \neq \emptyset$ can be expressed more adequately and trajectories $\mathbf{x} = (x(k,.), k = 1,\dots,K) \in X\,(\mu,\alpha)$ can be determined efficiently.

7.4.1. The Feasible Trajectory Problem

Definition 23 *Given a schedule* $(\mu,\alpha) \in \Omega$, *the* feasible trajectory problem (FTP) at (μ,α) *is the problem of determining trajectories* $\mathbf{x} = (x(k,.), k = 1,\dots,K) \in X\,(\mu,\alpha)$ *or establishing* $X\,(\mu,\alpha) = \emptyset$.

We characterize when the FTP at (μ,α) has a feasible solution, i.e. $X\,(\mu,\alpha) \neq \emptyset$, and define for this purpose the following *discrete version* of the FTP at (μ,α).

Consider the set $\mathcal{Q} = \{\alpha(o), \alpha(o)+d(o) : o \in O\}$ of distinct starting and completion times of all transfer steps, and order $\mathcal{Q}$ such that $\mathcal{Q} = \{t_1, ..., t_Q\}$ with $Q \leq 2|O|$ and $t_1 < ... < t_Q$. Also, for any $0 \leq t \leq t'$, let $\mathcal{P}[t,t'] = \{p : 1 \leq p \leq Q \text{ and } t \leq t_p \leq t'\}$. For all $k = 1,...,K$, $p = 1,...,Q$, denote by $x_{kp} = x(k,t_p)$ the location of the robot k at t_p and consider the system:

$$|x_{k,p+1} - x_{kp}| \leq (t_{p+1} - t_p)v_k \text{ for all } k = 1,...,K,\ p = 1,...,Q-1, \tag{7.7}$$

$$x_{kp} = a_o \text{ for all } k = 1,...,K,\ o \in O_k \text{ and } p \in \mathcal{P}[\alpha(o), \alpha(o)+d(o)], \tag{7.8}$$

$$x_{kp} + \delta \leq x_{k+1,p} \text{ for all } k = 1,...,K-1,\ p = 1,...,Q, \tag{7.9}$$

$$0 \leq x_{1p} \text{ and } x_{Kp} \leq L \text{ for all } p = 1,...,Q. \tag{7.10}$$

(7.7) to (7.10) give a discrete version of the FTP at (μ,α) as the following holds.

Proposition 24 *i) For any* $(\widehat{x}(k,.)$: $k = 1,...,K)$ *satisfying (7.3) to (7.6),* $\widehat{x}_{kp} = \widehat{x}(k,t_p)$, $k = 1,...,K$, $p = 1,...,Q$ *satisfies (7.7) to (7.10).*
ii) For any $\widehat{x}_{kp}$, $k = 1,...,K$, $p = 1,...,Q$, *satisfying (7.7) to (7.10),*
$\widehat{x}(k,.)$, $k = 1,...,K$, *defined by:*

$$\widehat{x}(k,t_p) = \widehat{x}_{kp},\ p = 1,...,Q \text{ and} \tag{7.11}$$

$$\widehat{x}(k,t) = \frac{t_{p+1}-t}{t_{p+1}-t_p}\widehat{x}_{kp} + \frac{t-t_p}{t_{p+1}-t_p}\widehat{x}_{k,p+1},\ t_p < t < t_{p+1},\ p = 1,...,Q-1 \tag{7.12}$$

satisfies (7.3) to (7.6).

Proof. i) is obvious. ii) is easily proven by observing that with (7.11) and (7.12), the trajectory $\widehat{x}(k,.)$, $k \in \{1,...,K\}$, is simply obtained by joining in the time-location space each pair of consecutive points $(t_p, \widehat{x}_{kp})$, $(t_{p+1}, \widehat{x}_{k,p+1})$ by a line segment. ■

Now consider two transfer steps o and o' that are executed by distinct cranes k and k' with $o \in O_k$ and $o' \in O_{k'}$. Without loss of generality, we may assume that $k < k'$. By constraints (7.9), crane k' has to be at any time at least $(k' - k)\delta$ above crane k. By constraints (7.8), crane k has to be at location a_o while executing transfer step o and crane k' has to be at $a_{o'}$ while o' is executed. Hence, if the transfer location $a_{o'}$ is not at least $(k' - k)\delta$ above the location a_o, i.e. if $a_{o'} - a_o < (k' - k)\delta$, then the two steps o, o' cannot occur simultaneously. Moreover, there must be a certain time lag between the executions of the two steps. This time lag can be specified as follows.

Definition 25 *For any two cranes k, k' with $1 \leq k < k' \leq K$, and any two transfer steps $o \in O_k$, $o' \in O_{k'}$, let*

$$\Delta_{oo'}^{kk'} = [(k' - k)\delta + a_o - a_{o'}]/\min\{v_l : k \leq l \leq k'\}. \tag{7.13}$$

Assume $\Delta_{oo'}^{kk'} > 0$. Then $a_{o'} - a_o < (k' - k)\delta$, so o and o' cannot occur simultaneously. Suppose o' is completed at time t' and o begins at time $t > t'$. In the interval $[t', t]$, robot k' needs to cover at least distance $(k' - k)\delta + a_o - a_{o'}$, and so do all robots l with $k \leq l \leq k'$. Similarly if $t < t'$, all robots l with $k \leq l \leq k'$ have to travel at least distance $(k' - k)\delta + a_o - a_{o'}$ in the interval $[t, t']$, hence

$$\alpha(o) + d(o) + \Delta_{oo'}^{kk'} \leq \alpha(o') \text{ or } \alpha(o') + d(o') + \Delta_{oo'}^{kk'} \leq \alpha(o).$$

Following a terminology in use, we call such a disjunctive constraint a collision avoidance constraint. Indeed, constraints of this type have been introduced e.g. by Manier et al. [74], Leung and Zhang [67] and Leung et al. [68] in the hoist scheduling problem. They establish necessity and sufficiency of these constraints by a case-by-case analysis of the various ways collisions between two hoists can occur. We show here necessity and sufficiency in the following lemma by identifying the discrete FTP as a network problem in a graph H and showing the equivalence of the collision avoidance constraints with the absence of negative cycles in H.

Lemma 26 *System (7.7) to (7.10) admits a solution if and only if for all $o \in O_k$, $o' \in O_{k'}$ with $k < k'$ and $\Delta_{oo'}^{kk'} > 0$:*

$$\alpha(o) + d(o) + \Delta_{oo'}^{kk'} \leq \alpha(o') \text{ or } \alpha(o') + d(o') + \Delta_{oo'}^{kk'} \leq \alpha(o) \tag{7.14}$$

Proof. Let $H = (W, B, c)$ be the following graph. Node set W consists of node w_* and $K \times Q$ nodes w_{kp}, $k = 1, ..., K$, $p = 1, ..., Q$. The arc set B and the valuation $c \in \mathbb{R}^B$ are given in the table below:

arcs of B	*weights c*	
$(w_{k+1,p}, w_{kp})$	$-\delta$	$k = 1, ..., K-1$, $p = 1, ..., Q$,
$(w_{kp}, w_{k,p+1})$ $(w_{k,p+1}, w_{kp})$	$(t_{p+1} - t_p)v_k$	$k = 1, ..., K$, $p = 1, ..., Q-1$,
(w_*, w_{kp}) (w_{kp}, w_*)	a_o $-a_o$	for all $k = 1, ..., K$, $o \in O_k$ and $p \in \mathcal{P}[\alpha(o), \alpha(o) + d(o)]$,
$(w_{1p}, w_*)^0$ $(w_*, w_{Kp})^L$	0 L	$p = 1, ..., Q$.

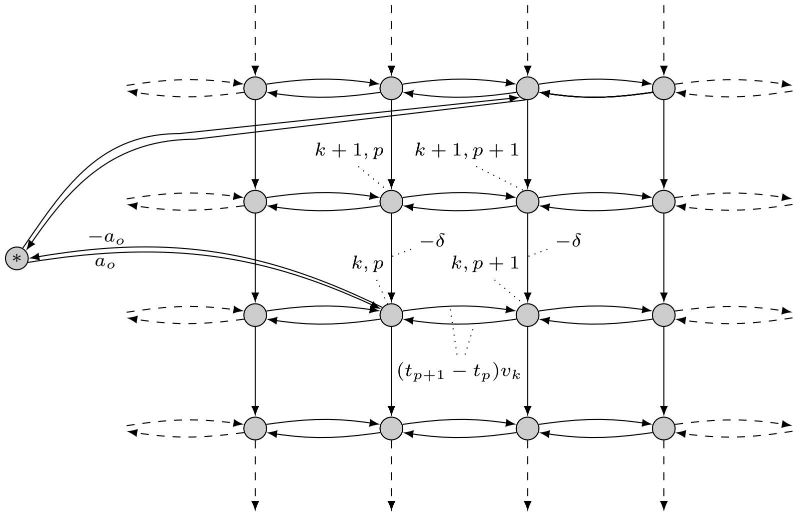


Figure 7.4.: Graph H. (Not all arcs are shown.)

Note that parallel arcs are present, explaining the indexing 0 and L in $(w_{1p}, w_*)^0$ and $(w_*, w_{Kp})^L$. Graph H is depicted in Figure 7.4.

It is easy to see that the system (7.7) to (7.10) is equivalent to the following system of inequalities in graph $H = (W, B, c)$:

$$x_w - x_v \leq c_{vw} \text{ for all } (v, w) \in B, \tag{7.15}$$

$$x_{w_*} = 0, \tag{7.16}$$

i.e. x satisfying (7.15) and (7.16) is a *feasible potential function.* By a well-known result of combinatorial optimization (see e.g. Cook et al. [26], p. 25), $H = (W, B, c)$ admits a feasible potential function - and hence (7.7) to (7.10) admits a feasible solution - if and only if no cycle of negative length exists in H.

We prove therefore that constraints (7.14) hold if and only if H has no negative cycle.

First, the following observations are useful. Consider the graph H^- obtained from H by deleting node w_*. H^- contains no cycle of negative length. Also, there exists a path in H^- from a node $w_{k'p'}$ to a node w_{kp} if and only if $k \leq k'$. Finally, it is easy to see that a shortest path in H^- from $w_{k'p'}$ to w_{kp} has length $|t_{p'} - t_p| \cdot \min\{v_l : k \leq l \leq k'\} - (k' - k)\delta$.

i) Now suppose that (7.14) does not hold: there are $o \in O_k$, $o' \in O_{k'}$, with $k' > k$ and $\Delta_{oo'}^{kk'} > 0$, such that $\Delta_{oo'}^{kk'} > \alpha(o') - \alpha(o) - d(o)$ and $\Delta_{oo'}^{kk'} > \alpha(o) - \alpha(o') - d(o')$. If o and o' are both in execution at some time $t_p, p \in \{1, \ldots, Q\}$, then the cycle Z in H consisting of arc $(w_*, w_{k'p})$, a shortest path in H^- from $w_{k'p}$ to w_{kp} and arc (w_{kp}, w_*) has length

$$c(Z) = a_{o'} - (k' - k)\delta - a_o = -\Delta_{oo'}^{kk'} \cdot \min\{v_l : k \leq l \leq k'\},$$

hence $c(Z) < 0$. If o is executed before o', i.e. $\alpha(o) + d(o) \leq \alpha(o')$, let p and p' be such that $t_p = \alpha(o) + d(o)$ and $t_{p'} = \alpha(o')$. Then the cycle Z in H consisting of arc $(w_*, w_{k'p'})$, a shortest path in H^- from $w_{k'p'}$ to w_{kp} and arc (w_{kp}, w_*) has length

$$\begin{aligned} c(Z) &= a_{o'} + |t_{p'} - t_p| \cdot \min\{v_l : k \leq l \leq k'\} - (k' - k)\delta - a_o \\ &= [\alpha(o') - \alpha(o) - d(o) - \Delta_{oo'}^{kk'}] \cdot \min\{v_l : k \leq l \leq k'\} < 0. \end{aligned}$$

Finally, if o' is executed before o, i.e. $\alpha(o') + d(o') \leq \alpha(o)$, the existence of a negative cycle is shown similarly.

ii) Conversely, suppose H has a cycle Z of negative length. By the preceding observations, Z must pass through node w_*, leaving w_* by an arc b' with head $w_{k'p'}$ and entering w_* by an arc b with tail w_{kp} for some $k \leq k'$ and p, p'. Also, we may assume that Z takes a shortest path in H^- from $w_{k'p'}$ to w_{kp}. Hence Z has length

$$c(Z) = c_{b'} + c_b + |t_{p'} - t_p| \cdot \min\{v_l : k \leq l \leq k'\} - (k' - k)\delta < 0.$$

First, we exclude the following three cases for b' and b. Case a): $b' = (w_*, w_{Kp'})^L$ and $b = (w_{1p}, w_*)^0$. We may assume $p' = p$ since $(w_{1p}, w_*)^0 \in B$ with same weight 0, so that $c(Z) = L + 0 - (K - 1)\delta < 0$, violating the standard assumption $\max\{a_m : m \in M\} \leq L-$ $(K - 1)\delta$. Case b): $b' = (w_*, w_{Kp'})^L$ and $b = (w_{kp}, w_*)$ with weight $-a_o$. We may assume $p' = p$ since $(w_*, w_{Kp'})^L \in B$ with same weight L, so that $c(Z) = L - a_o - (K - k)\delta < 0$, in contradiction to $\max\{a_m : m \in M\} \leq L-$ $(K - 1)\delta$. Case c) $b' = (w_*, w_{k'p'})$ with weight $a_{o'}$ and $b = (w_{1p}, w_*)^0$. We may assume $p = p'$, so that $c(Z) = a_{o'} - (k' - 1)\delta < 0$, contradicting the assumption $(K - 1)\delta \leq \min\{a_m : m \in M\}$.

Therefore arc b' is $(w_*, w_{k'p'})$ for some $o' \in O_{k'}$ and $\alpha(o') \leq t_{p'} \leq \alpha(o') + d(o')$, and arc b is (w_{kp}, w_*) for some $o \in O_k$ and $\alpha(o) \leq t_p \leq \alpha(o) + d(o)$. The case $k = k'$ can be excluded, using the fact that for any two $o, o' \in O_k$, with say, $\alpha(o) \leq \alpha(o')$, $\alpha(o') - (\alpha(o) + d(o)) \geq |a_{o'} - a_o|$ holds. Therefore $k < k'$ and the length of Z is

$$\begin{aligned} c(Z) &= a_{o'} - a_o + |t_{p'} - t_p| \cdot \min\{v_l : k \leq l \leq k'\} - (k' - k)\delta \\ &= [|t_{p'} - t_p| - \Delta_{oo'}^{kk'}] \cdot \min\{v_l : k \leq l \leq k'\} < 0 \end{aligned}$$

Therefore $|t_{p'} - t_p| - \Delta_{oo'}^{kk'} < 0$, so that $\Delta_{oo'}^{kk'} > 0$, and both $\Delta_{oo'}^{kk'} > t_{p'} - t_p$ and $\Delta_{oo'}^{kk'} > t_p -$ $t_{p'}$ hold. Then $\Delta_{oo'}^{kk'} > t_{p'} - t_p \geq \alpha(o') - \alpha(o) - d(o)$ and $\Delta_{oo'}^{kk'} > t_p -$ $t_{p'} \geq \alpha(o) - \alpha(o') - d(o')$, so that (7.14) is violated for this pair o, o'. ■

Assuming that the FTP at (μ, α) has a feasible solution, trajectories $\mathbf{x} = (x(k, .)$, $k = 1, \ldots, K)$ can be determined by finding a potential function in H - an elementary task in network flows - and applying Proposition 24. Furthermore, a natural objective is to find trajectories minimizing the total distance traveled by the robots. It is easy to define a network problem in an adapted graph H that finds a potential optimizing this objective and then apply Proposition 24. However, optimal trajectories can also be determined more efficiently with an algorithm based on geometric arguments, as will be shown in Section 7.7.

7.4.2. Projection onto the Space of Schedules

A compact disjunctive graph formulation of the BJR-RT is now readily obtained by introducing in $G = (V, A, E, \mathcal{E}, d)$ additional conjunctive and disjunctive arcs to take into account constraints (7.14) for any mode.

First, observe that each transfer step of a transport operation on a given robot is represented by a specific arc in G. Indeed, a transfer step o on robot k is either o_i or $\overline{o}_i$ for some $i \in I^R$ executed by k; o_i on k is represented in G by the arc (v^1_{ik}, v^2_{ik}) and $\overline{o}_i$ on k by (v^3_{ik}, v^4_{ik}).

Second, conflicts of a transfer step o of a transport operation executed by a robot k with the fictive initial and final transfer steps $\sigma_{k'}$ and $\tau_{k'}$ of the robots $k' \neq k$, simply result in an initial setup time and a final set up time for o on k. Also, a conflict between two transfer steps o, o' (of distinct transport operations) of a *same* job simply results in a precedence constraint.

Conjunctive arcs are now added to A and disjunctive arcs are added to E, respectively arc pairs to $\mathcal{E}$, as specified in the three steps below, convening that arcs *are added only if their weight is positive*:

1. For all $o \in \{o_i, \overline{o}_i : i \in I^R\}$ and all k, $1 \leq k \leq K$, if (v, w) represents o on k, add to A the arc (σ, v) with weight $\Delta^{\cdot k}_{\sigma o} = \max\{0; \Delta^{kk'}_{o\sigma_{k'}} : k < k'; \Delta^{k'k}_{\sigma_{k'}o} : k > k'\}$, and (w, τ) with weight $\Delta^{k\cdot}_{o\tau} = \max\{0; \Delta^{kk'}_{o\tau_{k'}} : k < k'; \Delta^{k'k}_{\tau_{k'}o} : k > k'\}$. (If an arc is added that is parallel to an arc already present, retain only the arc with largest weight.)
2. For each $o, o' \in \{o_i, \overline{o}_i : i \in I^R\}$ where o and o' are transfer steps of *distinct* transport operations of a *same* job, assuming without loss of generality that o precedes o', for each $k \neq k'$, if (v, w) and (v', w') represent o on k and o' on k', add to A the arc (w, v') with weight $\Delta^{kk'}_{oo'}$ if $k' > k$ and $\Delta^{k'k}_{o'o}$ if $k' < k$.
3. For all $o, o' \in \{o_i, \overline{o}_i : i \in I^R\}$ where o and o' are transfer steps of *distinct* jobs, and all k, k' with $1 \leq k < k' \leq K$, if (v, w) and (v', w') represent o on k and o' on k', add to E, respectively to $\mathcal{E}$, the pair of arcs $(w, v'), (w', v)$, both of weight $\Delta^{kk'}_{oo'}$.

Denote by $G' = (V, A', E', \mathcal{E}', d')$ the disjunctive graph thus obtained. Figure 7.5 depicts G' in the example, obtained by adding in G of Figure 7.3 conjunctive and disjunctive arcs as described above. For sake of clarity, only two additional arcs f and g from step 1, arc h from step 2 and disjunctive arc pair $e, \bar{e}$ from step 3 are displayed. The weights of e, $\bar{e}$, f, g and h are 3, 3, 1, 2 and 1. In the example, G contains altogether 30 disjunctive arcs pairs, and 24 arcs in step 1, 15 arcs in step 2 and 63 disjunctive arc pairs in step 3 are added to obtain G'.

Define in G' modes, (complete, acyclic, feasible) selections, and $\mathcal{F}'^{\mu}$, $\Omega'(\mu, S')$,$\Omega'(\mu)$ and Ω' similarly to the corresponding definitions in G given in Sections 2.4 and 7.3.1.

Theorem 27 *The projection Γ_{proj} of the set of schedules with trajectories (defined in G) is precisely the set of schedules Ω' defined in G'.*

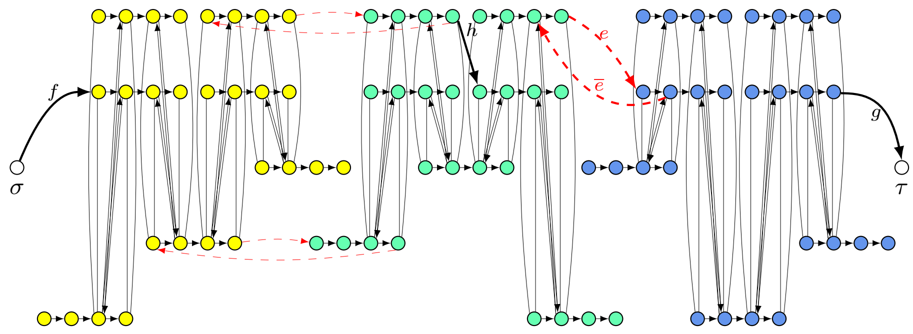


Figure 7.5.: Some collision avoidance arcs (e, $\bar{e}$, f, g and h) in the example.

Proof. i) The FTP at (μ,α) in G admits a feasible solution, i.e. $X(\mu,\alpha) \neq \emptyset$, if and only the constraints (7.14) hold. Indeed, by Proposition 24 the FTP at (μ,α) admits a feasible solution if and only if its discrete version (7.7) to (7.10) admits a feasible solution, hence by Lemma 26, if and only if (7.14) holds. Therefore $\Gamma_{\text{proj}} = \{(\mu,\alpha) : \mu \in \mathcal{M},\ \alpha \in \Omega(\mu) \text{ and (7.14) holds}\}$.

ii) Let $(\mu,\alpha) \in \Omega' = \{(\mu,\alpha) : \mu \in \mathcal{M},\ \alpha \in \Omega'(\mu)\}$, hence $\alpha \in \Omega'(\mu,S')$ for some $S' \in \mathcal{F}'^{\mu}$. Then for $S = S' \cap E \in \mathcal{F}^{\mu}$ and, since $\Omega'(\mu,S') \subseteq \Omega(\mu,S)$, $\alpha \in \Omega(\mu,S)$. Therefore $\alpha \in \Omega(\mu)$. Also, the constraints (7.1)' for $(v,w) \in A' \cup S' - A \cup S$ ensure that (7.14) is satisfied by α. Hence $\Omega' \subseteq \Gamma_{\text{proj}}$. Conversely, let $(\mu,\alpha) \in \Gamma_{\text{proj}}$. There exists $S \in \mathcal{F}^{\mu}$ such that $\alpha \in \Omega(\mu,S)$ and α satisfies (7.14). Then $S' = S \cup \{(v,w) \in E' - E : \alpha_v \leq \alpha_w\} \in \mathcal{F}'^{\mu}$ and $\alpha \in \Omega'(\mu,S')$, hence $\alpha \in \Omega'(\mu)$ and $\Gamma_{\text{proj}} \subseteq \Omega'$. ■

The BJS-RT can therefore be formulated as follows:

Among all feasible selections in G', find a selection (μ,S') minimizing the length of a longest path from σ to τ in $(V^{\mu}, A'^{\mu} \cup S', d')$.

7.5. The BJS-RT as an Instance of the CJS Model

As discussed in Section 6.4, the BJS-T can be formulated as an instance of the CJS model. Hence, in order to formulate the BJS-RT in the CJS model it is sufficient to show how to represent the additional conjunctive and disjunctive arcs introduced in steps 1) to 3) in Section 7.4.2.

Arcs introduced in step 1) represent initial and final setup times. In order to incorporate the initial setups, we update the duration $d^{\text{s}}(\sigma;o,k)$ to $\Delta^{\cdot k}_{\sigma o}$ if $d^{\text{s}}(\sigma;o,k) < \Delta^{\cdot k}_{\sigma o}$ in the BJS-T instance, i.e. the duration is updated if the initial setup without considering collision avoidance is smaller than $\Delta^{\cdot k}_{\sigma o}$. Similarly, update the duration $d^{\text{s}}(o,m;\tau)$ to $\Delta^{k\cdot}_{o\tau}$ if $d^{\text{s}}(o,m;\tau) < \Delta^{k\cdot}_{o\tau}$ in the BJS-T instance.

The collision avoidance arcs introduced in step 2) and 3) can be integrated in the CJS model in a straightforward manner by specifying the set of conflicting transfer steps $\mathcal{V}$ as follows.

For all $o, o' \in \{o_i, \overline{o}_i : i \in I^R\}$ where o and o' are transfer steps of *distinct* transport operations of a *same* job, assuming without loss of generality that o precedes o', for each $k \neq k'$, add $\{(o,k),(o',k')\}$ to $\mathcal{V}$ and set both weights $d^{\mathrm{s}}(o,k;o',k') = d^{\mathrm{s}}(o',k';o,k)$ to $\Delta^{kk'}_{oo'}$ if $k' > k$ and to $\Delta^{k'k}_{o'o}$ if $k' < k$.

For all $o, o' \in \{o_i, \overline{o}_i : i \in I^R\}$ where o and o' are transfer steps of *distinct* jobs, and all k, k' with $1 \leq k < k' \leq K$, add $\{(o,k),(o',k')\}$ to $\mathcal{V}$ and set both weights $d^{\mathrm{s}}(o,k;o',k') = d^{\mathrm{s}}(o',k';o,k)$ to $\Delta^{kk'}_{oo'}$.

As a result, the BJS-RT belongs to the class of CJS problems without time lags, so the JIBLS can be applied.

7.6. Computational Results

The tabu search (with neighborhood $\mathcal{N}$) described in Section 3.5 was implemented single-threaded in Java for the BJS-RT. It was run on a PC with 3.1 GHz Intel Core i5-2400 processor and 4 GB memory. Since the BJS-RT has not been addressed in the literature, we created a test set of 80 instances starting from the standard job shop instances la01 to la20 introduced by Lawrence [64] and adding data to describe the transportation system.

For each Lawrence instance lapq and number of robots $K = 1, \ldots, 4$, a BJS-RT instance was generated as follows. The location of machine $m_i, i = 0, \ldots, |M| - 1$ is $a_{m_i} = 120 + 50i$, where $|M|$ is the number of machines and i is the number attributed by Lawrence. Initial and final locations of robot $k = 1, \ldots, K$ are $a_{\sigma k} = a_{\tau k} = 40(k-1)$ and the maximum speed is $v_k = 10$. The minimum distance between adjacent robots is $\delta = 40$, and the rail length is $L = 120 + 50(|M| - 1) + \delta(K - 1)$. Transfer times are $d^{\mathrm{t}}(i, m_i; j, k) = d^{\mathrm{t}}(j, k; i, m_i) = 10$ for all $i \in I^M, j \in I^R, r \in R$, and loading time $d^{\mathrm{ld}}(i, m) = 10$ for all operations $i \in I^{\mathrm{first}}$ and unloading time $d^{\mathrm{ul}}(i, m) = 10$ for all $i \in I^{\mathrm{last}}$. Set-up times are $d^{\mathrm{s}}(i, m_i; j, m_j) = 0$ for distinct $i, j \in I^M$ with $m_i = m_j$, and initial and final setup times are $d^{\mathrm{s}}(\sigma; o_i, m) = d^{\mathrm{s}}(\sigma; \overline{o}_i, m) = d^{\mathrm{s}}(o_i, m; \tau) = d^{\mathrm{s}}(\overline{o}_i, m; \tau) = 0$ for all $i \in I^M$.

With la01 to la20, the problem sizes in the test set are 10×5 (10 jobs on 5 machines), 15×5, 20×5 and 10×10. These sizes might appear modest at first glance, but should be used with caution when comparing the BJS-RT for instance with the classical job shop problem for which la01 to la20 were originally introduced. In a BJS-RT, an $m \times n$ instance contains nearly twice the number of operations since for each job with m (machining) operations, $m - 1$ transport operations are introduced. Moreover, there are typically many collision avoidance constraints between the $2(m - 1)n$ transfer steps. Finally, flexibility in choosing a robot further increases complexity.

The computation settings were chosen similarly as in the FBJSS, Chapter 5. The initial solution was a permutation schedule generated by randomly choosing a job permutation and a mode. For each instance, five independent runs with different initial solutions were performed. The computation time of a run was limited to 1800 seconds. The following parameter values were chosen: $maxt = 12$, $maxl = 300$ and $maxiter = 3000$

# robots	*1 robot*		*2 robots*		*3 robots*		*4 robots*	
instance	*best*	*avg*	*best*	*avg*	*best*	*avg*	*best*	*avg*
10 × 5								
la01	1736	1746	1315	1356	1155	1196	1108	1136
la02	1727	1727	1329	1353	1203	1222	1155	1171
la03	1695	1695	1262	1284	1089	1124	1044	1104
la04	1748	1749	1280	1299	1140	1165	1044	1089
la05	1654	1655	1251	1270	1111	1130	1057	1099
15 × 5								
la06	2465	2478	1899	1974	1726	1760	1655	1693
la07	2473	2496	1964	1990	1707	1764	1638	1659
la08	2483	2502	1912	1949	1748	1790	1675	1716
la09	2501	2520	2018	2056	1747	1813	1696	1718
la10	2529	2550	1968	2014	1777	1818	1692	1748
20 × 5								
la11	3381	3399	2640	2749	2349	2478	2424	2446
la12	3296	3326	2541	2696	2188	2251	2128	2262
la13	3335	3373	2624	2655	2364	2402	2192	2338
la14	3391	3419	2690	2823	2499	2604	2323	2433
la15	3353	3384	2723	2807	2385	2514	2216	2407
10 × 10								
la16	4664	4967	2907	3216	2652	2853	2392	2666
la17	4608	4776	3079	3340	2774	2968	2539	2826
la18	4655	4827	3304	3438	2699	2857	2476	2881
la19	4562	4683	3051	3299	2500	2757	2333	2662
la20	4710	4786	3019	3362	2736	2986	2360	2761

Table 7.1.: Best and average results over the five runs in the BJS-RT instances (time limit: 1800 seconds per run).

Table 7.1 provides detailed results. The first line splits the table into four groups according to the number of robots. Columns *best* and *avg* refer to the best and average results, respectively, of the five runs. The table is subdivided horizontally according to size of the instances, e.g. the first block reports on instances 10 × 5 with 10 jobs and 5 machines. We now discuss the results, evaluating solution quality, convergence behavior of the tabu search and impact of increasing the number of robots.

Since the BJS-RT has not yet been addressed in publications, a comparison of the obtained results with benchmarks was not possible. For this reason, we tried to assess the quality of the tabu search with results obtained via a MIP model that we derived from the disjunctive graph formulation. Instances la01 to la05 with 1 robot have been solved to optimality with a MIP model implemented in LPL [54] using the solver Gurobi 5.0 [46] and a time limit of five hours. However, with 2 robots, no feasible solution could even be found for la01 to la05. We reduced the size of these instances by keeping only the first six jobs (out of ten). These instances, called la01* to la05*, were solved by Gurobi, after providing the best solution found by the tabu search as an initial solution and allowing more computation time. Table 7.2 shows the results obtained for the instances with 1 robot (left) and 2 robots (right). Columns *result* give the optimal values or the upper and lower bounds (ub;lb) if optimality could not be established. Columns *time* give the computation time in seconds used

1 robot	*MIP*		tabu search	
instance	*result*	*time*	*best*	*avg*
la01	1736	3000	1736	1746
la02	(1727;1556)	18000	1727	1727
la03	1695	1638	1695	1695
la04	1748	7016	1748	1749
la05	(1654;1347)	18000	1654	1655

2 robots	*MIP*		tabu search	
instance	*result*	*time*	*best*	*avg*
*la01**	832	8840	832	835
*la02**	864	51892	864	864
*la03**	833	15636	833	833
*la04**	823	13225	823	823
*la05**	765	230228	765	765

Table 7.2.: MIP results and computation times compared to *best* and *avg* (average) results over the five runs (time limit: 1800 seconds per run) of the tabu search in the BJS-RT instances.

by Gurobi, and columns *best* and *avg* the results of the tabu search. The following observations can be made. As is the case in other complex scheduling problems, only small instances could be solved to optimality with a MIP approach and even finding a feasible solution appears to be a challenge in multiple robot instances. Now comparing the MIP and tabu search results, in all 10 instances, the best of the five runs reached the MIP optimum or upper bound, and all five runs yield results that are as good or very close. Albeit limited, these results suggest that the JIBLS performs adequately in BJS-RT instances.

Further support is found by examining the evolution of attained solution quality over computation time. For this purpose, the best makespan ω at the beginning (initial solution) and during the execution of the tabu search were recorded for each instance and run, and its (relative) deviation from the final solution $(\omega - \omega_{\text{final}})/\omega_{\text{final}}$ determined. Figure 7.6 illustrates these deviations for instances with 3 robots in an aggregated way, depicting average deviations (in %) over runs and instances of the same size. The following can be observed. Initial solutions are far from the obtained final solutions, with makespans twice to three times as large. Also, most of the improvements are found within minutes. Consider for example the 10×10 instances with 3 robots. While the deviation is initially 175.6%, it drops to 9.4% and 5.5% after 300 and 600 seconds.

Furthermore, we investigated the impact of adding a robot to the transport system. Information of this type may be of interest at the design stage, when capacity is calibrated or the benefit of installing additional equipment is assessed. We compared each instance with K robots with the same instance with $K-1$ robots by determining the relative change in the makespan $(avg_K - avg_{K-1})/avg_{K-1}$, where $avg_i, i = 1, \ldots, 4$, can be found in Table 7.1, column *avg* of group *i robots*. Table 7.3 (left) reports these changes in % in an aggregated way. As expected, adding a robot reduces the makespan, and this return diminishes with the number of robots. Going from 1 to 2 robots (column *2 robots*) reduces the makespan significantly, the range of the decrease being 18% to 31%. Adding a third robot yields a decrease of 10% to 14%, and adding a fourth robot, a decrease of 3% to 5%.

Finally, Table 7.3 (right) shows the number of tabu search iterations averaged over instances of the same size. With increasing number of robots and problem size, the

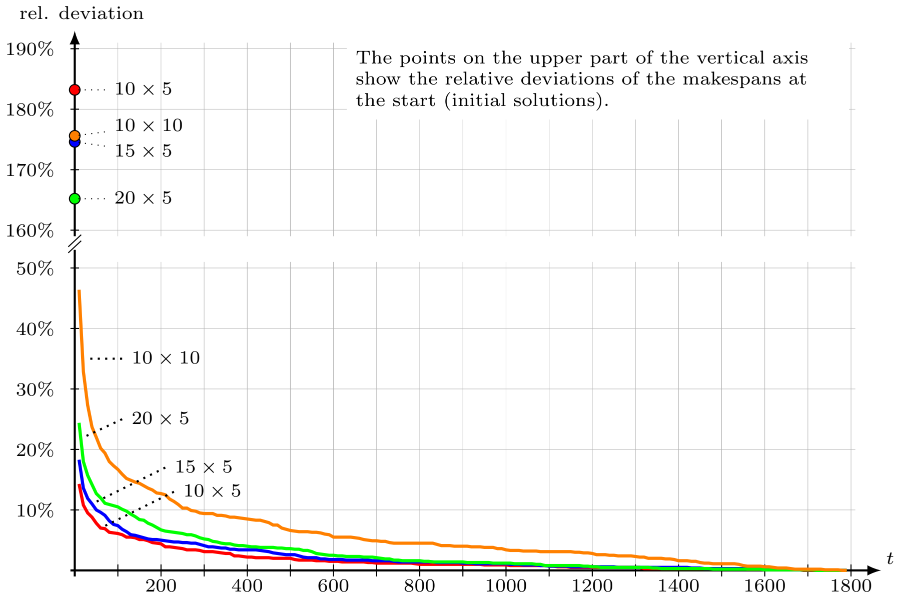


Figure 7.6.: Relative deviations of the makespan from the final makespan during runtime.

size	*2 robots*	*3 robots*	*4 robots*
10 × 5	-23.4%	-11.1%	-4.0%
15 × 5	-20.4%	-10.4%	-4.6%
20 × 5	-18.8%	-10.8%	-2.9%
10 × 10	-30.7%	-13.4%	-4.3%

size	1 robot	*2 robots*	*3 robots*	*4 robots*
10 × 5	632070	188973	153689	131235
15 × 5	664216	102871	83026	70769
20 × 5	429657	55872	48662	42766
10 × 10	467584	44353	36383	33824

Table 7.3.: (Left) Relative changes in the makespan when adding a robot in the BJS-RT instances. (Right) Number of tabu search iterations per run in the BJS-RT instances.

number of iterations drops drastically reflecting the increasing computation time per iteration. This is due to an increase of the neighborhood size and to the fact that the effort for generating a neighbor of type (3.3) is larger than for a neighbor of type (3.4) - (3.5) (about twice as large in our implementation).

The computational results are concluded with Figure 7.7 which displays a schedule with makespan 1155 for instance la01 with 3 robots. The transport operations are not shown, but can be inferred from the trajectories of the robots.

7.7. Finding Feasible Trajectories

In this section, we develop efficient methods to find feasible trajectories $\mathbf{x} = (x(k,.), k = 1, \ldots, K) \in X(\mu, \alpha)$ given some feasible schedule $(\mu, \alpha) \in \Gamma_{\text{proj}}$. In Subsection 7.7.1, we build trajectories that make use of the variable speed (up to its maximum v_k) of each robot k. If all v_k's are equal, say v, it is possible to adapt the trajectories so that a robot is either still ("stop") or moves at speed v ("go"). In Subsection 7.7.2, we show how to generate these so-called stop-and-go trajectories.

7.7.1. Trajectories with Variable Speeds

Finding feasible trajectories with minimum total travel distance is the problem of minimizing $\sum_{k=1}^{K} \sum_{p=1}^{Q-1} |x_{k,p+1} - x_{kp}|$ subject to constraints (7.7) to (7.10). It can be expressed by the linear program

$$\text{Minimize } \sum_{k=1}^{K} \sum_{p=1}^{Q-1} (y_{kp}^{+} + y_{kp}^{-}) \tag{7.17}$$

subject to

$$x_{k,p+1} - x_{kp} \leq (t_{p+1} - t_p)v_k \text{ for all } k = 1, \ldots, K,\ p = 1, \ldots, Q-1, \tag{7.18}$$

$$x_{kp} - x_{k,p+1} \leq (t_{p+1} - t_p)v_k \text{ for all } k = 1, \ldots, K,\ p = 1, \ldots, Q-1, \tag{7.19}$$

$$x_{kp} - x_* \leq a_p \text{ for all } k = 1, \ldots, K,\ o \in O_k \text{ and } p \in \mathcal{P}[\alpha(o), \alpha(o) + d(o)], \tag{7.20}$$

$$x_* - x_{kp} \leq -a_p \text{ for all } k = 1, \ldots, K,\ o \in O_k \text{ and } p \in \mathcal{P}[\alpha(o), \alpha(o) + d(o)], \tag{7.21}$$

$$x_{kp} - x_{k+1,p} \leq -\delta \text{ for all } k = 1, \ldots, K-1,\ p = 1, \ldots, Q, \tag{7.22}$$

$$x_* - x_{1p} \leq 0 \text{ for all } p = 1, \ldots, Q, \tag{7.23}$$

$$x_{Kp} - x_* \leq L \text{ for all } p = 1, \ldots, Q, \tag{7.24}$$

$$x_* = 0, \tag{7.25}$$

$$x_{k,p+1} - x_{kp} - y_{kp}^{+} \leq 0 \text{ for all } k = 1, \ldots, K,\ p = 1, \ldots, Q-1, \tag{7.26}$$

$$x_{kp} - x_{k,p+1} - y_{kp}^{-} \leq 0 \text{ for all } k = 1, \ldots, K,\ p = 1, \ldots, Q-1, \tag{7.27}$$

$$y_{kp}^{+} \geq 0,\ y_{kp}^{-} \geq 0 \text{ for all } k = 1, \ldots, K,\ p = 1, \ldots, Q-1. \tag{7.28}$$

The above linear program is the dual of a minimum cost circulation problem. It can be solved e.g. by a primal-dual algorithm for the minimum cost flow problem (see e.g.

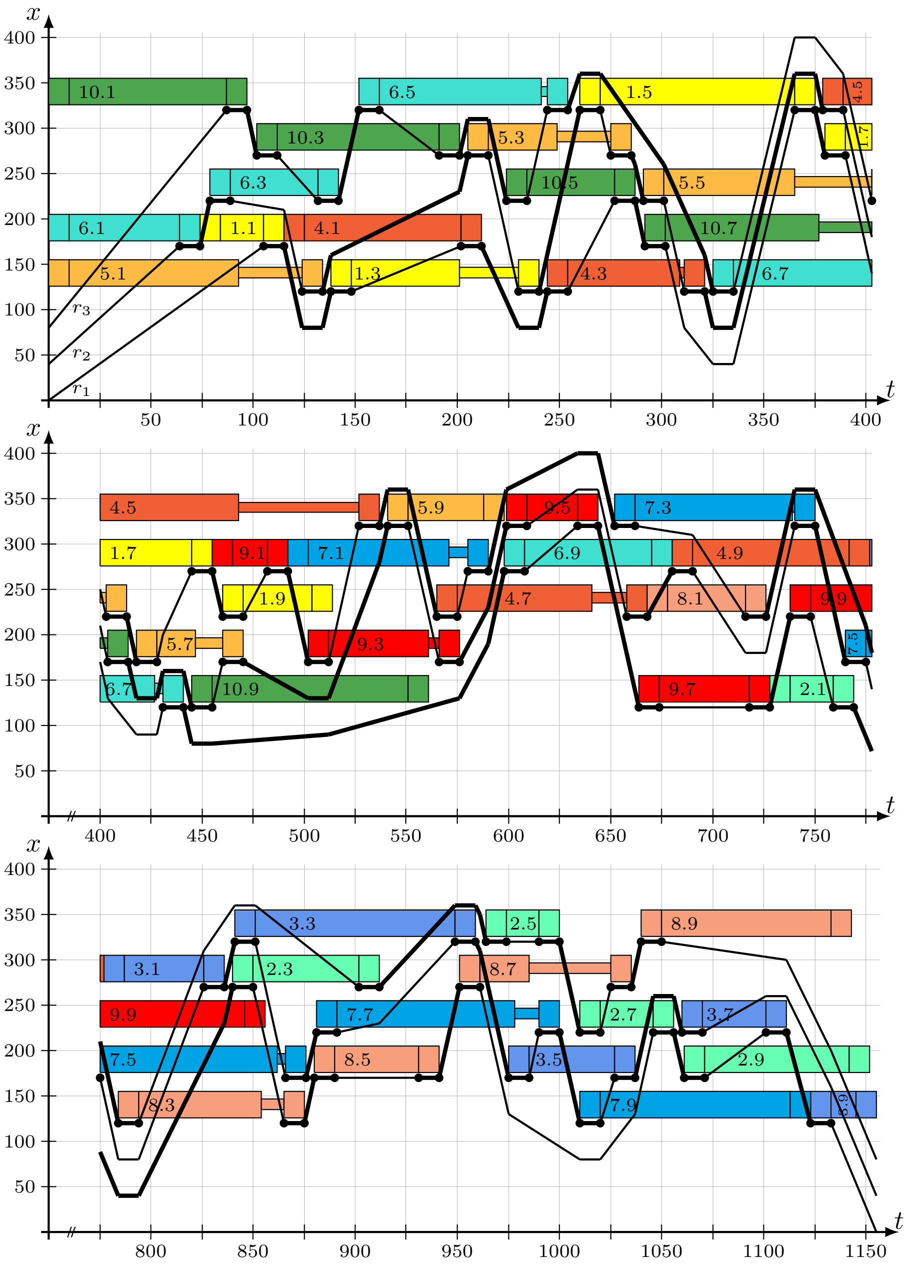


Figure 7.7.: A schedule with makespan 1155 of instance la01 with 3 robots.

Cook et al. [26], pp. 92 and 115). However, optimal trajectories can be determined more efficiently with the following algorithm based on geometric arguments.

Assume that the FTP at (π, α) has a feasible solution, and consider $\mathcal{Q} = \{t_1, \ldots, t_Q\}$ (see Section 7.4.1). At each t_p, $p = 1, \ldots, Q$, a given robot k is either required to be at a fixed location, say a_p^k, or there is no such requirement. For $p = 1, \ldots, Q$, let $\{a_p^{k(1)}, \ldots, a_p^{k(q_p)}\}$ be the set of fixed locations at t_p over all robots (it is non-empty as at least one robot is at a fixed location at t_p), and for $k = 1, \ldots, K$, let $\{a_{p(1)}^k, \ldots, a_{p(q^k)}^k\}$ be the set of all fixed locations of k over all times.

Given k and t_p, $p \in \{1, \ldots, Q\}$, the following *lower and upper bounds* l_p^k and u_p^k hold for the location $x(k, t_p)$ at t_p:

$$l_p^k = \max\{(k-1)\delta; a_p^{k(q)} + (k - k(q))\delta : k(q) \le k,\ 1 \le q \le q_p\}$$
$$u_p^k = \min\{L - (K-k)\delta; a_p^{k(q)} - (k(q) - k)\delta : k(q) \ge k,\ 1 \le q \le q_p\}$$

Note that if k is at a fixed location a_p^k, $l_p^k = a_p^k = u_p^k$.

It is helpful to consider trajectories in the two-dimensional time-location space with horizontal axis t and vertical axis x. In this space, a point will be denoted by $P = (t(P), x(P))$, $t(P)$, $x(P)$ denoting the t- and x-coordinate of P. For any two points P, P' where $t(P) < t(P')$, $[P, P']$ denotes the (line) segment joining P to P'. Its slope $(x(P') - x(P))/(t(P') - t(P))$ is denoted $\theta[P, P']$. A point P^* is *above* (*below*) the segment $[P, P']$ if there is $P'' \in [P, P']$ with $t(P'') = t(P^*)$ and $x(P'') < x(P^*)$, ($x(P^*) < x(P'')$). For any k, let F_q^k, $q = 1, \ldots, q^k$, be the *fixed points*, and L_p^k and U_p^k, $p = 1, \ldots, Q$, the *lower* and *upper points* for the trajectory of k, i.e. $t(F_q^k) = t_{p(q)}$, $x(F_q^k) = a_{p(q)}^k$, $t(L_p^k) = t(U_p^k) = t_p$, $x(L_p^k) = l_p^k$ and $x(U_p^k) = u_p^k$. The following algorithm constructs for each robot k a piecewise linear trajectory $\mathcal{T}^k$.

```
**Trajectory algorithm**
 **for** $k = 1, \ldots, K$ **do** *Trajectory*$(k, \mathcal{T}^k)$ **end**

**Subroutine** *Trajectory*$(k, \mathcal{T}^k)$
 $\mathcal{T} := \cup\{[F_q^k, F_{q+1}^k], 1 \le q < q^k\}$. No segment of $\mathcal{T}$ is scanned.
 **while** not all segments of $\mathcal{T}$ are scanned **do**
      Choose an unscanned $[P, P'] \in \mathcal{T}$, and scan $[P, P']$ as follows:
      **if** there exists some $L_p^k$ above $[P, P']$ **then**
           Determine $\theta^* = \max\{\theta[P, L_p^k] : L_p^k$ above $[P, P']\}$ and
           $L_{p^*}^k$ such that $p^*$ is the largest $p$ with $\theta[P, L_p^k] = \theta^*$.
           $\mathcal{T} := \mathcal{T} \cup \{[P, L_{p^*}^k] \cup [L_{p^*}^k, P']\} - [P, P']$.
      **else if** there exists some $U_p^k$ below $[P, P']$ **then**
           Determine $\theta^* = \min\{\theta[P, U_p^k] : U_p^k$ below $[P, P']\}$ and
           $U_{p^*}^k$ such that $p^*$ is the largest $p$ with $\theta[P, U_p^k] = \theta^*$.
           $\mathcal{T} := \mathcal{T} \cup \{[P, U_{p^*}^k] \cup [U_{p^*}^k, P']\} - [P, P']$.
```

```
        end if
    end while
$\mathcal{T}^k := \mathcal{T}$.
```

We illustrate some steps of the algorithm in the solution of the example that is depicted in Figure 7.2 (on p. 110). We assume here that the rail length is 5. Consider the two consecutive fixed points $P = (6, 2)$ and $P' = (13, 3)$ of crane 2. The segment $[P, P']$ and the lower and upper points of crane 2 that are between P and P' are depicted in Figure 7.8 (a) by a thick line, symbols $\wedge$, and $\vee$, respectively. The scan of segment $[P, P']$, illustrated in Figure 7.8 (b) and (c), is now described. There exists lower points that are above the line of segment $[P, P']$. Indeed, L^1, L^2, L^3 and L^4 are above the black line, see (b). Then, we determine for each of these points the slope of the line going through the point and point P, see the red lines in (b), and take among the points with the largest slope, the one latest in time. Here, this is point L^3. Then, we adjust the trajectory by replacing segment $[P, P']$ by the two segments $[P, L^3]$ and $[L^3, P']$, see (c). Note that no adjustments will be made when scanning segment $[P, L^3]$ while segment $[L^3, P']$ will be adjusted as it has a lower point above the line of the segment.

We now show that trajectories $\widetilde{\mathcal{T}}^k, k = 1, \ldots, K$, are feasible and discuss the time complexity of the algorithm.

Theorem 28 *The trajectories $\mathcal{T}^k$, $k = 1, \ldots, K$, are feasible and each $\mathcal{T}^k$ is a minimum travel distance trajectory for k.*

Proof. For any consecutive segments $[P, P'], [P', P''] \in \mathcal{T}^k$, call $\mathcal{T}^k$ *concave at* P' if P' is above $[P, P'']$ and *convex at* P' if P' is below $[P, P'']$. We first show that at any points that are not fixed points of k, $\mathcal{T}^k$ is concave at a lower point and convex at an upper point.

Indeed, suppose L' is a lower point of $\mathcal{T}^k$ (and not a fixed point). L' became part of the trajectory $\mathcal{T}$ in the subroutine *Trajectory*$(k, \mathcal{T}^k)$ as a lower point $L' = L^k_{p*}$ above a segment $[P, P']$ previously in $\mathcal{T}$, with the property that any L^k_p with $t(P) \le t(L^k_p) < t(L^k_{p*})$ is not above $[P, L^k_{p*}]$, and any L^k_p with $t(L^k_{p*}) < t(L^k_p) \le t(P')$ is below the line containing $[P, L^k_{p*}]$. As a result, from then on and until completion of the subroutine, for any t with $t(P) \le t < t(L')$, $\mathcal{T}$ is not above this line, $\mathcal{T}$ is on the line at $t(L')$, and at any t with $t(L') < t \le t(P')$, $\mathcal{T}$ is below the line. Hence $\mathcal{T}^k$ is concave at L'. Similarly, one shows that $\mathcal{T}^k$ is convex at any upper point that is not fixed.

Examining the constraints (7.3) to (7.6), we show now the feasibility of the trajectories $\mathcal{T}^k$, $k = 1, \ldots, K$.

Suppose (7.3) does not hold. Then, letting $[P, P'] \in \mathcal{T}^k$ be a steepest segment of $\mathcal{T}^k$ (with maximum $|\theta[P, P']|$),

$$|\theta[P, P']| = |x(P') - x(P)|/(t(P') - t(P)) > v_k. \tag{7.29}$$

Assume $\theta[P, P'] > 0$. P cannot be lower and not fixed since $\mathcal{T}^k$ is concave at such a point and P' cannot be upper and not fixed since $\mathcal{T}^k$ is convex at such a point. Hence

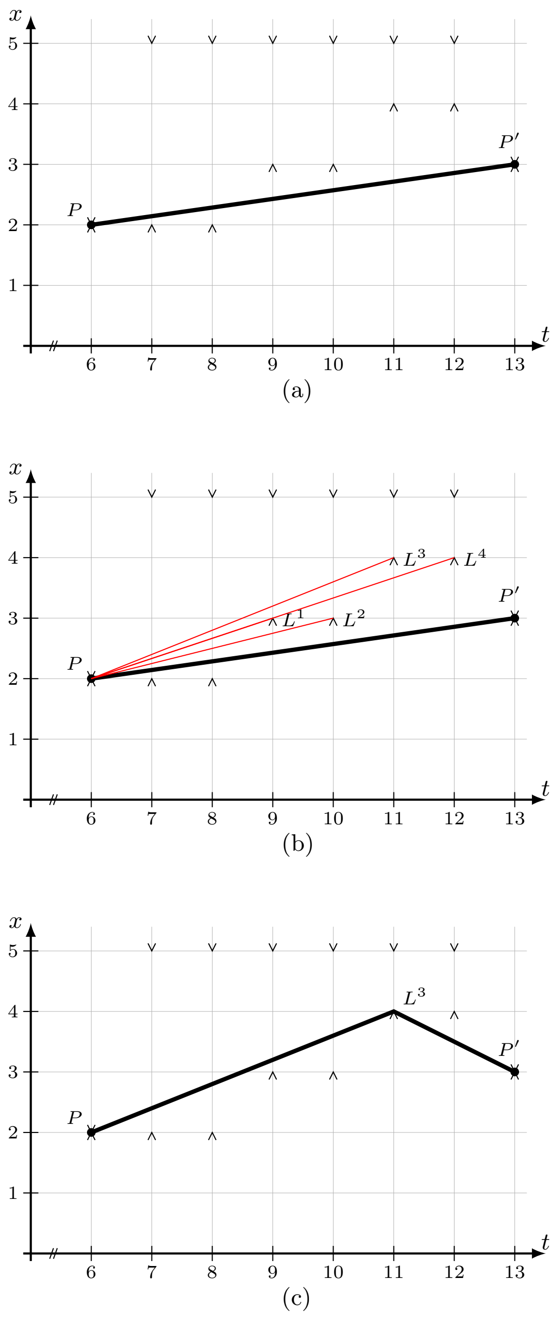


Figure 7.8.: A step in the trajectory algorithm.

P is upper or fixed and P' is lower or fixed, and $P = (t_p, u_p^k)$ and $P' = (t_{p'}, l_{p'}^k)$ for some t_p , $t_{p'}$, $1 \leq p < p' \leq Q$. By (7.29), $l_{p'}^k - u_p^k > (t_{p'} - t_p)v_k$, contradicting the feasibility of the FTP, since $l_{p'}^k \leq x(k, t_{p'})$, $x(k, t_p) \leq u_p^k$ and $x(k, t_{p'}) - x(k, t_p) \leq (t_{p'} - t_p)v_k$ hold for any feasible trajectory for k. The case $\theta[P, P'] < 0$ is similar.

Constraints (7.4) hold since for any transfer step of k with starting and completion time, say, $t_{p'}$ and $t_{p''}$, at any t_p with $p' \leq p \leq p''$, the trajectory of k is fixed (at the location of the transfer step). Constraints (7.5) also hold. Indeed, for any k, $1 \leq k < K$, let trajectories $\mathcal{T}^k$ and $\mathcal{T}^{k+1}$ be at some t at a minimal (x-) distance from each other. By the concavity and convexity properties described above, we may assume that at time t, $\mathcal{T}^k$ is at a lower or fixed point, or $\mathcal{T}^{k+1}$ at an upper or fixed point. In the first case, for p such that $t_p = t$, $\mathcal{T}^k$ is at point $L_p^k = (t_p, l_p^k)$. By definition of the lower bounds and feasibility of the FTP, $l_p^{k+1} - l_p^k \geq \delta$, and by construction of $\mathcal{T}^{k+1}$, l_p^{k+1} is below or on $\mathcal{T}^{k+1}$, hence $\mathcal{T}^k$ and $\mathcal{T}^{k+1}$ are at least at an (x-) distance δ from each other. The second case is similar. Finally, (7.6) obviously holds.

To show that each $\mathcal{T}^k$ is a trajectory of minimum travel distance, it is enough to observe that at any step of the subroutine *Trajectory*$(k, \mathcal{T}^k)$, the total distance traveled by $\mathcal{T}$ is a lower bound on the total travel distance of any feasible trajectory for k. ∎

Proposition 29 *The trajectory algorithm runs in time $\mathcal{O}(K|I^R|^2)$.*

Proof. Lower and upper bounds l_p^k and u_p^k can be computed in $\mathcal{O}(KQ)$ if one takes into account that for any robot k and event p the closest fixed robot at p below (above) k determines the lower bound l_p^k (upper bound u_p^k).

Now consider the runtime of the trajectory subroutine by estimating the number of scanned segments and the effort spent for one scan.

As each segment is scanned exactly once, we estimate the number of segments considered in a run. At the start, the number of segments is bounded by $Q - 1$. In any scan of a segment, two new segments are added if some point L_{p*}^k or U_{p*}^k is found. However, each point P of robot k can be selected at most in one scan as L_{p*}^k or U_{p*}^k. Therefore, at most $2Q$ new segments can be added to trajectory $\mathcal{T}$, hence there are at most $3Q$ segments considered in a run. The time spend for scanning a segment $[P, P']$ is bounded by the number of events in $[P, P']$, hence the effort of a scan is $\mathcal{O}(Q)$ and the effort of the subroutine is then $\mathcal{O}(Q^2)$.

The subroutine is executed for all robots $k = 1, \ldots, K$, hence the overall effort is $\mathcal{O}(KQ^2)$, or $\mathcal{O}(K|I^R|^2)$, since $Q \leq 4|I^R|$. Note that this effort is smaller than that of any generic min-cost flow algorithm since already the size of the network (number of nodes) is $KQ + 1$. ∎

We also observe that in the “classical” case where a job is assumed to have at most one machining operation on a given machine, this complexity can be related to the number m of machines and the number n of jobs. Then $|I^R| \leq (m - 1)n$, and the trajectory algorithm runs in time $\mathcal{O}(Km^2n^2)$.

7.7.2. Stop-and-Go Trajectories

The trajectories $\mathcal{T}^k$, $k = 1, \ldots, K$, make use of the variable speed (up to its maximum v_k) of each robot k. If all v_k's are equal, say v, it is possible to adapt the trajectories $\mathcal{T}^k$ to trajectories $\widetilde{\mathcal{T}}^k$, $k = 1, \ldots, K$, with the following properties. A robot is either still ("stop") or moves at speed v ("go") and switches between stop and go *at most once* in any time interval $[t_p, t_{p+1}], 1 \leq p < Q$, so that overall, the number of these switches is less than Q. Moreover, the travel distance of each crane remains the same as in $\mathcal{T}^k$, and is therefore minimal.

Let a point $P \in \mathcal{T}^k$ be called a *generating point* of $\mathcal{T}^k$ if P is an endpoint of some segment $[P, P'] \subseteq \mathcal{T}^k$. We will need the following property of the trajectories $\mathcal{T}^k, k = 1, \ldots, K$.

Proposition 30 *Let $\mathcal{T}^k$ and $\mathcal{T}^{k+1}$ be two adjacent trajectories. i) If $(t, x') \in \mathcal{T}^{k+1}$ is a generating point of $\mathcal{T}^{k+1}$ and $(t, x) \in \mathcal{T}^k$ is not a generating point of $\mathcal{T}^k$ then (t, x') is upper or fixed. ii) If $(t, x) \in \mathcal{T}^k$ is a generating point of $\mathcal{T}^k$ and $(t, x') \in \mathcal{T}^{k+1}$ is not a generating point of $\mathcal{T}^{k+1}$ then (t, x) is lower or fixed.*

Proof. Let $(t, x') \in \mathcal{T}^{k+1}$ be a generating point of $\mathcal{T}^{k+1}$. Then $t = t_p$ for some p and $x' = u_p^{k+1}$ or $x' = l_p^{k+1}$ or both. If (t, x') is a lower point of $\mathcal{T}^{k+1}$, $x' = l_p^{k+1}$ and $\mathcal{T}^{k+1}$ is concave at (t, x'). As (t, x') is lower, it is not fixed, so $l_p^{k+1} = l_p^k + \delta$. If additionally, $(t, x) \in \mathcal{T}^k$ is not a generating point of $\mathcal{T}^k$, and there is $[P, P'] \subseteq \mathcal{T}^k$ and $(t_p, x) \in [P, P']$ such that $t(P) < t_p < t(P')$. Since the trajectories are feasible, $x \geq l_p^k$ and $x' - x \geq \delta$. Hence $x \geq l_p^{k+1} - \delta = x' - \delta$ so that $x' - x = \delta$ must hold. But then, by the concavity of $\mathcal{T}^{k+1}$ at (t, x'), there is a (left or right) neighbor point of (t, x') in $\mathcal{T}^{k+1}$ at a distance smaller than δ from $[P, P']$, contradicting the feasibility of the trajectories. ii) is shown similarly. ■

Trajectories $\widetilde{\mathcal{T}}^k, k = 1, \ldots, K$, can be constructed with the following algorithm.

```
Stop-and-go trajectory algorithm
 for each trajectory $\mathcal{T}^k$
      $\widetilde{\mathcal{T}}^k := \emptyset$
      for each segment $[P, P'] \subseteq \mathcal{T}^k$
           if $\theta[P, P'] = 0$ or $|\theta[P, P']| = v$ then $\widetilde{\mathcal{T}}^k := \widetilde{\mathcal{T}}^k \cup [P, P']$
           else if $\theta[P, P'] > 0$ then $Up([P, P'], \widetilde{\mathcal{T}}^k)$ else $Down([P, P'], \widetilde{\mathcal{T}}^k)$
      end for
 end for
```

```
Subroutine $Up([P, P'], \widetilde{\mathcal{T}}^k)$
 $P^3 := P'$
 while $P^3 \neq P$ do
      Determine the line $L$ through $P^3$ with slope $v$.
      Let point set $\mathcal{P} = \{(t_p, l_p^k) : t(P) \leq t_p < t(P^3)$ and $t_p < t_{p'}$ for $(t_{p'}, l_p^k) \in L\}$.
```

```
        Determine $l^* = \max\{x(P'') : P'' \in \mathcal{P}\}$ and let $P^1 \in \mathcal{P}$ be the first point in
            time among all points in $\mathcal{P}$ at location $l^*$.
        Determine the intersection point $P^2$ of $L$ with the horizontal line through $P^1$.
        $\widetilde{\mathcal{T}}^k := \widetilde{\mathcal{T}}^k \cup [P^1, P^2] \cup [P^2, P^3]$
        $P^3 := P^1$
  end while
```

```
Subroutine Down([P, P'], $\widetilde{\mathcal{T}}^k$)
  $P^3 := P'$
  while $P^3 \neq P$ do
        Determine the line $L$ through $P^3$ with slope $-v$.
        Let point set $\mathcal{P} = \{(t_p, u_p^k) : t(P) \leq t_p < t(P^3)$ and $t_p < t_{p'}$ for $(t_{p'}, u_p^k) \in L\}$.
        Determine $u^* = \min\{x(P'') : P'' \in \mathcal{P}\}$ and let $P^1 \in \mathcal{P}$ be the first point in
            time among all points in $\mathcal{P}$ at location $u^*$.
        Determine the intersection point $P^2$ of $L$ with the horizontal line through $P^1$.
        $\widetilde{\mathcal{T}}^k := \widetilde{\mathcal{T}}^k \cup [P^1, P^2] \cup [P^2, P^3]$
        $P^3 := P^1$
  end while
```

We illustrate some steps of the algorithm in the solution of the example that is depicted in Figure 7.2 (on p. 110). Consider the segment $[P, P']$ with $P = (6, 2), P' = (11, 4)$ of the trajectory of crane 2. This segment and the lower bounds l_p^2 with $6 \leq t_p \leq 11$ are depicted in Figure 7.9 (a) by a thick line and symbols $\wedge$, respectively. As the crane moves up in this segment with a speed lower than its maximum speed, we have to adjust the segment in subroutine *Up* as follows. The adjustments are illustrated in Figure 7.9 (b)-(d). Introduce point P^3 and let $[P, P^3]$ be the subsegment of $[P, P']$ we currently consider. Set $P^3 := P'$ in the beginning. Then, we draw line L through P^3 with slope v (red line in (b)). Then, consider all points (t_p, l_p^2) that are in time between P and P^3 and strictly left (earlier in time) of the point on line L having the same location l_p^2, and call this set of points $\mathcal{P}$. Here, $\mathcal{P} = \{(6, 2), (7, 2), (8, 2), (9, 3)\}$. Note that point $(10, 3) \notin \mathcal{P}$ as this point lies on line L. The highest location among the points in $\mathcal{P}$ is 3, hence $l^* = 3$, and $P^1 = (9, 3)$ is the first point (in time) among all points in $\mathcal{P}$ at this location. Draw a horizontal line through P^1 (blue line in (b)) and let P^2 be the intersection of this line with L. The two segments $[P^1, P^2]$ and $[P^2, P^3]$ are now part of the trajectory $\widetilde{\mathcal{T}}^k$, $[P^1, P^2]$ being a “stop” segment and $[P^2, P^3]$ a “go” segment. Point P^3 is reset to P^1 and the above steps are re-executed, see (c), until $P = P^3$, see (d).

We now show that trajectories $\widetilde{\mathcal{T}}^k, k = 1, \ldots, K$, are feasible.

Proposition 31 *For any $[P, P'] \subseteq \mathcal{T}^k$ and t_p with $t(P) \leq t_p \leq t(P')$, $l_p^k \leq x \leq u_p^k$ holds for $(t_p, x) \in \widetilde{\mathcal{T}}^k$.*

Proof. Follows from the construction of $\widetilde{\mathcal{T}}^k$. ■

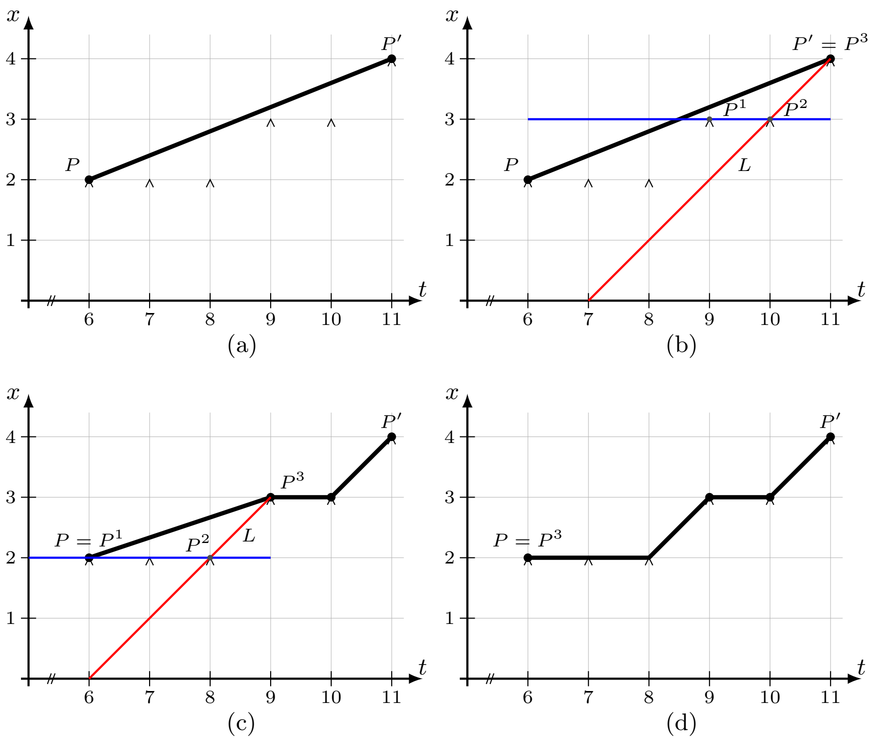


Figure 7.9.: A step in the stop-and-go algorithm.

Theorem 32 *Trajectories $\widetilde{\mathcal{T}}^k, k = 1, \ldots, K,$ are feasible.*

Proof. Denote by δ_t the distance at time t of $\mathcal{T}^{k+1}$ from $\mathcal{T}^k$, i.e. $\delta_t = x' - x$ where $(t, x) \in \mathcal{T}^k$ and $(t, x') \in \mathcal{T}^{k+1}$, and by $\widetilde{\delta}_t$ the distance at time t of $\widetilde{\mathcal{T}}^{k+1}$ from $\widetilde{\mathcal{T}}^k$. To establish the feasibility of $\widetilde{\mathcal{T}}^k, k = 1, \ldots, K$ it is sufficient to prove $\widetilde{\delta}_t \geq \delta$ for $t_1 \leq t \leq t_Q$ (the other conditions being obviously fulfilled).

i) First, we prove $\widetilde{\delta}_t \geq \delta$ at a time t where a) $(t, x') \in \widetilde{\mathcal{T}}^{k+1}$ is a generating point of $\mathcal{T}^{k+1}$ or b) $(t, x) \in \widetilde{\mathcal{T}}^k$ is a generating point of $\mathcal{T}^k$. If a) and b) hold, $\widetilde{\delta}_t = \delta_t \geq \delta$. If a) and not b) hold, $t = t_p$ for some p and $x' = u_p^{k+1}$ by Proposition 30. Also, $u_p^k = u_p^{k+1} - \delta$ and by Proposition 31, $x \leq u_p^k$, hence $\widetilde{\delta}_t = x' - x \geq \delta$. If b) and not a) hold, $\widetilde{\delta}_t \geq \delta$ is shown similarly.

ii) Now given any t, there are $[P^1, P^2] \subseteq \mathcal{T}^k$ and $[P^3, P^4] \subseteq \mathcal{T}^{k+1}$ such that $t' \leq t \leq t''$ where $t' = \max\{t(P^1), t(P^3)\}$ and $t'' = \min\{t(P^2), t(P^4)\}$. We show that $\widetilde{\delta}_{t't''} = \min\{\widetilde{\delta}_s : t' \leq s \leq t''\} \geq \delta$.

If $\theta[P^1, P^2] \leq 0$ and $\theta[P^3, P^4] \geq 0$, $\widetilde{\delta}_{t't''} = \widetilde{\delta}_{t'}$. Since at time t', $P \in \widetilde{\mathcal{T}}^k$ is a generating point of $\mathcal{T}^k$ or $P'' \in \widetilde{\mathcal{T}}^{k+1}$ is a generating point of $\mathcal{T}^{k+1}$, $\widetilde{\delta}_{t'} \geq \delta$ by i), hence $\widetilde{\delta}_{t't''} \geq \delta$. The case $\theta[P^1, P^2] > 0$ and $\theta[P^3, P^4] \leq 0$ is similar.

If $\theta[P^1, P^2] \geq 0$ and $\theta[P^3, P^4] \geq 0$, either $\widetilde{\delta}_{t't''}$ is attained at time t' or at t'', in which case $\widetilde{\delta}_{t't''} \geq \delta$ (see above), or it is attained at some $s, t' < s < t''$, where $\widetilde{P} = (s, x) \in \widetilde{\mathcal{T}}^k$ and $[\widetilde{P}, \widetilde{P}'] \subseteq \widetilde{\mathcal{T}}^k$ is a horizontal ("stop") segment. Then, $s = t_p$ and $x = l_p^k$ for some p. Moreover, $l_p^{k+1} = l_p^k + \delta$ and by Proposition 31, $x' \geq l_p^{k+1}$ for $(s, x') \in \widetilde{\mathcal{T}}^{k+1}$, hence $\widetilde{\delta}_{t't''} = x' - x \geq \delta$. The case $\theta[P^1, P^2] \leq 0$ and $\theta[P^3, P^4] \leq 0$ is similar. ■

Remark 33 *The trajectories $\widetilde{\mathcal{T}}^k, k = 1, \ldots, K,$ are latest move-time trajectories. It is easy to devise a version of the stop-and-go trajectory algorithm where each robot moves as early as possible.*

The discussion on finding feasible trajectories is concluded with two figures showing stop-and-go trajectories. Figures 7.10 and 7.11 depict the same schedules as Figures 7.2 and 7.7, but now with stop-and-go trajectories.

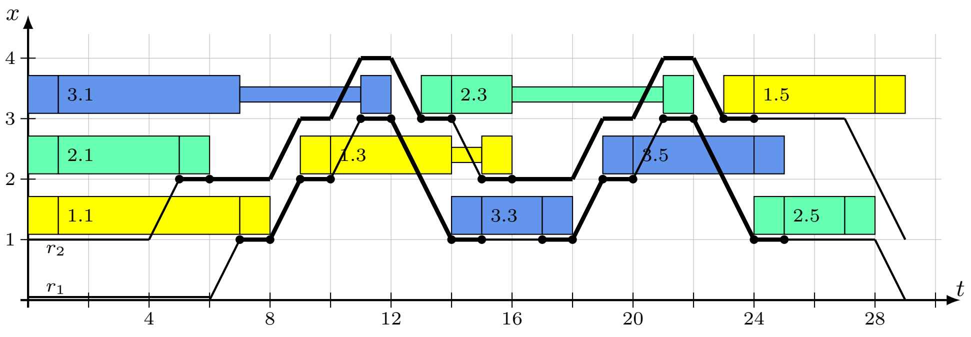


Figure 7.10.: A solution of the BJS-RT example with stop-and-go trajectories.

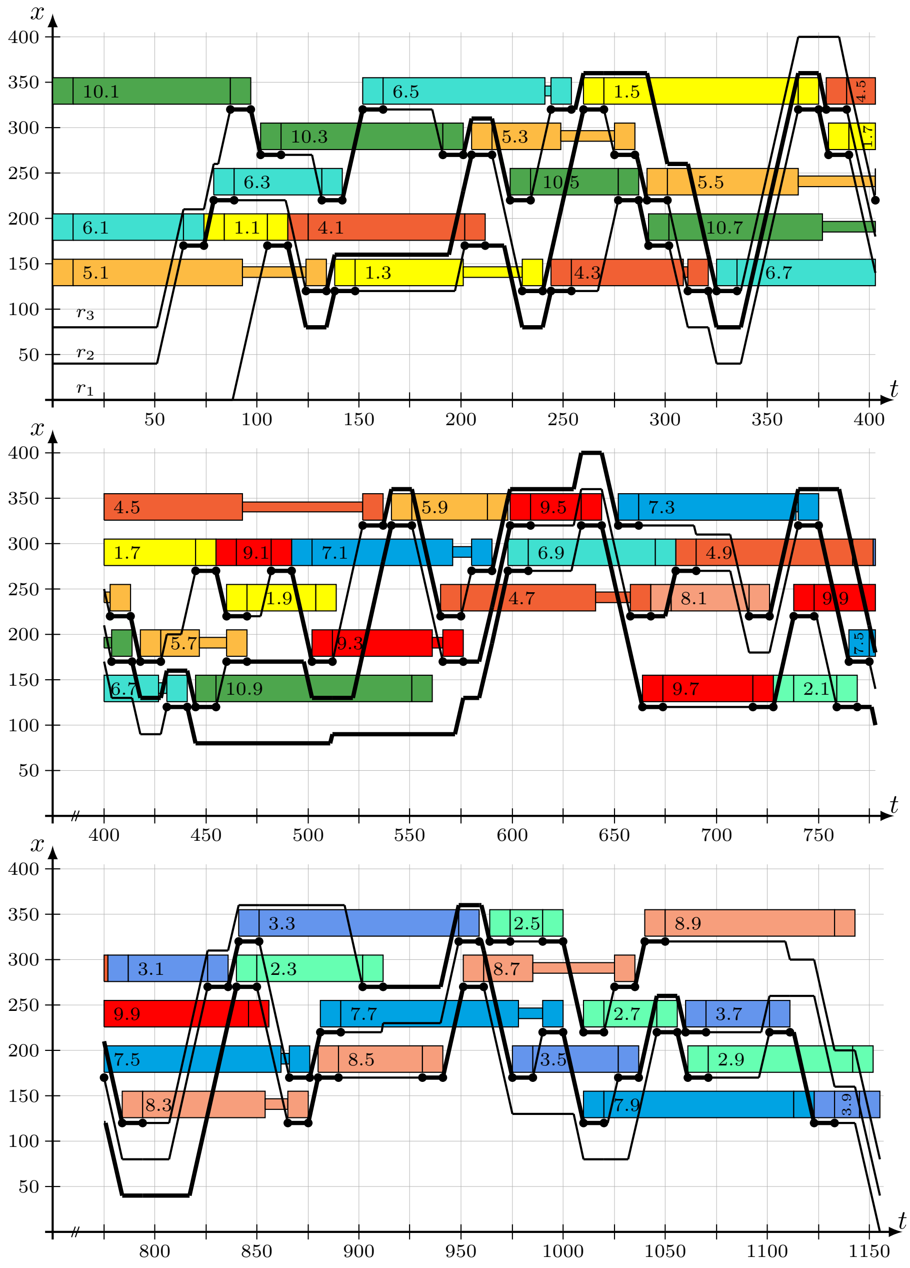


Figure 7.11.: A schedule with makespan 1155 of instance la01 with 3 robots and stop-and-go trajectories.

CHAPTER 8 CONCLUSION

Starting in 1954 with the seminal article on the flow shop problem by Johnson [55], scheduling has become a major field in Operations Research over the last fifty years, and has attracted a large number of researchers.

Potts and Strusevich mentioned in 2009 in their article "Fifty years of scheduling: a survey of milestones" [98]: "Scheduling research carried out during the current, fifth decade is diverse in nature. Thus, it is difficult to detect strong themes that have attracted most attention. Even though many researchers were motivated by the need to create scheduling models that capture more of the features that arise in practice, the enhancements to classical scheduling models cannot be embedded into a unified framework." And they conclude: "At the end of this decade, scheduling had become much more fragmented. A large number of researchers were working in the area but seemingly on a huge variety of problems and adopting a multitude of different approaches. In spite of the vast body of research being produced, a large gap remains between theory and practice."

Following the philosophy of former work, e.g. by Gröflin and Klinkert [42, 43], Klinkert [62] and Pham [92], this thesis aimed at contributing to narrow the gap between theory and practice in scheduling problems of the job shop type. We first summarize what we believe to be the main contributions and then point to future research appearing of interest in our eyes.

We proposed a general model called Complex Job Shop (CJS) capturing a broad variety of practical features, and developed a comprehensive formulation in a general disjunctive graph, which is of a more complex structure than disjunctive graphs for the classical job shop.

We then presented a general solution method called Job Insertion Based Local Search (JIBLS), which can be used for most CJS problems. A key component of the JIBLS is the generation of feasible neighbors, where typically a critical operation is moved together with some other operations whose moves are "implied". The moves are defined and generated in the framework of job insertion with local flexibility.

We evaluated the CJS model and the JIBLS solution method by tailoring and applying them to a selection of complex job shop problems. Some of these problems are well-known in the literature, while the others are new. In the well-known problems, i.e. the Flexible Job Shop with Setup Times, the Job Shop with Transportation and the Blocking Job Shop, the JIBLS provides good results when compared to the state of the art. In the new problems, i.e. the Flexible Blocking Job Shop with Transfer and Setup Times, the Blocking Job Shop with Transportation and the Blocking Job Shop with Rail-Bound Transportation (BJS-RT), the JIBLS establishes first benchmarks. Altogether, the results obtained support the validity of our approach.

The BJS-RT deserves special mention. In this problem, not only machine (robot) assignments and starting times must be specified as in a usual scheduling problem, but also feasible trajectories of the robots must be determined. A projection of the solution space of the BJS-RT onto the space of the assignment and starting time variables yields a disjunctive graph formulation of the BJS-RT as a CJS. In addition, efficient algorithms solve the feasible trajectory problem.

As future work directions, we see further applications of the CJS model and the JIBLS solution method, extensions of the model, and extensions of the solution method.

First, the complex job shop landscape given in Figure 1.12 suggests applying and tailoring the CJS model and the JIBLS to the Flexible No-Wait Job Shop with Setup Times (FNWJSS). The No-Wait Job Shop with Transportation (NWJS-T), which can be seen as a special FNWJSS, is also of interest as no-wait constraints arise in various settings with transportation, e.g. in electroplating plants [66, 89] and in robotic cells [2, 30].

Second, from an application perspective, addressing more general structures of a job would be valuable. For example, the assembly job shop, where a job is a (converging or diverging) arborescence of operations, is common in practice, typically occurring when components are assembled into a final product. It is also worthwhile to further study job shop scheduling problems with transportation where the robots interfere in their movements. The approach taken for the derivation of the disjunctive graph formulation in the BJS-RT can be used in other cases by changing the relaxed scheduling problem, for example substituting the BJS-T with the JS-T if sufficient buffer space is available, or changing the characteristics of the transportation system (e.g. a carousel instead of a rail line), and accordingly, the corresponding feasible trajectory problem. While the first change is straightforward, the second raises interesting opportunities from both a research and application perspective.

Third, additional work is needed in CJS problems with general time lags. While problems with specific time lags possessing the Short Cycle Property (SCP) may be found and solved with the JIBLS, a method applicable to problems with general time lags would be valuable. Finally, in the JIBLS, neighbors are generated by local "changes". Considering a larger neighborhood, where a job is extracted and re-inserted in an optimal fashion would be interesting. This mechanism, known as optimal job insertion, is not only valuable in local search but also in the construction of an initial schedule or when scheduling is done in a rolling fashion, inserting a job at its arrival into the current schedule. As the optimal job insertion problem is non-trivial in many complex job shop problems, it would be interesting to find methods that insert a job in a "near-optimal" way.

BIBLIOGRAPHY

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

[2] A. Agnetis. Scheduling no-wait robotic cells with two and three machines. *European Journal of Operational Research*, 123(2):303–314, June 2000.

[3] A. Allahverdi, C. Ng, T. Cheng, and M. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, June 2008.

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

[5] I. Aron, L. Genç-Kaya, I. Harjunkoski, S. Hoda, and J. Hooker. Factory crane scheduling by dynamic programming. In R. K. Wood and R. F. Dell, editors, *Operations Research, Computing and Homeland Defense (ICS 2011 Proceedings)*, pages 93–107. INFORMS, 2010.

[6] C. Artigues and D. Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, 159(1):135–159, 2008.

[7] K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling.* Wiley, 2009.

[8] E. Balas. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research*, 17(6), 1969.

[9] E. Balas, N. Simonetti, and A. Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11(4):253–262, 2008.

[10] U. Bilge and G. Ulusoy. A time window approach to simultaneous scheduling of machines and material handling system in an FMS. *Operations Research*, 43(6):1058–1070, 1995.

[11] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications.* Springer, 2007.

[12] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993.

[13] C. A. Brizuela, Y. Zhao, and N. Sannomiya. No-wait and blocking job-shops: challenging problems for GA's. In *IEEE international conference on systems, man, and cybernetics*, pages 2349–2354, 2001.

[14] P. Brucker, E. K. Burke, and S. Groenemeyer. A branch and bound algorithm for the cyclic job-shop problem with transportation. *Computers & Operations Research*, 39(12):3200–3214, Dec. 2012.

[15] P. Brucker, S. Heitmann, J. Hurink, and T. Nieberg. Job-shop scheduling with limited capacity buffers. *OR Spectrum*, 28(2):151–176, 2006.

[16] P. Brucker and T. Kampmeyer. Cyclic job shop scheduling problems with blocking. *Annals of Operations Research*, 159(1):161–181, 2008.

[17] P. Brucker and S. Knust. *Complex Scheduling.* Springer, 2nd edition, 2011.

[18] P. Brucker and C. Strotmann. Local search procedures for job-shop problems with identical transport robots. In *Eight International Workshop on Project Management and Scheduling*, Valencia (Spain), 2002.

[19] P. Brucker and O. Thiele. A branch & bound method for the general-shop problem with sequence dependent setup-times. *OR Spectrum*, 18(3):145–161, 1996.

[20] R. Bürgy and H. Gröflin. Optimal job insertion in the no-wait job shop. *Journal of Combinatorial Optimization*, 26(2):345–371, 2013.

[21] R. Bürgy and H. Gröflin. The blocking job shop with rail-bound transportation. *Journal of Combinatorial Optimization*, 2014. Published Online First. March 7, 2014. doi: 10.1007/s10878-014-9723-3.

[22] J. R. Callahan. *The Nothing Hot Delay Problems in the Production of Steel.* PhD thesis, University of Toronto, Canada, 1971.

[23] J. Chambers and J. Barnes. Reactive search for flexible job shop scheduling. Technical report, University of Texas at Austin, Austin, 1997.

[24] R. B. Chase, F. R. Jacobs, and N. J. Aquilano. *Operations & Supply Management.* McGraw-Hill, New York, 12th edition, 2008.

[25] A. Che and C. Chu. Cyclic hoist scheduling in large real-life electroplating lines. *OR Spectrum*, 29(3):445–470, May 2006.

[26] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* Wiley-Interscience, 1997.

[27] Y. Crama and V. Kats. Cyclic scheduling in robotic flowshops. *Annals of Operations Research*, 96(1-4):97–124, 2000.

[28] Y. Crama and J. V. D. Klundert. Cyclic scheduling of identical parts in a robotic cell. *Operations Research*, 45(6):952–965, 1997.

[29] S. Dauzère-Pérès and J. Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306, 1997.

[30] M. Dawande, H. N. Geismar, S. P. Sethi, and C. Sriskandarajah. Sequencing and scheduling in robotic cells: recent developments. *Journal of Scheduling*,

8(5):387–426, Oct. 2005.

[31] L. Deroussi, M. Gourgand, and N. Tchernev. A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, 46(8):2143–2164, Apr. 2008.

[32] F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, volume scheduling, pages 92–101, 2000.

[33] G. Galante and G. Passannanti. Minimizing the cycle time in serial manufacturing systems with multiple dual-gripper robots. *International Journal of Production Research*, 44(4):639–652, Feb. 2006.

[34] J. Gao, L. Sun, and M. Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35:2892–2907, 2008.

[35] H. Geismar, C. Sriskandarajah, and N. Ramanan. Increasing throughput for robotic cells with parallel machines and multiple robots. *IEEE Transactions on Automation Science and Engineering*, 1(1):84–89, July 2004.

[36] H. N. Geismar, M. Pinedo, and C. Sriskandarajah. Robotic cells with parallel machines and multiple dual gripper robots: a comparative overview. *IIE Transactions*, 40(12):1211–1227, Oct. 2008.

[37] F. W. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[38] F. W. Glover and M. Laguna. *Tabu search*. Kluwer, Boston, 1997.

[39] F. W. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41(1):3–28, 1993.

[40] V. Gondek. *Hybrid Flow-Shop Scheduling mit verschiedenen Restriktionen: Heuristische Lösung und LP-basierte untere Schranken*. Phd thesis, Universität Duisburg-Essen, 2011.

[41] M. A. González, C. R. Vela, and R. Varela. An efficient memetic algorithm for the flexible job shop with setup times. *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013.

[42] H. Gröflin and A. Klinkert. Feasible insertions in job shop scheduling, short cycles and stable sets. *European Journal of Operational Research*, 177(2):763–785, 2007.

[43] H. Gröflin and A. Klinkert. A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics*, 157(17):3643–3655, 2009.

[44] H. Gröflin and D. N. Pham. The flexible blocking job shop with transfer and set-up times. Technical report, University of Fribourg, Fribourg, 2008.

[45] H. Gröflin, D. N. Pham, and R. Bürgy. The flexible blocking job shop with transfer and set-up times. *Journal of Combinatorial Optimization*, 22(2):121–144, 2011.

[46] Gurobi Optimization. Gurobi Optimizer reference manual, retrieved August 13, 2013, from http://www.gurobi.com/documentation/5.5/reference-manual/.

[47] N. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, 1996.

[48] N. G. Hall, H. Kamoun, and C. Sriskandarajah. Scheduling in robotic cells: classification, two and three machine cells. *Operations Research*, 45(3):421–439, May 1997.

[49] S. Heitmann. *Job-shop scheduling with limited buffer capacities.* PhD thesis, Universität Osnabrück, 2007.

[50] A. Hertz, Y. Mottet, and Y. Rochat. On a scheduling problem in a robotized analytical system. *Discrete Applied Mathematics*, 65(13):285–318, 1996.

[51] A. Hmida, M. Haouari, M. Huguet, and P. Lopez. Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research*, 37:2192–2201, 2010.

[52] J. Hurink, B. Jurisch, and M. Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, pages 205–215, 1994.

[53] J. Hurink and S. Knust. Tabu search algorithms for job-shop problems with a single transport robot. *European Journal of Operational Research*, 162(1):99–111, 2005.

[54] T. Hürlimann. Reference manual of LPL. Retrieved August 13, 2013, from http://www.virtual-optima.com/download/docs/manual.pdf.

[55] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[56] G. E. Khayat, A. Langevin, and D. Riopel. Integrated production and material handling scheduling using mathematical programming and constraint programming. *European Journal of Operational Research*, 175(3):1818–1832, Dec. 2006.

[57] B. Khosravi, J. Bennell, and C. Potts. Train scheduling and rescheduling in the UK with a modified shifting bottleneck procedure. In D. Delling and L. Liberti, editors, *ATMOS*, pages 120–131, Dagstuhl, Germany, 2012. OASICS.

[58] K. Kim and K. Kim. An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science*, 33(1):17–33, 1999.

[59] T. Kis. *Insertion Techniques for Job Shop Scheduling.* Phd thesis, EPFL, 2001.

[60] T. Kis. Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151(2):307–332, Dec. 2003.

[61] T. Kis and A. Hertz. A lower bound for the job insertion problem. *Discrete Applied Mathematics*, 128:395–419, 2003.

[62] A. Klinkert. *Optimierung automatisierter Kompaktlager in Entwurf und Steuerung.* PhD thesis, University of Fribourg, 2001.

[63] P. Lacomme, M. Larabi, and N. Tchernev. Job-shop based framework for simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Economics*, 143(1):24–34, July 2010.

[64] S. Lawrence. Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University, Pittsburgh, PA*, 1984.

[65] J. K. Lenstra and A. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[66] J. M. Y. Leung and E. Levner. An efficient algorithm for multi-hoist cyclic scheduling with fixed processing times. *Operations Research Letters*, 34(4):465–472, July 2006.

[67] J. M. Y. Leung and G. Zhang. Optimal cyclic scheduling for printed circuit board production lines with multiple hoists and general processing sequence. *IEEE Transactions on Robotics and Automation*, 19(3):480–484, June 2003.

[68] J. M. Y. Leung, G. Zhang, X. Yang, R. Mak, and K. Lam. Optimal cyclic multi-hoist scheduling: a mixed integer programming approach. *Operations Research*, 52(6):965–976, Nov. 2004.

[69] W. Li, Y. Wu, M. Petering, M. Goh, and R. de Souza. Discrete time model and algorithms for container yard crane scheduling. *European Journal of Operational Research*, 198(1):165–172, Oct. 2009.

[70] R. W. Lieberman and I. B. Turksen. Two-operation crane scheduling problems. *IIE Transactions*, 14(3):147–155, 1982.

[71] S. Liu and E. Kozan. Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model. *Transportation Science*, 45(2):175–198, 2011.

[72] R. Logendran and A. Sonthinen. A tabu search-based approach for scheduling job-shop type flexible manufacturing systems. *Journal of the Operational Research Society*, 48(3):264–277, Dec. 1997.

[73] M.-A. Manier and C. Bloch. A classification for hoist scheduling problems. *International Journal of Flexible Manufacturing Systems*, 15(1):37–55, 2003.

[74] M.-A. Manier, C. Varnier, and P. Baptiste. Constraint-based model for the cyclic multi-hoists scheduling problem. *Production Planning & Control*, 11(3):244–257, Jan. 2000.

[75] A. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

[76] A. Mascis and D. Pacciarelli. Machine scheduling via alternative graphs. Technical report, Università degli Studi Roma Tre, Rome, 2000.

[77] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[78] S. J. Mason, J. W. Fowler, and W. Matthew Carlyle. A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling*, 5(3):247–262, May 2002.

[79] M. Mastrolilli and L. Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.

[80] C. Meloni, D. Pacciarelli, and M. Pranzo. A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131(1):215–235, 2004.

[81] A. Narasimhan and U. Palekar. Analysis and algorithms for the transtainer routing problem in container port operations. *Transportation Science*, 36(1):63–78, 2002.

[82] W. Ng. Crane scheduling in container yards with inter-crane interference. *European Journal of Operational Research*, 164(1):64–78, July 2005.

[83] W. Ng and K. Mak. Yard crane scheduling in port container terminals. *Applied Mathematical Modelling*, 29(3):263–276, Mar. 2005.

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

[85] A. Oddi, R. Rasconi, A. Cesta, and S. Smith. Applying iterative flattening search to the job shop scheduling problem with alternative resources and sequence dependent setup times. *ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, 2011.

[86] A. Oddi, R. Rasconi, A. Cesta, and S. F. Smith. Solving job shop scheduling with setup times through constraint-based iterative sampling: an experimental analysis. *Annals of Mathematics and Artificial Intelligence*, 62(3-4):371–402, Aug. 2011.

[87] A. Oddi, R. Rasconi, A. Cesta, and S. F. Smith. Iterative improvement algorithms for the blocking job shop. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[88] D. Pacciarelli and M. Pranzo. A tabu search algorithm for the railway scheduling problem. In *4th Metaheuristics International Conference*, pages 159–164, 2001.

[89] H. J. Paul, C. Bierwirth, and H. Kopfer. A heuristic scheduling procedure for multi-item hoist production lines. *International Journal of Production Economics*, 105(1):54–69, Jan. 2007.

[90] B. Peterson, I. Harjunkoski, S. Hoda, and J. N. Hooker. Scheduling multiple factory cranes on a common track. Technical report, Carnegie Mellon University, Pittsburgh, 2012.

[91] F. Pezzella and E. Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2):297–310, Jan. 2000.

[92] D. N. Pham. *Complex Job Shop Scheduling: Formulations, Algorithms and a Healthcare Application*. PhD thesis, University of Fribourg, 2008.

[93] D. N. Pham and A. Klinkert. Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research*, 185(3):1011–1025, Mar. 2008.

[94] L. W. Phillips and P. S. Unger. Mathematical programming solution of a hoist scheduling program. *AIIE Transactions*, 8(2):219–225, 1976.

[95] M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, 2nd edition, 2009.

[96] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 4th edition, 2012.

[97] J. Poppenborg, S. Knust, and J. Hertzberg. Online scheduling of flexible job-shops with blocking and transportation. *European Journal of Industrial Engineering*, 6(4):497–518, 2012.

[98] C. N. Potts and V. A. Strusevich. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, 60:S41–S68, May 2009.

[99] M. Pranzo and D. Pacciarelli. An iterated greedy metaheuristic for the blocking job shop scheduling problem. Technical Report 2, Università degli Studi Roma Tre, Rome, Italy, 2013.

[100] W. Raaymakers and J. Hoogeveen. Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, 126(1):131–151, 2000.

[101] A. Rossi and G. Dini. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23(5):503–516, 2007.

[102] P. Schönsleben. *Integrales Logistikmanagement: Operations und Supply Chain Management innerhalb des Unternehmens und unternehmensübergreifend.* Springer, 6th edition, 2011.

[103] S. Sethi, C. Sriskandarajah, S. G, J. Blazewicz, and K. W. Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, 4(3-4):331–358, 1992.

[104] G. W. Shapiro and H. L. W. Nuttle. Hoist scheduling for a PCB electroplating facility. *IIE Transactions*, 20(2):157–167, 1988.

[105] J. Smith, B. Peters, and A. Srinivasan. Job shop scheduling considering material handling. *International Journal of Production Research*, 37(7):1541–1560, 1999.

[106] F. Sourd and W. Nuijten. Scheduling with tails and deadlines. *Journal of Scheduling*, 2001.

[107] K. Stecke. Design, planning, scheduling, and control problems of flexible manufacturing systems. *Annals of Operations Research*, 3:3–12, 1985.

[108] W. J. Stevenson. *Operations Management.* McGraw-Hill, 11th edition, 2012.

[109] L. Tang, X. Xie, and J. Liu. Scheduling of a single crane in batch annealing process. *Computers & Operations Research*, 36(10):2853–2865, Oct. 2009.

[110] H. W. Thornton and J. L. Hunsucker. A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. *European Journal of Operational Research*, 152(1):96–114, Jan. 2004.

[111] J. J. J. van den Broek. *MIP-based Approaches for Complex Planning Problems.* PhD thesis, Technische Universiteit Eindhoven, 2009.

[112] C. R. Vela, R. Varela, and M. A. González. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*, 16:139–165, 2010.

[113] F. Werner and A. Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58(2):191–211, 1995.

[114] M. Wiedmer. Ein Modellierungstool für Job Shop Scheduling Probleme. Bachelorarbeit, Universität Freiburg, 2012.

[115] Q. Zhang, H. Manier, and M.-A. Manier. A modified shifting bottleneck heuristic and disjunctive graph for job shop scheduling problems with transportation constraints. *Int. Journal of Production Research*, 52(4):985–1002, 2014.