JULIA LANGE

Solution Techniques for the Blocking Job Shop Scheduling Problem with Total Tardiness Minimization

2019

DISSERTATION

Solution Techniques for the Blocking Job Shop Scheduling Problem with Total Tardiness Minimization

zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)

vorgelegt von

Julia Lange, M. Sc.

geb. am 05.03.1988 in Burg

genehmigt durch die Fakultät für Mathematik
der Otto-von-Guericke-Universität Magdeburg

Gutachter: apl. Prof. Dr. Frank Werner
Otto-von-Guericke-Universität Magdeburg
Prof. Dr. Erwin Pesch
Universität Siegen

eingereicht am 07.01.2019

verteidigt am 11.04.2019

Abstract

Planning the course of events and activities constitutes an everyday challenge in many companies. Complex decisions are to be made regarding the allocation of a diverse range of resources and the corresponding processing sequences of tasks. Especially in the manufacturing of highly customized goods and the operation of cost-intensive logistics systems, efficient schedules have an enormous effect on the long-term success of an enterprise. Therefore, the ambitious idea of a reliable decision support for human planners is pursued by researchers in scheduling theory for many decades. While several classical scheduling problems are well studied, the integration of real-world constraints and objectives shows a lack of theoretical understanding. In order to provide a profound study in this research direction, a practically relevant class of job shop scheduling problems is comprehensively investigated with regard to the boundaries of exact solvability by mixed-integer programming techniques, the applicability of well-known heuristic methods and the advantageousness of hybrid matheuristic approaches.

The job shop scheduling problem is known as a combinatorial optimization problem of considerable intricacy, for which even simple variants are proven to be strongly NP-hard. A set of jobs is required to be handled by a set of machines, where every job features an individual technological route of processing. A single processing step of a job is denoted as an operation, which is defined by a designated machine and processing time. In order to implement real-world conditions, a release date is given for every job and the recirculation of jobs is allowed. The absence of intermediate buffers in

the planning situation is considered as a special circumstance, for instance, occurring in the production of huge items, in robotic storage frameworks and in railbound logistics systems. A schedule is to be determined, which assigns certain periods of working time of the required machines to the operations of each job, so that the given restrictions are met. Motivated by an ever-growing need to generate reliable schedules and increase customer satisfaction, the minimization of the total tardiness of all jobs is examined as an optimization criterion of practical relevance.

For the purposes of providing a detailed structural description of the problem under study, on the one hand, and detecting the current boundaries of exact solvability, on the other hand, the blocking job shop scheduling problem with total tardiness minimization (BJSPT) is comparatively modeled by two mathematical formulations. Significantly smaller numbers of required variables and constraints in the optimization program can be reported for one type of sequence-defining variables in contrast to the other. In line with this observation, the computational results obtained by a state-of-the-art mixed-integer programming solver clearly indicate the advantages of the usage of binary ordering variables for all pairs of operations of different jobs requiring the same machine. Furthermore, the experiments show that the capability of exact general-purpose solution techniques do not meet the practical conditions in solvable problem size, solution quality and runtime. Additionally, several instance key measures are proposed in order to characterize the given problems and detect relationships between specific values and required computational effort in the solving process. Based on the results, it can be pointed out that the mean machine utilization rate and the mean machine slack constitute good indicators for the complicatedness of a BJSPT, even though more complex figures are still needed to cover the full range of effects.

The thesis mainly contributes to the research in complex job shop scheduling by introducing and applying a permutation-based heuristic to the BJSPT. First, different classical encodings of a schedule are discussed with respect to redundancy and feasibility. A procedure to construct a feasible schedule from any given permutation of all operations is presented. Subsequently, three different neighborhood structures, which are based on widely-used operators such as interchanges of adjacent operations on a

machine and shifts of operations in the permutation, are defined. It is observed that interchange-based moves cause significant feasibility issues in the construction of neighboring schedules. Therefore, an advanced repair technique is proposed to generate a feasible schedule, which incorporates the predefined move and does not feature a structure arbitrarily different from the initially given solution. Together with this modification scheme, all neighborhoods are applicable in metaheuristic methods to solve the BJSPT, even if the connectivity property cannot be shown due to the involved real-world conditions. The computational results obtained by a simulated annealing algorithm provide empirical evidence for the advantageousness of the usage of a permutation-based heuristic method to find good schedules for the BJSPT in reasonable computation time. Furthermore, it is shown that the problem under study features a fairly rugged search space, a reduction of the set of neighbors is not favorable and the execution of several independent runs of the heuristic procedure is beneficial. However, the applied heuristic solution technique cannot compete with the mixed-integer programming solver in solution quality and computation time for small instances. Moreover considering problems of large size, the heuristically determined feasible schedules improve the results obtained by solving the mathematical models, but the solution quality is not yet satisfactory for practical applications. This is why the idea of combining promising components and beneficial effects of both solution approaches is pursued.

As a pioneering work in this field, a matheuristic technique is proposed for and tested on the BJSPT. Purely mixed-integer programming-based construction schemes and neighborhood structures, which involve solving reduced optimization programs, are integrated into a heuristic framework. Extensive preliminary experiments are conducted to evaluate the individual capability of the most well-known general and scheduling-tailored methods. The best performing mechanisms are accordingly chosen as components of a variable neighborhood search. The computational results clearly highlight matheuristic techniques as a favorable research direction in job shop scheduling. The hybrid solution scheme outperforms the general-purpose solver and the permutation-based heuristic in solution quality and computation time on the given benchmark instances.

Overall, two new solution approaches for the BJSPT are proposed generating high quality schedules for instances of previously critical size. Comprehensive empirical and theoretical studies on the internal structures of models and techniques reveal reasons for the difficulties in solving the problem under study and shed light onto the impact of real-world conditions. Enhancements in research on job shop scheduling are reported and first steps into a promising research direction are made. Altogether, the thesis gives rise to novel options in the development of advanced decision support systems for complex planning situations.

Zusammenfassung

Die Planung verschiedenster Arbeitsabläufe stellt eine tägliche Herausforderung in vielen Unternehmen dar. Komplexe Entscheidungen bezüglich der Verteilung unterschiedlicher Ressourcen und der dazugehörigen Bearbeitungsreihenfolge von Aufgaben müssen getroffen werden. Insbesondere bei der Herstellung individualisierter Güter und im Betrieb von Logistiksystemen mit hoher Kapitalbindung spielen effiziente Ablaufpläne eine wichtige Rolle für den langfristigen ökonomischen Erfolg eines Unternehmens. Aus diesem Grund verfolgen Forscher im Bereich des Scheduling seit vielen Jahrzehnten das ambitionierte Vorhaben einer verlässlichen Entscheidungsunterstützung für strategische Planer. Während einige klassische Probleme des Scheduling gut verstanden sind, fehlen wissenschaftliche Untersuchungen zur Betrachtung vieler praktischer Bedingungen und Ziele. Um diesen Forschungsbereich mit einer fundierten Studie zu erweitern, wird eine praxisrelevante Klasse von Job Shop Scheduling Problemen umfassend in Bezug auf die Grenzen der Lösbarkeit mit exakten Verfahren der gemischt-ganzzahligen Optimierung, die Anwendbarkeit von bekannten heuristischen Methoden und die Vorteilhaftigkeit von hybriden matheuristischen Ansätzen untersucht.

Das Job Shop Problem ist als kombinatorisches Optimierungsproblem von außergewöhnlicher Schwierigkeit bekannt, für das selbst einfache Varianten als NP-schwer klassifiziert sind. Eine Menge von Aufträgen ist von einer Menge von Maschinen zu bearbeiten, wobei jeder Auftrag eine individuelle technologische Reihenfolge des Arbeitsablaufs verlangt. Ein einzelner Bearbeitungsschritt eines Auftrags wird als Operation bezeichnet, die mit einer

Bearbeitungszeit und der notwendigen Maschine gegeben ist. Um praxisrelevante Bedingungen zu integrieren, ist ein Bereitstellungszeitpunkt für jeden Auftrag definiert, und die wiederholte Bearbeitung eines Auftrags auf einer Maschine ist erlaubt. Das Fehlen von Zwischenlagermöglichkeiten im Planungssystem wird als spezielle Einschränkung betrachtet, die beispielsweise bei der Produktion außergewöhnlich großer Güter sowie in automatisierten Lagerkonzepten und schienenbasierten Logistiksystemen zu finden ist. Ein Ablaufplan muss bestimmt werden, der allen Operationen der Aufträge die verfügbaren Arbeitszeiten der benötigten Maschinen zuordnet, so dass die gegebenen Restriktionen erfüllt sind. Motiviert durch das fortwährende Bestreben nach verlässlichen Ablaufplänen und steigender Kundenzufriedenheit wird die Minimierung der Gesamtverspätungszeit als praxisorientiertes Optimierungskriterium untersucht.

Um eine detaillierte theoretische Beschreibung des betrachteten Problems zu geben und anschließend die aktuell bestehenden Grenzen der exakten Lösbarkeit auszuloten, wird das Job Shop Problem mit Blockierungsbeschränkungen und der Minimierung der Gesamtverspätungszeit (BJSPT) vergleichend mit zwei mathematischen Formulierungen modelliert. Für einen der beiden Typen von verwendeten, die Reihenfolge definierenden Variablen kann eine erheblich kleinere Anzahl notwendiger Unbekannter und Nebenbedingungen im Optimierungsmodell festgestellt werden. Übereinstimmend zeigen auch die mit einem der modernsten Lösungsverfahren für gemischt-ganzzahlige Probleme erhaltenen Ergebnisse klare Vorteile der Nutzung von binären Ordnungsvariablen für alle Paare von Operationen unterschiedlicher Jobs, die die gleiche Maschine verlangen. Außerdem belegen die Experimente, dass die Leistungsfähigkeit von allgemeinen exakten Lösungstechniken die praktischen Anforderungen in der handhabbaren Problemgröße, der Qualität der Ablaufpläne und der Rechenzeit nicht erfüllen kann. Des Weiteren werden einige Problemkennzahlen eingeführt, um die gegebenen Instanzen zu charakterisieren und Verbindungen zwischen bestimmten Ausprägungen und dem erforderlichen Rechenaufwand im Lösungsprozess herzustellen. Basierend auf den Ergebnissen kann beobachtet werden, dass die durchschnittliche Nutzungsrate und die mittlere Pufferzeit der Maschinen gute Indikatoren für den Schwierigkeitsgrad eines

Problems darstellen, auch wenn komplexere Kenngrößen zur vollständigen Beschreibung aller Interdependenzen notwendig sind.

Der maßgebliche Beitrag dieser Arbeit zur Forschung an komplexen Job Shop Problemen besteht in der Einführung eines permutationsbasierten Näherungsverfahrens und dessen Anwendung auf das BJSPT. Nach der Diskussion verschiedener klassischer Darstellungsformen von Ablaufplänen, in Bezug auf auftretende Redundanz und Unzulässigkeit, wird ein Verfahren zur Konstruktion eines zulässigen Plans aus jeder beliebigen Liste aller Operationen präsentiert. Anschließend werden drei Nachbarschaftsstrukturen definiert, die auf bekannten Operatoren wie dem Austausch zweier benachbarter Operationen auf einer Maschine und der Verschiebung einer Operation in der Permutation beruhen. Es ist festzustellen, dass austauschbasierte Übergänge erhebliche Probleme in der Zulässigkeit der resultierenden Nachbarlösung verursachen. Hierfür wird ein komplexes Reparaturverfahren konstruiert, das stets einen zulässigen Ablaufplan erzeugt, der den gegebenen Austausch enthält und in seinem Aufbau nicht willkürlich von der Anfangslösung abweicht. Mit dieser Reparaturmethode können alle beschriebenen Nachbarschaften in Metaheuristiken eingebettet und auf das BJSPT angewendet werden, auch wenn aufgrund der praxisrelevanten Bedingungen der theoretische Zusammenhang für keine der Strukturen gegeben ist. Die mit einem Simulated Annealing erzielten Rechenergebnisse unterstützen empirisch die Zweckmäßigkeit der Nutzung einer permutationsbasierten Heuristik zur Generierung guter Ablaufpläne für das BJSPT in vertretbarer Laufzeit. Außerdem zeigt sich, dass das betrachtete Problem einen eher zerklüfteten Lösungsraum aufweist, dass eine Reduktion der Anzahl zu betrachtender Nachbarn nicht vorteilhaft ist und dass sich die Ausführung mehrerer unabhängiger Durchläufe des heuristischen Verfahrens positiv auswirkt. Dennoch kann die angewendete Heuristik besonders für kleine Instanzen nicht mit der erreichten Qualität der Ablaufpläne und der benötigten Rechenzeit des exakten Lösungsverfahrens konkurrieren. Bei der Betrachtung großer Instanzen weisen die heuristisch erzielten Pläne zwar bessere Zielfunktionswerte auf als die der exakten Methode, jedoch genügt die Lösungsqualität den Anforderungen einer realen Planungssituation noch nicht. Aus diesem Grund scheint die

Idee der Kombination von vorteilhaften Komponenten und positiven Effekten der beiden Ansätze erfolgversprechend.

Als erster Vorstoß in diesem Forschungsfeld wird ein matheuristisches Verfahren für das BJSPT konstruiert und getestet. Auf rein gemischt-ganzzahliger Programmierung basierende Konstruktionsschemata und Nachbarschaften, die das wiederholte Lösen von reduzierten Optimierungsmodellen erfordern, werden in eine heuristische Struktur eingebettet. Umfangreiche vorgelagerte Experimente erlauben zunächst die Bewertung der individuellen Leistungsfähigkeit der meist genutzten, allgemeinen und auf das Scheduling zugeschnittenen Verfahren. Darauf aufbauend werden die vielversprechendsten Methoden als Komponenten für eine variable Nachbarschaftssuche gewählt. Die erzielten Rechenergebnisse belegen deutlich, dass die Anwendung von Matheuristiken eine zukunftsweisende Forschungsrichtung im Bereich des Job Shop Scheduling ist. Das hybride Lösungsverfahren übertrifft sowohl die allgemeine gemischt-ganzzahlige Optimierung und als auch die permutationsbasierte Metaheuristik in der Qualität der generierten Ablaufpläne und der notwendigen Rechenzeit für die gegebenen Vergleichsinstanzen.

Insgesamt werden in dieser Arbeit zwei neue Lösungsansätze für das BJSPT präsentiert, die die Generierung von Ablaufplänen hoher Qualität für Instanzen mit vorherig kritischer Problemgröße ermöglichen. Umfassende empirische und theoretische Untersuchungen von Strukturen der Modelle und Methoden legen Gründe für die Schwierigkeiten in der Lösung der betrachteten Probleme offen und geben Aufschluss über die Auswirkungen von praxisrelevanten Bedingungen. Entwicklungen im Bereich des Job Shop Scheduling sind erzielt und erste Schritte in einer vielversprechenden Forschungsrichtung sind getan. Die Arbeit bietet damit neue Möglichkeiten für die Entwicklung verbesserter Systeme zur Entscheidungsunterstützung für komplexe Planungssituationen.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

API	Adjacent pairwise interchange
ART	Advanced repair technique
ATC	Apparent tardiness cost rule
BJSPT	Blocking job shop problem with total tardiness minimization
BRT	Basic repair technique
FD	Fix and dive
FMS	Fix one machine and solve
FP	Feasibility pump
FS	Fix and solve
inf	Infinity
JI	Job insertion
JIF	Job insertion with flexibility
LBR	Local branching
LP	Linear programming
LPC	Local position changes
LS	Local neighborhood search
MF	Machine fixing
MIP	Mixed-integer linear programming
MOD	Modified operation due date rule
RINS	Relaxation induced neighborhood search

SA	Simulated annealing
SJI	Sequential job insertion
SL	Slack rule
SMS	Sequential machine scheduling
SPT	Shortest processing time rule
S/OPN	Slack per operation rule
TAPI	Tardy adjacent pairwise interchange
TJ	Tardy job
VNS	Variable neighborhood search

List of Symbols

General

$\mathbb{R}_{\geq 0}$	the set of non-negative real numbers
$\mathbb{Z}_{\geq 0}$	the set of non-negative integral numbers
$\mathbb{Z}_{>0}$	the set of positive integral numbers

Job Shop Scheduling

$\mathcal{J}$	set of jobs
$\mathcal{M}$	set of machines
$\mathcal{O}$	set of operations
$\mathcal{O}^i$	set of operations of job J_i
Ω^k	set of operations requiring machine M_k
$\mathcal{R}^k$	set of order-positions of machine M_k

C_i	completion time of job J_i
C_{max}	makespan of a schedule
J_i	i-th job
M_k	k-th machine
$O_{i,j}$	j-th operation of job J_i
R_k	number of order-positions of machine M_k

T_i	tardiness time of job J_i
TR_i	technological route of job J_i
block	blocking constraints
d_i	due date (due time) of job J_i
$d_{i,j}$	due time of operation $O_{i,j}$
l_k	machine slack of machine M_k
$\bar{l}$	mean machine slack of all machines $M_k \in \mathcal{M}$
λ_k	inter-arrival time of operations on machine M_k
m	number of machines
n	number of jobs
n_i	number of operations of job J_i
n_{op}	(total) number of operations
$p_{i,j}$	processing time of operation $O_{i,j}$
$\bar{p}_k$	mean processing time of operations on machine M_k
r	order-position
r_i	release date (release time) of job J_i
$r_{i,j}$	release time of operation $O_{i,j}$
recrc	recirculation
$s_{i,j}$	starting time of operation $O_{i,j}$
u_k	utilization rate of machine M_k
$\bar{u}$	mean utilization rate of all machines $M_k \in \mathcal{M}$

Mathematical Modeling

M	sufficiently large positive constant
T	total number of time periods
t	time period $(t-1, t]$
$x_{i,j}^r$	order-position variable assigning operation $O_{i,j}$ to order-position r
$y_{i,j,i',j'}$	precedence variable sequencing operations $O_{i,j}$ and $O_{i',j'}$
$z_{i,j}^t$	time-indexed variable indicating the processing of operation $O_{i,j}$ at time point t

Heuristics

$\mathcal{A}$	set of available operations
$\mathcal{F}$	set of fixed precedence relations between pairs of operations
$\mathcal{N}(s)$	general neighborhood of a schedule s
$\mathcal{N}_A(s)$	API neighborhood of a schedule s
$\mathcal{N}_J(s)$	TJ neighborhood of a schedule s
$\mathcal{N}_T(s)$	TAPI neighborhood of a schedule s
$\mathcal{T}$	set of tardy jobs
$Cand$	candidate list of operations
$H(s, s')$	Hamming distance between two solutions s and s'
P	acceptance probability
Q	queue (list) of operations
$T(s)$	total tardiness of a schedule s
W	swap group of operations
$\alpha(O_{i,j})$	machine predecessor of operation $O_{i,j}$
$\beta(O_{i,j})$	machine successor of operation $O_{i,j}$

$b_{i,j}$	blocking time of operation $O_{i,j}$
c	cooling factor
$const$	a constant number
$\delta(s, s')$	distance between two solutions s and s'
$f_{i,j}$	finishing time of operation $O_{i,j}$
$h_{i,j,i',j'}$	variable indicating a reversed ordering of the operations $O_{i,j}$ and $O_{i',j'}$
$lidx(O_{i,j})$	list index (position) of operation $O_{i,j}$ in the operation-based schedule representation
$lidx'(*)$	currently considered list index
$midx(O_{i,j})$	machine index (order-position) of operation $O_{i,j}$ on machine M_k
$perm$	permutation of all operations $O_{i,j} \in \mathcal{O}$
$prio(O_{i,j})$	priority of operation $O_{i,j}$
s^{job}	job-based representation of a schedule
s^{op}	operation-based representation of a schedule (permutation)
s^{ma}	machine-based representation of a schedule
τ	look-ahead parameter
t	temperature parameter
t_{init}	initial value of temperature t
t_{term}	terminal value of temperature t

Matheuristic

$\mathcal{N}_\varphi(\mathbf{x})$	φ-th neighborhood of a solution $\mathbf{x}$
$T(\mathbf{x})$	total tardiness of a solution $\mathbf{x}$
φ^{max}	maximum number of neighborhoods
$\bar{\mathbf{x}}$	initial or current feasible (mixed-integer) solution

$\mathbf{x}^{LP}$	solution of the linear relaxation of an MIP model
$\mathbf{x}^{N}$	feasible neighboring solution

1 Introduction

'I've always been interested in using mathematics to make the world work better.' [3]

Alvin E. Roth, Nobel Laureate, 2012

Following this ambitious idea of thousands of scientists working in various fields of mathematical optimization, this thesis is intended to shed light onto a highly constrained combinatorial optimization problem. Since people are required to make challenging decisions in complex work environments every day, a theoretical understanding of the underlying problems is absolutely essential to create a reliable technical decision support and

efficient work flows. For this purpose, mathematical modeling, reasoning and solution techniques provide a perfect framework to develop new optimization methods and enhance the implementation of business processes.

1.1 Scheduling Research

Our world consists of a variety of complex events and ongoing technical advancements, which give rise to the occurrence of permanent planning and coordinating activities. This motivates scheduling research to deal with the general allocation of resources over time to fulfill a given set of tasks, efficiently, cf. for instance [25], [72] and [102]. As a multifaceted research field, scheduling applies to classical planning problems in manufacturing and logistics as well as to technical and non-technical assignment decisions in IT infrastructures, data analytics, controlling and marketing. The resources may be represented for instance by machines, work stations, tracks, databases, manpower, energy and money, while the tasks may refer to jobs, trucks, airplanes, digital requests, client queries and realizable projects, cf. [72] and [102]. This thesis generally focuses on work environments in manufacturing systems, where an available restricted set of machines is used to operate given jobs. Hence, the following study considers renewable resources, which are repeatedly and entirely applicable to an infinite sequence of tasks, cf. [72].

The criteria of efficiency are as various as the conceivable applications, whereby most of them are related to resource consumption. For stationary resources, such as machines and tracks, a task consumes a period of working time for its execution and well-known optimization criteria are accordingly processing time- or flow time-related. During the last decades, the consideration of satisfaction and comfort of the people affected by the planning decision has gained more and more importance. Driven by the practical relevance of scheduling problems and with regard to the economic purposes of successfully operating enterprises, a tardiness-based objective, which is highly related to reliability and customer satisfaction, is studied here.

1.2 Shop Scheduling Problems

Machine Scheduling involves deterministic scheduling problems, which appear at the tactical planning level in a manufacturing system. Strategic decisions, such as how many customers of which type to attract and how to assemble machines, tools and work stations at the shop floor, are taken as given, cf. [102]. Due to this, there exists a variety of different problem settings, characteristics and restrictions. Detailed explanations on popular types of machine scheduling problems, their properties, constraints and objective functions are provided for instance in the books of Blazewicz et al. [24], Jaehn and Pesch [72] and Pinedo [102]. According to the literature, the scheduling problems under study are defined as follows, cf.[72]:

Definition 1.1. *A **machine scheduling problem** $\alpha \mid \beta \mid \gamma$ is described by the specification of the resources α, predefined instance characteristics and constraints β and an objective function γ that is to be minimized.*

Resources may be given as a single machine, $\alpha = 1$, a set of m parallel machines, $\alpha = Pm$ or a finite number of individual dedicated machines. The latter planning situation is denoted as *shop scheduling*, where $\alpha \in \{Fm, Jm, Om\}$ indicates a *flow shop problem* Fm, a *job shop problem* Jm or an *open shop problem* Om involving m machines, respectively, cf. [24], [72] and [102]. For all shop scheduling problems, a job consists of a finite number of *operations*, which express the processing of the job by a required machine, cf. [32]. Depending on the existence and structure of a predefined ordering of these operations, all jobs are to be processed on the machines according to a unique sequence in a flow shop, the jobs feature individual sequences of required machines in a job shop or the operations of all jobs may be processed in an arbitrary ordering in an open shop environment. However, the following basic assumptions are taken for all classical shop scheduling problems, cf. [24], [72] and [102].

Basic Assumptions

- There exists a predetermined and finite number of jobs and machines.
- Every job can only be processed on one machine at a time.
- Every machine can only process one job at a time.

- The preemption of operations is not allowed.
- There exist infinite buffers for jobs to be stored prior to, in between and posterior to processing.

The second problem-defining parameter β may either be empty or involve one or more particular job characteristics and constraints of the scheduling problem. The occurrence of specific job properties, such as release dates, due dates or recirculation, can be considered as well as the exclusion of basic assumptions due to preemptive tasks or limited buffers.

The solution to the shop scheduling problem is expected to assign working time periods of machines to jobs, so that the general setting and all restrictions are met, cf. [30]. Since the operations of the jobs compete for resource consumption, a schedule needs to provide a processing sequence, cf. [72]. Based on this, the starting times of all operations are determined as points in time, at which the corresponding preceding operations of the jobs and the preceding operations on the machines are both finished. To uniquely define these starting times, schedules are always considered to be *semi-active* in the following. This implies that no operation can be finished earlier without changing the processing sequence on any machine or, in other words, the processing of every operation begins at the earliest possible starting time, see Pinedo [102] for further explanations.

Definition 1.2. *A **feasible schedule** is defined by the sequences of operations on all machines satisfying the given constraints, which uniquely determine the earliest possible starting times of all operations.*

Among all feasible solutions, a schedule, which features a minimal value of the objective function given by γ, is denoted as optimal. With regard to machine scheduling problems, widely-applied objective functions are either completion time-related, such as the makespan and the total completion time, or due date-related like the total tardiness and the number of tardy jobs. Dependent on the considered application, priorities of jobs may be integrated by job weights into all efficiency measures.

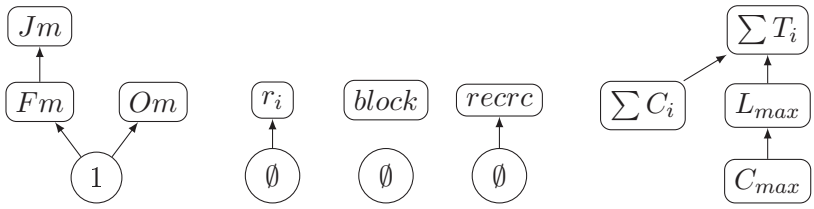
An important characteristic of the optimization criterion is its regularity. Given the objective function is non-decreasing in the completion times of the jobs, it is said to be a *regular performance measure*, cf. [102]. For each

shop scheduling problem featuring a regular optimization criterion, it can be argued that operation starting times, which do not implement a semi-active schedule, can never improve the objective function value compared to a schedule, where the processing of all operations starts as early as possible. Thus, it is sufficient to consider the set of semi-active schedules as the set of feasible solutions, when searching for an optimal schedule given a regular performance measure.

1.3 Job Shop Scheduling

The job shop scheduling problem examined in this thesis includes predefined individual routes of processing for all jobs, cf. for instance [72] and [102], and constitutes one of the hardest combinatorial optimization problems, cf. [13] and [129]. This setting is of high practical relevance, since the production of customized goods with individual processing requirements plays an important role in the range of many enterprises. Furthermore, railbound and robotic manufacturing and logistics systems form well-known applications, cf. [36] and [100], as well as the scheduling of trains in single-track railways networks, cf. [46] and [115].

In the 1970s, several authors have proven the hardness of even simple variants of the job shop scheduling problem using concepts of complexity theory, see [57] for further reading. According to the literature, the makespan of a schedule is denoted by C_{max} in the following, cf. among others [72] and [102]. Garey, Johnson and Sethi [58] show by a transformation of 3-Partition that the decision problem on the existence of a schedule for $Jm \mid\mid C_{max}$ given an upper bound on the objective function value is NP-complete in the strong sense for $m \geq 2$. For reasons of clarity, the ongoing review of complexity results will focus on the optimization problems and consequently transfer the findings on the NP-completeness of a decision problem provided in the literature to NP-hardness. Taking the simplifying assumption of unit processing times for the operations of all jobs, the job shop problem $J3 \mid p = 1 \mid C_{max}$ involving three machines is still proven to be NP-hard in the ordinary sense by the reduction of a knapsack problem by Lenstra and Rinnooy Kan in [81]. Gonzalez and Sahni [61] investigate flow shop and job shop instances with a bounded number of machines and



(a) Resource specifications α **(b)** Instance characteristics and constraints β **(c)** Objective functions γ

Figure 1.1: Complexity hierarchies of shop scheduling problems (in excerpts)

a restricted number n_i of operations for each job. The authors show that $J2 \mid n_i \leq 3 \mid C_{max}$ as well as $J3 \mid n_i \leq 2 \mid C_{max}$ are NP-hard in the ordinary sense for preemptive and non-preemptive schedules adapting a result from Lenstra, Rinnooy Kan and Brucker [82].

Polynomial time algorithms exist for the problem $J2 \mid n_i \leq 2 \mid C_{max}$ involving two machines and at most two operations per job given by Jackson and for $Jm \mid n = 2 \mid C_{max}$ scheduling exactly two jobs on an arbitrary number of machines provided by Akers, cf. [72] and [102]. Since this thesis is intended to approach job shop scheduling instances of practically relevant size, the number of involved machines and the quantities of jobs and operations per job will exceed the boundaries of polynomial solvability. Furthermore, completion time-related criteria are well-studied for many job shop scheduling problems, while real-world planning situations demand for customer-oriented optimization. Therefore, this thesis is focused on a due date-related objective function. According to the complexity hierarchies of shop scheduling problems, which are illustrated in relevant excerpts in Figure 1.1 according to Pinedo [102] and Blazewicz et al. [24], the considered job shop problems are expected to be challenging and of extraordinary intricacy.

In line with the result given by Garey, Johnson and Sethi, part (a) of the figure indicates that the job shop setting Jm involves harder problems than a flow shop environment Fm, which is in turn expected to be more complicated than a single machine problem featuring the same parameters β and γ. Considering characteristics and constraints of practical relevance, the appearance of release dates r_i for all jobs and recirculation ($recrc$) are

proven to increase the complexity of the problem, see part (b) of Figure 1.1. On the contrary, for the integration of blocking constraints (*block*), which implement the absence of intermediate buffers in the scheduling system, there exists no theoretical result on the expected effect of such restrictions. In part (c) of the figure, the complexity hierarchy for different objective functions is illustrated. The minimization of the makespan C_{max} constitutes the simplest and a well-studied optimization criterion for job shop scheduling problems. Involving job due dates and minimizing the maximum lateness L_{max} of the jobs or the total tardiness $\sum T_i$ complicates the job shop problem, significantly. While a relationship between the makespan objective and the minimization of the total completion time $\sum C_i$ has not yet been proven, it can be stated that the minimization of total tardiness is of higher complexity than taking the total completion time as the objective function. Considering these facts, the following questions are proposed.

Question 1. *What are the boundaries of exact solvability of job shop problems featuring practically relevant characteristics and constraints?*

Question 2. *Which instance properties affect the solvability of complex job shop problems?*

Since even the basic job shop setting $Jm \mid\mid C_{max}$ is classified as strongly NP-hard, cf. [58], researchers aim in developing smart heuristic solution techniques to obtain high quality schedules in reasonable runtime. The most popular method to approach $Jm \mid\mid C_{max}$ is the shifting bottleneck heuristic proposed by Adams, Balas and Zawack [2] in 1988. The solution technique is based on the repeated relaxation of the job shop problem to single machine problems $1 \mid r_i \mid L_{max}$ and the stepwise improvement of the best found solution. Since the relaxed problem $1 \mid r_i \mid L_{max}$ is proven to be strongly NP-hard itself and the procedure requires an exact solution of this subproblem, the computation time of the shifting bottleneck method might be considerably large especially for real-world instances, cf. [72]. Unfortunately, this is true for most relaxation-based resolution approaches of job shop scheduling problems, see for instance [31], so that neighborhood-based metaheuristics are studied with increasing intensity during the last decades. Watson et al. [126] and Bierwirth et al. [20] observe well performing heuristic methods in relation to the characteristics

of the search space of the basic problem $Jm \mid\mid C_{max}$ arguing that specific search techniques benefit from the irregularity of the solution space. Nonetheless, Mattfeld and Bierwirth [92] state in 2004 that the structure of promising neighborhoods for job shop problems featuring practically relevant due date-based objective functions is not yet known. Additionally, reliable strategies on how to choose and modify heuristic solution methods to obtain good results for the job shop problem involving release dates, recirculation and blocking constraints do not yet exist. This leads to further research issues examined in this thesis.

Question 3. *Are scheduling-tailored heuristic methods able to produce high quality schedules for complex job shop scheduling problems?*

Question 4. *Do well-known permutation-based representation techniques apply to practically relevant job shop scheduling problems?*

Question 5. *Can a combination of exact solution methods and scheduling-tailored heuristics be beneficial for solving job shop scheduling problems of practical relevance?*

1.4 Methodology and Contribution of this Thesis

The basic job shop scheduling problem $Jm \mid\mid C_{max}$ is a well-understood and strongly NP-hard combinatorial optimization problem, which can be solved quite efficiently by heuristic methods. On the contrary, problem settings with practically relevant objective functions and properties show a lack of structured experiments and conclusive remarks on promising solution techniques. To examine a general and complex job shop scheduling problem, release dates and recirculation are considered as job characteristics while the absence of intermediate buffers is involved as a special property of the machine environment. Additionally, the minimization of the total tardiness is chosen as a sparsely-studied customer-oriented optimization criterion. Comprehensive investigations of the complex job shop scheduling problem $Jm \mid r_i, block, recrc \mid \sum T_i$ are performed to analytically answer the research questions raised in the previous section.

Standard mixed-integer (linear) programming (MIP) software constitutes a widely-applied tool to accurately solve all types of planning problems

in practice. Therefore, it is utilized in the following to generate state-of-the-art benchmark results on the exact resolvability of complex job shop scheduling problems. Different mathematical formulations for job shop problems are provided in the literature, whereby advantages and disadvantages in structure and obtainable solution quality are not precisely examined. In this thesis, three well-known models are analyzed and discussed with regard to the involved sequence-defining variables, the required quantities of variables and constraints as well as the performance of a general-purpose MIP solver. Consequently, the boundaries of exact resolvability are defined based on the best performing mathematical formulation. Furthermore, instance key figures are proposed for the purpose of relating the characteristics of the problems to the computational effort required to obtain an exact optimal solution.

A simple but powerful foundation of many heuristic solution approaches to shop scheduling problems is formed by permutation-based encoding schemes and operators, which are barely applied to the job shop environment. Especially the incorporation of blocking constraints yields almost non-investigated issues. As indicated by the complexity hierarchy in Figure 1.1, it is not clear to which extend blocking constraints complicate or simplify a given shop scheduling problem. In the following, occurring redundancy in permutation-based representations of a schedule is discussed and the problem characteristics causing significant feasibility issues are detected. It is apparently shown that a job shop problem involving blocking constraints is significantly more complex and harder to tackle especially for construction-based procedures compared to the basic job shop setting. Nonetheless, the empirically given necessity for smart heuristic methods is addressed. As the main contribution, this thesis presents a repairing scheme to construct a feasible schedule from any given permutation and an even more advanced technique to generate feasible neighboring solutions defined by pairwise interchanges of operations in the schedule. Correspondingly, three neighborhood structures for the complex job shop problem under study are proposed. All components are implemented and tested in metaheuristic frameworks to evaluate the heuristically obtainable solution quality.

With regard to the last research question, this thesis pioneers in examining the advantageousness of a hybrid approach, which integrates MIP components into a heuristic procedure, for job shop scheduling problems. To first broaden the knowledge on the applicability of different construction schemes and neighborhoods in the generation of high quality schedules, MIP-based construction mechanisms and neighborhood structures are tested according to their performance in an iterative improvement scheme on complex job shop instances. Based on these preliminary experiments, the most promising techniques are combined in a metaheuristic method and the performance of the hybrid approach is reported.

1.5 Outline

The conducted research study on solution techniques for the job shop scheduling problem with blocking constraints and total tardiness minimization is organized as follows:

Chapter 2 motivates the consideration of the problem $Jm \mid r_i, block, recrc \mid \sum T_i$ by various real-world applications and introduces the essential notation for the ongoing scientific investigation. The job shop scheduling problem is stepwise explained, starting from the well-studied standard case, implementing several generalizations and extensions. Thereby, the increasing complexness is examined through first observations on occurring feasibility issues. The set of benchmark instances used in this thesis is set up and several instance key measures are proposed to quantify properties of the problems under study.

In *Chapter 3*, three types of sequence-defining variables are discussed with regard to their applicability in modeling the considered job shop problem. Due to structural arguments, two promising mathematical formulations are chosen to be explicitly described and comparatively tested. Before the performance of a general MIP solver on the models is evaluated, the relaxation-based lower bounds gain special attention as a key feature of every MIP technique. Thereafter, extensive computational experiments are reported and a broad discussion is led concerning the exact solvabil-

ity of complex job shop problems, the performance of the mathematical formulations and the effects of the instance properties.

Driven by an observable necessity to provide scheduling-tailored heuristic solution approaches, *Chapter 4* presents a summary of existing applications of metaheuristics to job shop scheduling problems as a starting point for the construction of a permutation-based heuristic. After that, fundamental characteristics of list encodings of schedules and necessary modifications of general ideas are examined in detail. An adapted procedure to construct a feasible schedule from any given permutation is proposed. To construct successful heuristic techniques, three neighborhood structures are set up based on interchanges and shifts of operations in the permutation. The main appearing difficulty, namely the generation of an extraordinary proportion of infeasible neighboring solutions, is overcome by an advanced repair technique, which is applied together with the neighborhood operators from this point on. Several structural properties of the neighborhoods and advantageous effects in approaching high quality schedules are investigated through the implementation in two generic metaheuristics. Computational results obtained by an iterative improvement scheme and a simulated annealing procedure are reported and compared. In the evaluation, special emphasize is given to contrasting the results of the heuristics with the findings on the MIP technique as well as to potential benefits of a reduction of the search space led by the objective function.

Following the findings on the exact and heuristic solvability of the considered problem, *Chapter 5* addresses a matheuristic solution approach. First, a literature review on the application of such hybrid techniques in machine scheduling is provided. Subsequently, a preliminary study is reported to base the choice of promising components for the proposed method. The performance of various MIP-based construction schemes and neighborhoods is evaluated through individual numerical experiments. A variable neighborhood search is used as a framework to combine beneficial effects of the selected construction and transition mechanisms. Computational results are discussed with regard to the advantageousness of matheuristic methods to solve complex job shop problems.

Chapter 6 summarizes the main theoretical and empirical outcomes of the study. Concluding remarks are given on the scientific findings and on new insights gained by the work provided in this thesis.

2 The Job Shop Scheduling Problem with Blocking Constraints and Total Tardiness Minimization

In this chapter, the job shop problem treated in this thesis is theoretically described by means of different generic concepts, such as the Gantt chart, disjunctive programming and graph-based representations. At first, the subsequent Section 2.1 indicates the practical relevance of specific aspects of such scheduling tasks. To entirely set up the problem formulation, Section 2.2 introduces the general notation of the standard job shop scheduling problem, while particular features are explained in Section 2.3. An overall compact description of the key problem $Jm \mid r_i,\ recrc,\ block \mid \sum T_i$ can be found in Subsection 2.3.3. Distinct feasibility issues, which are caused

by the involved blocking restrictions, are highlighted in Section 2.4. To conclude, Section 2.5 contains the composition of the set of benchmark instances used in the computational experiments.

2.1 Practical Issues in Production Planning and Logistics

The job shop scheduling problem shows a wide range of applications in production planning and logistics. Individual processing requirements of products refer to the production of highly customized goods in make-to-order manufacturing systems. In many practical cases, products of this kind are complex, cost-intensive and huge, such as building components of vehicles, machines or facilities. Thus, there are economic intentions not to store intermediate products due to bonded capital and large investments for providing storage capacity. Additionally, the usage of robotic manufacturing and stacking systems as well as concepts like lean production, where the avoidance of work-in-progress inventory is desired, constitute practical necessities to study scheduling problems without intermediate buffers, cf. [36], [65] and [89]. The lack of storage capacity causes the blockage of machines or general processors in the production or transport system. Real-world examples of job shop scheduling problems without intermediate buffers can be found for instance in steel manufacturing [100], automated warehousing [64] and aircraft scheduling [45].

Another related field, in which the absence of buffers occurs, is railbound logistics. Scheduling trains on track segments of a railway network can be interpreted as scheduling jobs on machines with predefined routes, cf. for instance [46], [77], [78], [83], [84] and [115]. In this problem setting, there is no other possibility to let a train leave a track segment after traversal than entering a consecutive track. Given the case that this next segment is still occupied by another vehicle, the train will block the current track segment until it can continue its journey. The idea of solving train scheduling instances as job shop problems is initially published by Szpigel [115] in 1973 and followed by several other authors, cf. [46], [33], [78], [83], [84] and [134]. There is a strong need to generate good solutions to this complex

scheduling task, since companies operating in the field of railbound logistics take intensive investments and, thus, are encouraged to run their assets efficiently, cf. [83]. Furthermore, there are many people, such as planners on a tactical level and passengers, affected by the planning decisions and requesting well realizable schedules, [46] and [115]. The concept of job shop scheduling offers a well-structured theoretical basis, which is expendable by various problem-specific aspects, cf. [33].

Both areas mentioned above feature a high degree of customer orientation. Nowadays, satisfaction and loyalty of customers is a key facet of successfully operating a manufacturing or transport business. Considering the production of goods, there may exist contractually agreed delivery dates, out-of-stock costs and clients expecting a reliable delivery notification, cf. [76]. Hence, the consideration of generic flow time-oriented objective functions does not completely reflect current economic purposes. With regard to railbound logistics, passengers expect their trains to arrive and depart on time and planners look for an accurate execution of existing timetables so that synchronized events proceed without disruption. Consequently, meeting given schedules and avoiding delays is of high practical relevance in aircraft and train scheduling, cf. [45], [46], [99], [110], [111], [115] and [134]. For these reasons, researchers focus increasingly on approaching tardiness-based objectives to account for customer satisfaction and reliability in existing optimization problems, cf. [78], [92], [123] and [133].

2.2 Introducing the Standard Job Shop Scheduling Problem and its Notation

This section describes the basic structure of the considered scheduling problem theoretically and introduces the main notation applied in this thesis. The following explanations constitute a compact characterization of the standard job shop scheduling problem, which is provided similarly by various authors in the literature. Comprehensive expositions of different aspects of the general problem are given for instance by Brucker and Knust [32], Jaehn and Pesch [72] and by Pinedo [102].

The *standard job shop scheduling problem* involves a set of n jobs $\mathcal{J} = \{J_i \mid i = 1, \ldots, n\}$ and a set of m machines $\mathcal{M} = \{M_k \mid k = 1, \ldots, m\}$. Every instance can easily be characterized by its *size* (n, m). The continuous processing of a job J_i on a machine M_k defines an operation. Thus, the number of operations of a job expresses the number of required processing steps, which is indicated by n_i for $J_i \in \mathcal{J}$. In the standard problem setting, every job is assumed to require processing on every machine exactly once, so that $n_i = m$ holds for all jobs. The symbol $O_{i,j}$ indicates the j-th operation of job J_i with $j = 1, \ldots, n_i$. All processing steps of a job are pooled in a set of operations $\mathcal{O}^i$. The overall set of operations is defined by $\mathcal{O} = \bigcup_{J_i \in \mathcal{J}} \mathcal{O}^i$, where the total number of operations is stated as $|\mathcal{O}| = n_{op}$. With regard to the standard problem setting, the total number of operations can be specified by $n_{op} = n \cdot m$.

An important aspect of the characterization of an instance is the *technological route* TR_i, which is given for every job $J_i \in \mathcal{J}$ as a sequence of machines. Thereby, the required machine is defined for every operation $O_{i,j} \in \mathcal{O}^i$ and sets Ω^k of operations being processed on the same machine can be set up for all $M_k \in \mathcal{M}$. There exist two main quantities related to the operations in the job shop scheduling problem. First, for every operation $O_{i,j} \in \mathcal{O}$, a processing time $p_{i,j} \in \mathbb{Z}_{\geq 0}$ is given. It indicates the number of time units that are required to process the operation. Second, starting times $s_{i,j}$, which mark the beginning of the processing of the corresponding operation $O_{i,j}$, are determined for all operations by the construction of a schedule. Since a job cannot be processed by more than one machine at a time, the starting times of the operations need to be chosen so that the predefined *processing sequence* $O_{i,1} \to O_{i,2} \to \cdots \to O_{i,n_i}$ can be realized without overlapping for each job $J_i \in \mathcal{J}$.

As an example for the standard job shop scheduling problem, the instance SJSP is considered involving the set of jobs $\mathcal{J} = \{J_1, J_2, J_3\}$ and the set of machines $\mathcal{M} = \{M_1, M_2, M_3\}$. The technological routes of the jobs and the processing times of the operations are given as follows:

TR_1:	$M_1 \to M_2 \to M_3$	$p_{1,1} = 2,\ p_{1,2} = 1,\ p_{1,3} = 3$
TR_2:	$M_3 \to M_1 \to M_2$	$p_{2,1} = 1,\ p_{2,2} = 3,\ p_{2,3} = 2$
TR_3:	$M_2 \to M_3 \to M_1$	$p_{3,1} = 2,\ p_{3,2} = 4,\ p_{3,3} = 1$

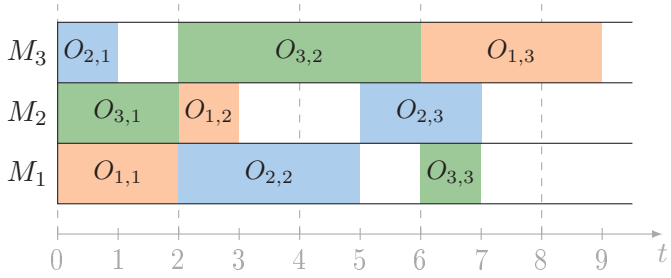


Figure 2.1: Gantt chart representation of a feasible schedule for instance SJSP

Accordingly, the sets of operations requiring the same machine can be derived as $\Omega^1 = \{O_{1,1}, O_{2,2}, O_{3,3}\}$, $\Omega^2 = \{O_{1,2}, O_{2,3}, O_{3,1}\}$ and $\Omega^3 = \{O_{1,3}, O_{2,1}, O_{3,2}\}$.

Figure 2.1 shows a Gantt chart representation of a feasible schedule for SJSP. Observing feasibility, the operations are arranged on the machines, so that the technological routes, the machine requirements and the processing sequences of the jobs are satisfied. The Gantt chart illustrates the earliest possible starting times for all operations as one possible starting time assignment referring to the given operation sequences on the machines. Since the most common optimization criterion for the standard job shop scheduling problem is regular, namely the minimization of the makespan C_{max} (see Section 1.2), unforced idleness of a machine can never improve the schedule with regard to this measure. Thus, a *schedule* is generally and uniquely described by the operation sequences on the machines, whereby the starting times of the operations are automatically defined as the earliest possible points in time for the processing to begin. The makespan of the schedule illustrated in Figure 2.1 is determined by

$$C_{max} = \max\{C_i \mid J_i \in \mathcal{J}\} = \max\{C_1, C_2, C_3\} = \max\{9,\ 7,\ 7\} = 9.$$

Since the standard job shop scheduling problem involves sequencing decisions for all pairs of operations requiring the same machine, it can be described through a disjunctive program as shown in [2]. The formulation is based on the well-known disjunctive graph representation for machine sequencing problems introduced by Balas [10] in 1969. The following op-

timization program models the determination of a feasible schedule with minimal makespan for a standard job shop scheduling instance.

Basic Disjunctive Programming Formulation

$$C_{max} \rightarrow \min! \tag{2.1}$$

subject to

$$s_{i,j} + p_{i,j} \leq s_{i,j+1} \qquad O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,n_i}\}, J_i \in \mathcal{J} \tag{2.2}$$

$$C_{max} \geq s_{i,n_i} + p_{i,n_i} \qquad J_i \in \mathcal{J} \tag{2.3}$$

$$s_{i',j'} \geq s_{i,j} + p_{i,j} \vee s_{i,j} \geq s_{i',j'} + p_{i',j'} \qquad O_{i,j}, O_{i',j'} \in \Omega^k, M_k \in \mathcal{M} \tag{2.4}$$

$$s_{i,j} \geq 0 \qquad O_{i,j} \in \mathcal{O} \tag{2.5}$$

The minimization of the makespan, indicating the optimization criterion, is given in (2.1). Inequality (2.2) implements the predefined processing sequences of the operations of the same job J_i. The makespan C_{max} is determined with Inequality (2.3) as the maximum completion time of all jobs. Constraint (2.4) incorporates a pair of disjunctive inequalities for every pair of operations $O_{i,j}$ and $O_{i',j'}$ that requires processing on the same machine M_k. Either the precedence relation $O_{i,j} \rightarrow O_{i',j'}$ holds on machine M_k and implies $s_{i',j'} \geq s_{i,j} + p_{i,j}$ or $s_{i,j} \geq s_{i',j'} + p_{i',j'}$ is to be true implementing the sequencing decision $O_{i',j'} \rightarrow O_{i,j}$. Inequality (2.5) assures the starting times of the operations to be non-negative. With this, the standard job shop scheduling problem is well described.

2.3 Generalizing and Extending the Standard Job Shop Scheduling Problem

Several adaptations of the standard job shop scheduling problem referring to the practically relevant issues discussed in Section 2.1 are explained in the following. Section 2.3.1 generalizes the standard problem with regard to the technological routes and release dates of the jobs. In Section 2.3.2, the problem is extended by job due dates to implement a tardiness-based

objective function. The generic assumption of infinite buffers in the system is dropped and the absence of storage capacity is considered. To illustrate the properties of the problem, the disjunctive graph representation is used. For general explanations on the structure of directed and undirected graphs as well as related concepts such as paths and cycles, the reader is referred to the books of Diestel [49] and Blazewicz et al. [24]. To summarize, Section 2.3.3 contains a compact description of the problem $Jm \mid r_i,\ recrc,\ block \mid \sum T_i$ considered in this thesis.

2.3.1 Generalizations

In the standard job shop scheduling problem, every job $J_i \in \mathcal{J}$ is processed on every machine $M_k \in \mathcal{M}$ exactly once. This constitutes a special type of technological routes, since $n_i = m$ holds for all jobs and no two operations of the same job are processed on the same machine. In the following, more general technological routes are considered, which differ from the standard case in two aspects. First, a job J_i does not necessarily need to visit all machines $M_k \in \mathcal{M}$. Second, a job J_i may be processed more than once on a particular machine M_k. Thus, the so-called *recirculation* of jobs, cf. [32] and [102], is allowed with one exception. Two consecutive operations $O_{i,j}$ and $O_{i,j+1}$ of the same job J_i shall not be defined to require the same machine M_k, since they can be combined and interpreted as one single operation in this case. Due to this generalization, the disjunctive constraint (2.4) in the basic mathematical formulation presented in Section 2.2 requires an additional condition. These types of sequencing constraints do only need to be set up for operations $O_{i,j}$ and $O_{i',j'}$ belonging to different jobs J_i and $J_{i'}$, since the ordering of pairs of operations of the same job is already defined by the processing sequence.

In the standard job shop scheduling problem, moreover, all jobs $J_i \in \mathcal{J}$ are available at the beginning of the planning horizon. This refers to given release times $r_i = 0$ for all $J_i \in \mathcal{J}$. In the following, the problem is generalized by considering arbitrary *release dates* $r_i \in \mathbb{Z}_{\geq 0}$. To integrate this generalization into the disjunctive programming formulation given in Sec-

tion 2.2, inequalities of the following type is added to the set of constraints.

$$s_{i,1} \geq r_i \qquad J_i \in \mathcal{J} \tag{2.6}$$

The restriction requires that the first operation of every job $O_{i,1}$ is not allowed to start earlier than the release date of the job. Since the release times of the jobs are assumed to be non-negative and the processing times of all operations are defined to be positive, the non-negativity constraint of the starting times of all operation given in (2.5) becomes redundant and can be excluded. To correctly implement the given generalizations, the following assumptions are taken.

Additional Assumptions

- The release dates r_i are deterministic and given in advance as non-negative integers for all $J_i \in \mathcal{J}$.
- The processing times $p_{i,j}$ are deterministic and given in advance as strictly positive integers for all $O_{i,j} \in \mathcal{O}$.
- No two consecutive operations $O_{i,j}$ and $O_{i,j+1}$ require the same machine.

In accordance with the disjunctive programming formulation, the job shop problem can be illustrated by the well-known disjunctive graph model, cf. for instance [2], [10], [32], [102]. Applying the notation given by Brucker and Knust in [32], a *disjunctive graph* $G = (V, E, A)$ is defined as a mixed graph with

- a set V that is composed of $n_{op}+2$ nodes representing all operations $O_{i,j} \in \mathcal{O}$, an artificial source 0 and an artificial sink $*$,
- a set E that consists of (undirected) edges representing all pairs of operations $O_{i,j}$ and $O_{i',j'}$ with $i \neq i'$ requiring the same machine $M_k \in \mathcal{M}$ and
- a set A that involves (directed) arcs expressing predefined precedence relations between all pairs of operations $O_{i,j}$ and $O_{i,j+1}$ of the same job as well as between the source node 0 and all first operations $O_{i,1}$ and between the sink node $*$ and all last operations O_{i,n_i} for all $J_i \in \mathcal{J}$.

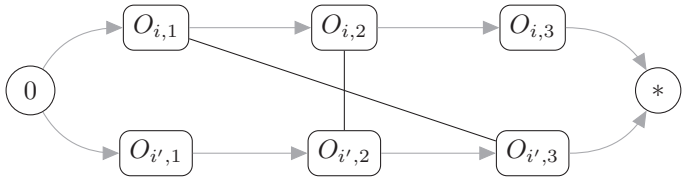


Figure 2.2: Disjunctive graph representation of the instance GJSP1

As an example, consider the general instance GJSP1 with two jobs J_i and $J_{i'}$ consisting of three operations each and four machines $M_{k_1}, M_{k_2}, M_{k_3}$ and M_{k_4}. The corresponding disjunctive graph $G = (V, E, A)$ is given in Figure 2.2. The node set V includes eight nodes and the gray arcs illustrate the set of arcs A. There exist two directed $0 - *-$paths in the graph G representing the processing sequences of J_i and $J_{i'}$, respectively. The edge set E consists of the two edges $\{O_{i,2}, O_{i',2}\}$ and $\{O_{i,1}, O_{i',3}\}$, which indicate that the two arbitrarily chosen pairs of operations $O_{i,2}$ and $O_{i',2}$ as well as $O_{i,1}$ and $O_{i',3}$ require the same machines. Assume that the operations $O_{i,2}$ and $O_{i',2}$ need to be processed on machine M_{k_1}, while the operations $O_{i,1}$ and $O_{i',3}$ require machine M_{k_2}. Further, let the operation $O_{i',1}$ be processed on machine M_{k_3} and the operation $O_{i,3}$ on machine M_{k_4}.

The existence of edges in the disjunctive graph shows the existence of pairs of operations that require an ordering decision. Thus, a complete schedule, for which the operation sequences on the machines are properly determined, refers to a disjunctive graph, where the set of edges is empty and the initial set of arcs is extended by one sequence-defining arc for each pair of operations requiring the same machine. In particular, the disjunctive graph represents a *feasible* schedule for a job shop scheduling problem if and only if it corresponds to an acyclic directed graph, cf. [32] and [102]. This implies that in the determination of a feasible schedule, every edge needs to be replaced by exactly one arc, so that the resulting graph is acyclic. Figure 2.3 sketches a feasible schedule for GJSP1 assuming equal processing times for all operations. The operation sequences $O_{i,2} \to O_{i',2}$ and $O_{i,1} \to O_{i',3}$ are implemented on the machines M_{k_1} and M_{k_2}, respectively. The corresponding acyclic graph is shown in Figure 2.4. In contrast to the disjunctive graph representation of the problem in Figure 2.2, the

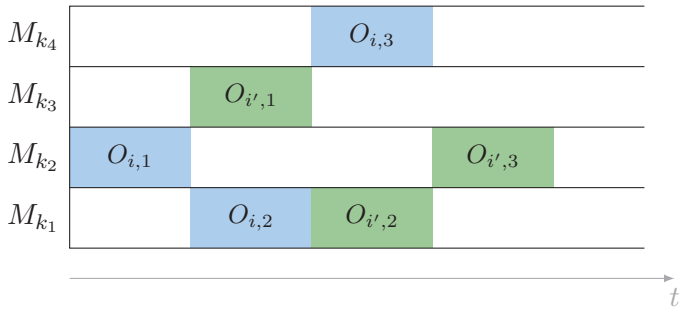


Figure 2.3: Gantt chart representation of a feasible schedule for the instance GJSP1

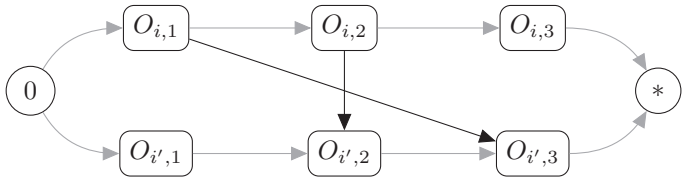


Figure 2.4: Disjunctive graph representation of the feasible schedule for the instance GJSP1

set of edges E is empty and the set of arcs A does additionally include the arcs $(O_{i,2}, O_{i',2})$ and $(O_{i,1}, O_{i',3})$.

For practical applications of job shop scheduling algorithms, an efficient implementation of the underlying graph structure is of high importance. However, since this is not the main focus of this thesis, the reader is referred to [26] for further information.

The generalized job shop scheduling instance GJSP2 is introduced in the following. It differs from the standard instance SJSP given in Section 2.2 and the general instance GJSP1 in the occurrence of recirculation, different numbers of operations of the jobs and non-zero release dates. GJSP2 involves five jobs $J_1, \ldots, J_5$ and three machines M_1, M_2, M_3 and will repeatedly be used throughout this study for different explanations.

Generalized Job Shop Scheduling Instance GJSP2

set of jobs	$\mathcal{J} = \{J_i \mid i = 1, \ldots, 5\}$
set of machines	$\mathcal{M} = \{M_k \mid k = 1, 2, 3\}$

Job J_i	Technological route TR_i	n_i	r_i	$p_{i,j}$ for all $O_{i,j} \in \mathcal{O}^i$
J_1	$M_1 \to M_3$	2	2	$p_{1,1} = 3, p_{1,2} = 1$
J_2	$M_2 \to M_3$	2	0	$p_{2,1} = 1, p_{2,2} = 2$
J_3	$M_1 \to M_2 \to M_3$	3	2	$p_{3,1} = 2, p_{3,2} = 4, p_{3,3} = 1$
J_4	$M_2 \to M_1 \to M_2$	3	3	$p_{4,1} = 1, p_{4,2} = 2, p_{4,3} = 3$
J_5	$M_3 \to M_2$	2	0	$p_{5,1} = 2, p_{5,2} = 2$

The instance incorporates three jobs J_1, J_2 and J_5 that do not require processing on each machine of the machine set. Furthermore, the job J_4 needs to be processed twice on machine M_2, so that its technological route involves a recirculation. The release dates of the jobs J_1, J_3 and J_4 are strictly greater than zero, so that the processing of these jobs is not allowed to start at the beginning of the planning horizon $t = 0$.

Figure 2.5 shows the disjunctive graph $G = (V, E, A)$ representing instance GJSP2. The set of nodes V consists of 14 nodes, which indicate all operations $O_{i,j} \in \mathcal{O}$ as well as the source 0 and the sink $*$. The set of arcs A is formed by the gray-colored arcs implementing the processing sequences of the jobs. Consequently, every job $J_i \in \mathcal{J}$ is illustrated by a directed $0 - *$−path in the mixed graph G, whereby these paths are pairwise disjoint. The set of edges E contains 18 elements depicting the sequencing decisions to be made on the machines. There exists one pair of operations $O_{4,1}$ and $O_{4,3}$ that requires the same machine but is not connected by an edge, since these operations belong to the same job and their ordering is predefined. To improve the perceivability of Figure 2.5, the sets of operations Ω^k and their corresponding edges are highlighted in green and with dotted lines for machine M_1, in blue and with solid lines for machine M_2 and in red and with dashed lines for machine M_3.

A feasible schedule for the given job shop scheduling problem is illustrated in a Gantt chart in Figure 2.6. The release dates of the jobs are indicated by black triangles on top of the diagram. Since the operation sequences

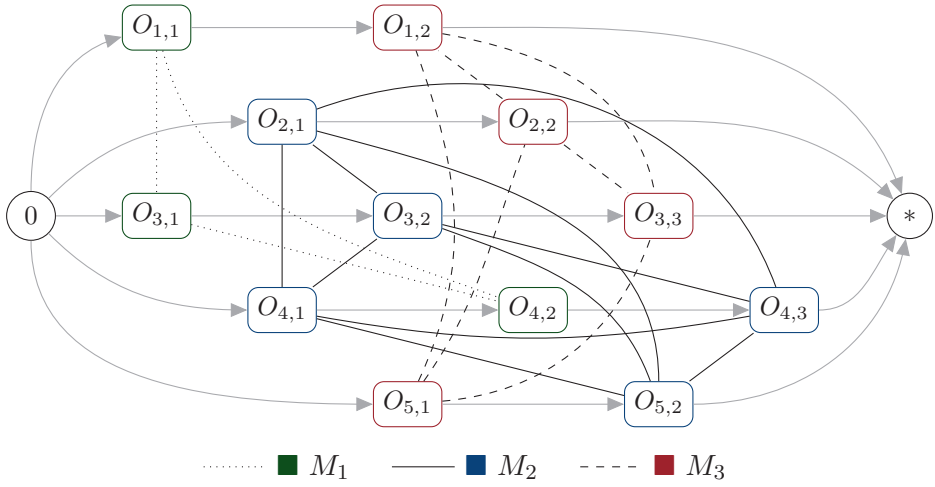


Figure 2.5: Disjunctive graph representation of the instance GJSP2

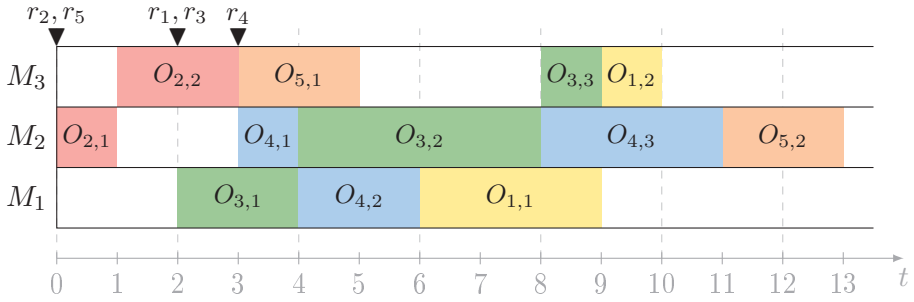


Figure 2.6: Gantt chart of a feasible schedule for GJSP2

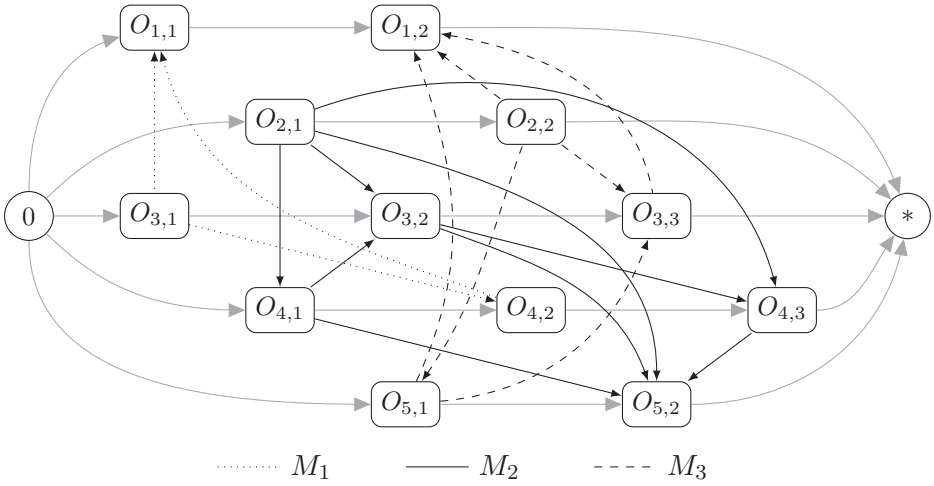


Figure 2.7: Disjunctive graph representation of the feasible schedule of GJSP2

$O_{3,1} \to O_{4,2} \to O_{1,1}$ on machine M_1, $O_{2,1} \to O_{4,1} \to O_{3,2} \to O_{4,2} \to O_{5,2}$ on machine M_2 and $O_{2,2} \to O_{5,1} \to O_{3,3} \to O_{1,2}$ on machine M_3 satisfy the processing sequences and technological routes, the earliest starting time schedule can be set up with regard to these release dates. Due to the ordering requirements, there are idle times on all three machines occurring and in accordance to the assumption stated in Section 1.2 the jobs J_4 and J_5 are stored in an infinite buffer from $t = 6$ to $t = 8$ and from $t = 5$ to $t = 11$, respectively. Transferring the feasible schedule to the disjunctive graph, all edges of the set E are replaced by arcs following the operation sequences on the machines given above. The resulting acyclic directed graph is shown in Figure 2.7, where the arcs referring to precedence relations on the same machine are again marked by dotted, solid and dashed lines.

2.3.2 Extensions

In the standard job shop scheduling problem, the most common optimization criterion is the minimization of the makespan C_{max}. With regard to practically relevant aspects like customer satisfaction and contractually agreed delivery dates, see Section 2.1, it is reasonable to extend the problem by predefined due dates and to consider a tardiness-based objective

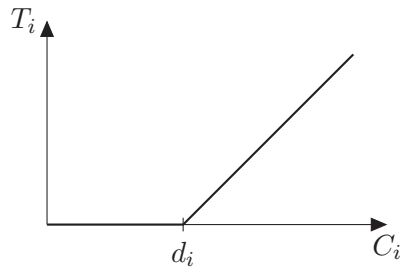


Figure 2.8: The tardiness of a job as a regular performance measure

function. Therefore, a *due date* $d_i \in \mathbb{Z}_{\geq 0}$ is additionally given for each job $J_i \in \mathcal{J}$ and the minimization of the *total tardiness* is considered as the optimization criterion in this thesis. The following assumption regarding the due dates of the jobs is added to the description of the problem.

Additional Assumption

- The due dates d_i are deterministic and given in advance as non-negative integers for all $J_i \in \mathcal{J}$.

The total tardiness is determined as the summation of the tardiness T_i of all jobs $J_i \in \mathcal{J}$, where

$$T_i = \max\{C_i - d_i, 0\}.$$

This implies that a job J_i will only be considered to be tardy, if its completion time C_i, which refers to the end of the processing of its last operation O_{i,n_i}, is strictly greater than its due date d_i. The relationship between the completion time, the due date and the tardiness of a job is graphically shown in Figure 2.8, cf. [72] and [102]. It can be observed that the tardiness T_i is a monotonically non-decreasing function in the completion time C_i for every $J_i \in \mathcal{J}$. Thus, the total tardiness is monotonically non-decreasing in the total completion time of all jobs. This reasons the statement, shortly given in Section 1.2, that the minimization of total tardiness constitutes a regular performance measure, cf. [32], [72] and [102]. Consequently, the operation sequences on the machines and the earliest starting times of the operations can substitutively be used as unique descriptions of a schedule for the job shop scheduling problem with total tardiness minimization.

The considered extension is implemented in the disjunctive programming formulation of the job shop scheduling problem in the following way.

Extended Disjunctive Programming Formulation

$$\sum_{J_i \in \mathcal{J}} T_i \rightarrow \min! \tag{2.7}$$

subject to

$$\begin{aligned}
T_i &\geq C_i - d_i && J_i \in \mathcal{J} && (2.8)\\
T_i &\geq 0 && J_i \in \mathcal{J} && (2.9)\\
C_i &= s_{i,n_i} + p_{i,n_i} && J_i \in \mathcal{J} && (2.10)\\
s_{i,1} &\geq r_i && J_i \in \mathcal{J} && (2.11)\\
s_{i,j} + p_{i,j} &\leq s_{i,j+1} && O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,n_i}\}, J_i \in \mathcal{J} && (2.12)\\
s_{i',j'} \geq s_{i,j} + p_{i,j} \;\vee\; s_{i,j} &\geq s_{i',j'} + p_{i',j'} && O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', && \\
& && M_k \in \mathcal{M} && (2.13)
\end{aligned}$$

The minimization of the total tardiness is stated as the optimization criterion in (2.7). The Inequalities (2.8) and (2.9) depict a linear formulation of the maximum operator in the determination of the tardiness T_i for $J_i \in \mathcal{J}$. Equation (2.10) specifies the completion times C_i for all jobs $J_i \in \mathcal{J}$ and the integration of the general release dates $r_i \in \mathbb{Z}_{\geq 0}$ is given by Inequality (2.11). The constraints (2.12) and (2.13) implement the processing sequences of the jobs and the disjunctive starting time relations for pairs of operations of different jobs requiring the same machine, respectively. Recalling the arguments given in Section 2.3.1, an explicit formulation of the non-negativity constraint on the starting times $s_{i,j}$ for all operations $O_{i,j} \in \mathcal{O}$ is not required due to (2.11) and (2.12).

The instance GJSP2 presented in Section 2.3.1 is extended by the following due dates:

$$d_1 = 7,\ d_2 = 4,\ d_3 = 11,\ d_4 = 11,\ d_5 = 5.$$

Evaluating the feasible schedule for GJSP2 illustrated in Figure 2.6, the completion times of the jobs are determined with

$$C_1 = 10,\ C_2 = 3,\ C_3 = 9,\ C_4 = 11,\ C_5 = 13.$$

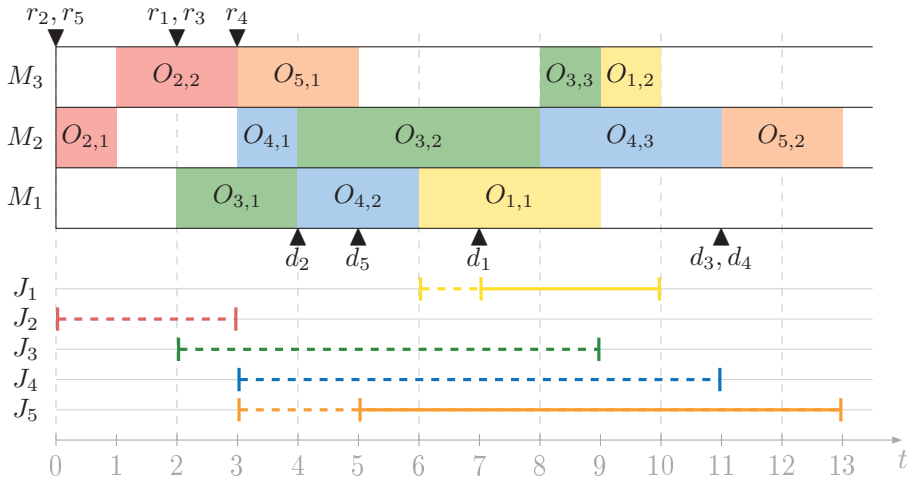


Figure 2.9: Illustration of the total tardiness of a feasible schedule for GJSP2

Figure 2.9 shows the Gantt chart of this feasible schedule, whereby the due dates are additionally indicated by black triangles below the graph. Furthermore, the processing times of the jobs are represented row-wise by colored line segments at the bottom of the figure. Dashed segments express a processing of the job prior to its due date, while solid segments visualize processing periods taking place when the job is already tardy. Thus, if the processing of a job J_i is indicated by an entire dashed line, the job will be completed before or exactly at its due date and $C_i - d_i \leq 0$ holds. This is true for the jobs J_2, J_3 and J_4 in the given schedule. Consequently, these jobs are completed on time with $T_2 = T_3 = T_4 = 0$. The processing of the jobs J_1 and J_5 is started prior to their due dates, but they are not completed by then. Job J_1 is tardy by three units of time, $T_1 = 3$, since its completion time is $C_1 = 10$, while its due date $d_1 = 7$ is given. For job J_5, the tardiness is equally determined with $T_5 = 8$. Here, the total tardiness of all jobs can graphically be interpreted as the summation of all solid line segments in the diagram. The objective function value of the given schedule is defined as follows.

$$\sum_{J_i \in \mathcal{J}} T_i = T_1 + T_2 + T_3 + T_4 + T_5 = 3 + 0 + 0 + 0 + 8 = 11$$

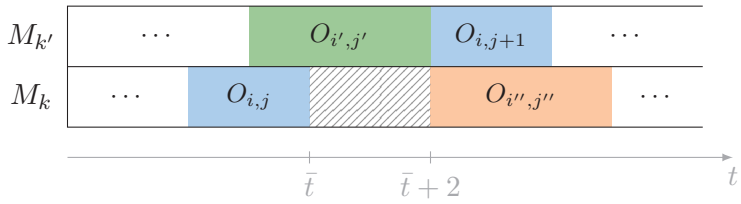


Figure 2.10: Illustration of a job J_i blocking a machine M_k

For a graphical integration of the total tardiness calculation into the disjunctive graph representation, the reader is referred to Kuhpfahl and Bierwirth [76] and Pinedo [102] among others.

Another important practical aspect, which leads to an extension of the job shop scheduling problem, is the absence of intermediate storage capacity in the considered system, see Section 2.1. Since the existence of infinite buffers constitutes a basic assumption in machine scheduling, cf. Section 1.2, this aspect induces additional restrictions. The basic assumption is excluded and replaced by the following statement.

Replaced Assumption

- There exist no intermediate buffers for jobs to be stored in between their processing sequence.

Thus, the considered job shop scheduling problem is extended by so-called *blocking constraints*, cf. [32] and [102]. If the processing of an operation $O_{i,j}$ is completed on M_k but the consecutively required machine $M_{k'}$ is occupied, the job J_i will block the current machine M_k until the succeeding operation $O_{i,j+1}$ can be processed on machine $M_{k'}$. Consequently, the processing of the machine successor of operation $O_{i,j}$ cannot start earlier on machine M_k than the processing of its job successor $O_{i,j+1}$ on machine $M_{k'}$. Figure 2.10 illustrates the described generic situation of a *blocking job shop scheduling problem*. The job J_i, drawn in blue, blocks the machine M_k for two units of time, since the processing of operation $O_{i,j}$ is completed at $t = \bar{t}$, but the consecutively required machine $M_{k'}$ is occupied by operation $O_{i',j'}$ until the point $\bar{t}+2$. The blocking period is indicated by a hatched area in the Gantt chart. As a consequence, the starting time of operation $O_{i'',j''}$ on machine M_k needs to be equal to or greater than the starting time

of operation $O_{i,j+1}$ on machine $M_{k'}$. This induced starting time relation between two operations of different jobs on different machines constitutes the actual blocking constraint. According to the unchanged structure of the sequencing problem, starting time relations of this type are added as disjunctions to the set of constraints in the disjunctive programming formulation for each pair of operations of different jobs requiring the same machine. If one of the considered operations constitutes the last operation of its job, it will depend on the chosen ordering whether a true starting time relation will be introduced. The resulting inequalities are set up as two classes of disjunctive constraints in the following way.

$$s_{i',j'} \geq s_{i,j+1} \vee \quad s_{i,j} \geq s_{i',j'+1} \qquad O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', \; j \neq n_i, j' \neq n_{i'}, M_k \in \mathcal{M} \tag{2.14}$$

$$s_{i',j'} \geq 0 \qquad \vee \; s_{i,n_i} \geq s_{i',j'+1} \qquad O_{i,n_i}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', \; j' \neq n_{i'}, M_k \in \mathcal{M} \tag{2.15}$$

Since these blocking restrictions extend the constraint set of the considered problem, the schedule of instance GJSP2 presented in the Figures 2.6 and 2.9 is not necessarily feasible for the blocking job shop scheduling problem. As observed in Section 2.3.1, the jobs J_4 and J_5 are stored in an intermediate buffer for different periods of time. According to the replaced assumption, this buffer does not exist for the blocking job shop case and the schedule is infeasible with regard to the blocking constraints.

Considering the first storing necessity, the jobs J_1, J_3 and J_4 on the machines M_1 and M_2 are involved. The operation $O_{4,2}$ is completed at $t = 6$ on machine M_1. The successor operation $O_{4,3}$ requires machine M_2, which is still occupied by operation $O_{3,2}$. Thus, job J_4 will block machine M_1 until the processing of operation $O_{3,2}$ is finished at $t = 8$ and the starting time of operation $O_{1,1}$ needs to be shifted to this time. Here, the given operation sequences on the machines are feasible with regard to blocking constraints, but an adaption of the earliest starting times of the operations is required. A partial schedule exclusively including the jobs J_1, J_3 and J_4 and indicating the blocking time on machine M_1 is shown in Figure 2.11. The completion time of job J_1 increases by two units of time to $C_1 = 12$.

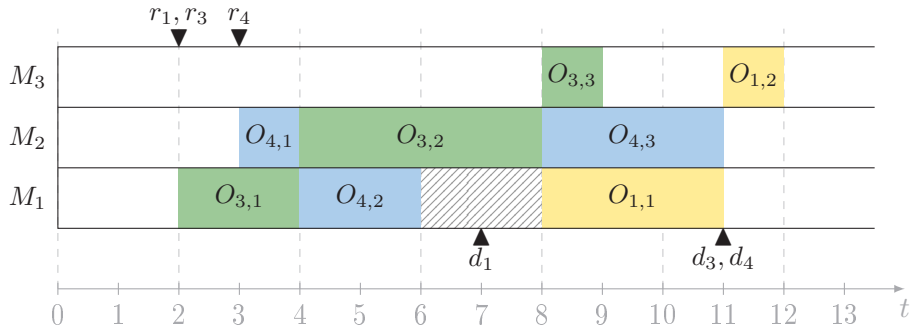


Figure 2.11: Partial schedule for GJSP2 with blocking constraints

Regarding the second storing necessity of the schedule in Figure 2.9, the jobs J_3, J_4 and J_5 are to be observed on the machines M_2 and M_3. Operation $O_{5,1}$ is completed on machine M_3 at $t = 5$ and the consecutively required machine M_2 is occupied by operation $O_{3,2}$. Due to the given operation sequence $O_{3,2} \rightarrow O_{4,3} \rightarrow O_{5,2}$ on machine M_2, job J_5 cannot directly follow operation $O_{3,2}$ but needs to wait further for operation $O_{4,3}$ to be processed. However, the processing of operation $O_{4,3}$ on machine M_2 can only be started, when the processing of operation $O_{3,3}$ is started on machine M_3, which is still blocked by job J_5. In this case, an adaption of the earliest starting times of the operations is not sufficient to construct a feasible schedule for the blocking job shop scheduling problem. The partial operation sequences $O_{5,1} \rightarrow O_{3,3}$ on machine M_3 and $O_{3,2} \rightarrow O_{4,3} \rightarrow O_{5,2}$ on machine M_2 are infeasible with regard to blocking constraints. More precisely, they imply the following cyclic and intransitive starting time relations.

$$(s_{3,3} \geq s_{5,2} \geq s_{4,3} + p_{4,3}) \wedge (s_{4,3} \geq s_{3,3})$$

This structural feasibility issue is explained in detail in Section 2.4 and an algorithmic procedure to obtain feasible schedules for the blocking job shop scheduling problem is developed throughout this thesis. Nonetheless, it is clear at this point that a change in the operation sequences on the machines is required to construct a feasible schedule for instance GJSP2. Here, the adaptation of the operation sequence on machine M_3 to $O_{2,2} \rightarrow O_{3,3} \rightarrow O_{5,1} \rightarrow O_{1,2}$ is sufficient. The resulting feasible schedule is shown in Figure 2.12. The completion time of job J_1 increases compared to the

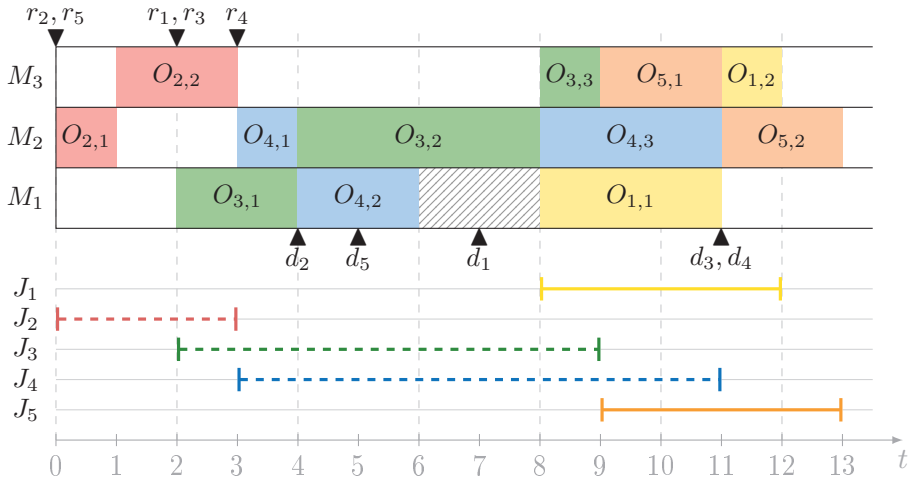


Figure 2.12: A feasible schedule for the instance GJSP2 with blocking constraints

problem formulation without blocking constraints, while the completion times of the other jobs remain unchanged. The total tardiness of the feasible schedule for GJSP2 with blocking constraints is determined as follows.

$$\sum_{J_i \in \mathcal{J}} T_i = T_1 + T_2 + T_3 + T_4 + T_5 = 5 + 0 + 0 + 0 + 8 = 13$$

To graphically implement the structural interdependencies induced by blocking constraints in the job shop scheduling problem, the generic disjunctive graph representation requires adaptations. While the basic disjunctive constraints given in (2.13) only include the starting and processing times of the considered operations $O_{i,j}$ and $O_{i',j'}$, the disjunctive blocking constraints stated in (2.14) and (2.15) involve the starting times of at most four operations $O_{i,j}$, $O_{i,j+1}$, $O_{i',j'}$ and $O_{i',j'+1}$. Thus, the representation of disjunctive blocking constraints through edges between pairs of operations, as it is shown for the basic disjunctive constraints in Figure 2.2, is not possible. Masics and Pacciarelli firstly introduce an *alternative graph representation* for the blocking job shop scheduling problem in [87], which is used for graph-based explanations in the following, cf. also [32]. The alternative graph $G = (V, A)$ is directed, since all kinds of disjunctive

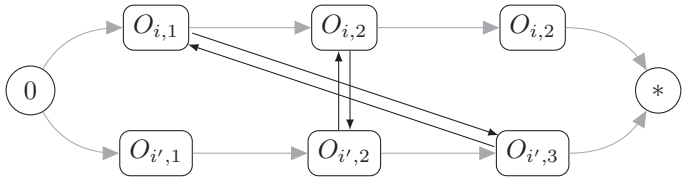


Figure 2.13: Arc-based disjunctive graph representation of the instance GJSP1

constraints are not illustrated by edges but by pairs of arcs. Depicting disjunctive constraints with anti-parallel arcs is generally presented by Pinedo [102] for the job shop scheduling problem without blocking constraints. Considering the general instance GJSP1 introduced in Section 2.3.1, the basic disjunctive constraints involving the pairs of operations $O_{i,2}$ and $O_{i',2}$ as well as $O_{i,1}$ and $O_{i',3}$ are expressed by pairs of so-called disjunctive arcs, cf. [32] and [102], as shown in Figure 2.13. Additionally, the following blocking constraints are introduced.

$$\begin{aligned} s_{i,2} \geq s_{i',3} \wedge s_{i',2} \geq s_{i,3} \quad & \text{for } O_{i,2} \text{ and } O_{i',2} \text{ on } M_{k_1} \text{ with } i \neq i' \\ s_{i',3} \geq s_{i,2} \wedge s_{i,1} \geq 0 \quad & \text{for } O_{i,1} \text{ and } O_{i',3} \text{ on } M_{k_2} \text{ with } i \neq i' \end{aligned}$$

Figure 2.14 depicts the arc-based disjunctive graph for GJSP1, where all pairs of disjunctive constraints implementing starting time relations are included. With regard to the operations $O_{i,2}$ and $O_{i',2}$ both requiring machine M_{k_1}, it can be observed that the basic disjunctive starting time relation indicated by solid arcs becomes redundant through the insertion of the disjunctive blocking constraint marked by the dotted arcs. Thus, the basic disjunctive constraint can be replaced by the disjunctive blocking constraint, cf. [32] and [87]. Considering the operations $O_{i,1}$ and $O_{i',3}$ processed on machine M_{k_2}, there is only one starting time relation induced by the blocking constraint and given as a dashed arc, since operation $O_{i',3}$ constitutes the last operation of job $J_{i'}$. Hence, the solid arc $(O_{i,1}, O_{i',3})$ becomes redundant due to the dashed blocking constraint arc, while the other solid arc $(O_{i',3}, O_{i,1})$ remains active. Altogether, there exists exactly one pair of arcs, named alternative arcs, cf. [32] and [87], for each pair of operations requiring the same machine. The resulting alternative graph for the general instance GJSP1 is shown in Figure 2.15. The potential sequencing decisions concerning the operations $O_{i,2}$ and $O_{i',2}$ requiring machine

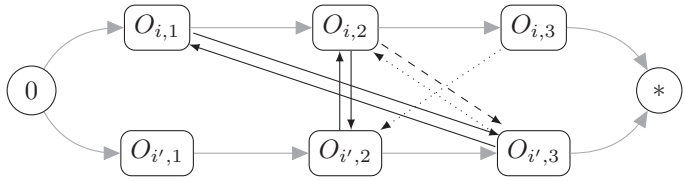


Figure 2.14: Illustration of all pairs of disjunctive arcs for GJSP1

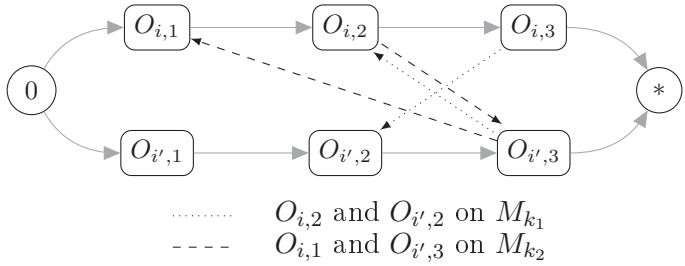


Figure 2.15: Alternative graph representation of the general instance GJSP1

M_{k_1} are drawn as dotted arcs, while the potential orderings induced by the operations $O_{i,1}$ and $O_{i',3}$ requiring machine M_{k_2} are plotted as dashed arcs. Similarly to the disjunctive graph representation, a schedule is defined by a selection of exactly one arc out of every pair of alternative arcs, cf. [32] and [87]. Specific properties of selections defining feasible schedules are discussed in the subsequent section on aspects of feasibility.

Figure 2.16 shows the alternative graph $G = (V, A)$ of the instance GJSP2 of the blocking job shop scheduling problem introduced in Section 2.3.1. Equivalent to the disjunctive graph depicted in Figure 2.5, the set V of the alternative graph contains 14 nodes. The set of arcs A of the alternative graph involves 17 arcs implementing the processing sequences and 36 alternative arcs. A similar graph-based representation for the blocking job shop scheduling problem additionally incorporating set-up and transfer times can be found in [34], [36] and [64].

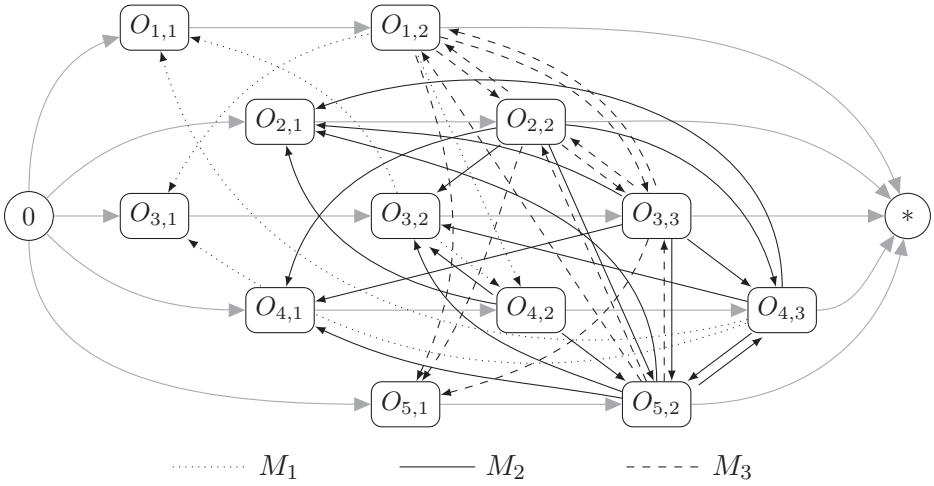


Figure 2.16: Alternative graph representation of the instance GJSP2

2.3.3 The Considered Blocking Job Shop Scheduling Problem with Total Tardiness Minimization

Combining all aspects explained in the foregoing Sections 2.2, 2.3.1 and 2.3.2, the optimization problem treated in this thesis is the *blocking job shop scheduling problem with total tardiness minimization* (BJSPT), where release dates are given for all jobs and recirculation is allowed. Applying the well-known three field notation introduced by Graham et al. in [62], the problem is characterized by

$$Jm \mid r_i, recrc, block \mid \sum T_i.$$

The BJSPT involves sequencing decisions, which induce starting time restrictions for all pairs of operations of different jobs requiring the same machine. These constraints are of a disjunctive structure, so that the optimization program is described by a disjunctive programming formulation.

Disjunctive Programming Formulation of the BJSPT

$$\sum_{J_i \in \mathcal{J}} T_i \rightarrow \min! \tag{2.16}$$

subject to

$$T_i \geq C_i - d_i \qquad J_i \in \mathcal{J} \tag{2.17}$$

$$T_i \geq 0 \qquad J_i \in \mathcal{J} \tag{2.18}$$

$$C_i = s_{i,n_i} + p_{i,n_i} \qquad J_i \in \mathcal{J} \tag{2.19}$$

$$s_{i,1} \geq r_i \qquad J_i \in \mathcal{J} \tag{2.20}$$

$$s_{i,j} + p_{i,j} \leq s_{i,j+1} \qquad O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,n_i}\}, J_i \in \mathcal{J} \tag{2.21}$$

$$s_{i',j'} \geq s_{i,j} + p_{i,j} \ \vee \ s_{i,j} \geq s_{i',j'} + p_{i',j'} \qquad O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', M_k \in \mathcal{M} \tag{2.22}$$

$$s_{i',j'} \geq s_{i,j+1} \ \vee \ s_{i,j} \geq s_{i',j'+1} \qquad O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', j \neq n_i, j' \neq n_{i'}, M_k \in \mathcal{M} \tag{2.23}$$

$$s_{i',j'} \geq 0 \ \vee \ s_{i,n_i} \geq s_{i',j'+1} \qquad O_{i,n_i}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', j' \neq n_{i'}, M_k \in \mathcal{M} \tag{2.24}$$

The optimization criterion, namely the minimization of the total tardiness, is given in (2.16). The tardiness T_i is determined by the Inequalities (2.17) and (2.18) for every job $J_i \in \mathcal{J}$, while the required completion time C_i of each job is defined in (2.19). Inequality (2.20) assures the start of the processing of a job J_i not to be earlier than its release date r_i. The processing sequence of every job is implemented by the Inequality (2.21) for all $J_i \in \mathcal{J}$. The first set of disjunctive constraints, given in (2.22), incorporates the starting time relations for each pair of operations of different jobs requiring the same machine, when one of the potential orderings $O_{i,j} \rightarrow O_{i',j'}$ and $O_{i',j'} \rightarrow O_{i,j}$ is realized. The blocking constraints are similarly stated by disjunctive inequalities in (2.23) and (2.24). Constraint (2.23) declares that the processing of an operation cannot start earlier on a machine than the beginning of the processing of the job successor of its machine predecessor, provided that the job successor exists. In case that one operation of the pairing constitutes the last operation of its job, there is only one actual starting time relation defined, as shown in (2.24). Since the release dates r_i and the processing times $p_{i,j}$ are assumed to be given as non-negative and strictly positive integers for all $O_{i,j} \in \mathcal{O}, J_i \in \mathcal{J}$, re-

spectively, an explicit non-negativity constraint on the variables involved is not required.

The presented BJSPT is discussed and solved throughout this thesis taking the following assumptions.

Assumptions for the BJSPT

1. There exist finite numbers of jobs n and machines m.
2. Every job $J_i \in \mathcal{J}$ can only be processed on one machine at a time.
3. Every machine $M_k \in \mathcal{M}$ can only process one job at a time.
4. Every job $J_i \in \mathcal{J}$ has a predefined processing sequence and technological route.
5. The preemption of operations is not allowed.
6. The release dates r_i and the due dates d_i are deterministic and given in advance as non-negative integers for all $J_i \in \mathcal{J}$.
7. The processing times $p_{i,j}$ are deterministic and given in advance as strictly positive integers for all $O_{i,j} \in \mathcal{O}$.
8. No two consecutive operations $O_{i,j}$ and $O_{i,j+1}$ belonging to the same job J_i require the same machine.
9. There exist no intermediate buffers for jobs to be stored in between their processing sequence.

2.4 Aspects of Feasibility

The occurrence of blocking constraints in job shop scheduling necessitates a detailed consideration of the properties that define a schedule to be feasible. While for job shop problems without blocking constraints the requirement of an acyclic disjunctive graph representation is sufficient to determine a feasible schedule, this is not directly applicable for the BJSPT. Consider the feasible schedule given in Figure 2.17 for the general instance GJSP1 introduced in Section 2.3.1. This schedule is feasible for the job shop problem without blocking constraints, since the corresponding disjunctive graph shown in Figure 2.18 does not contain any cycle. Consid-

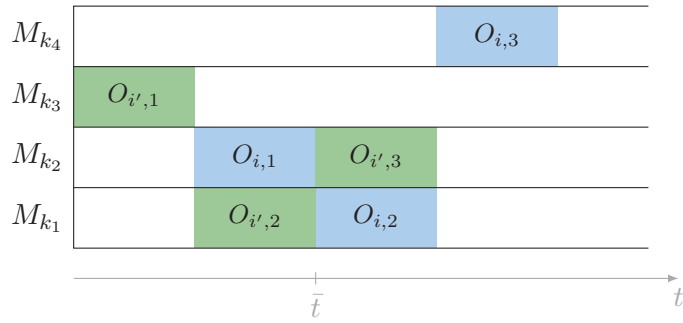


Figure 2.17: Gantt chart of a feasible schedule for the instance GJSP1 including a swap

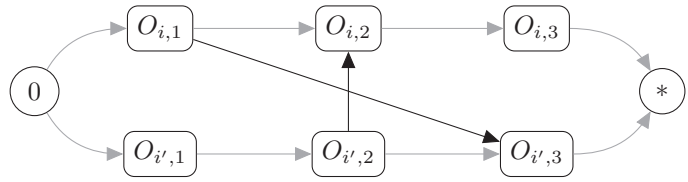


Figure 2.18: Disjunctive graph representation of a feasible schedule for the instance GJSP1 including a swap

ering the problem as a BJSPT, the following blocking constraints need to be fulfilled simultaneously by the starting times of the operations.

$$\begin{aligned} s_{i,2} &\geq s_{i',3} \quad \text{related to } O_{i',2} \to O_{i,2} \text{ on machine } M_{k_1} \\ s_{i',3} &\geq s_{i,2} \quad \text{related to } O_{i,1} \to O_{i',3} \text{ on machine } M_{k_2} \end{aligned}$$

These restrictions can only be met by the equality $s_{i,2} = s_{i',3}$ as indicated in the Gantt chart at point $t = \bar{t}$ (Figure 2.17). This means that the processing of the operations $O_{i,2}$ and $O_{i',3}$ starts at the same point in time and the jobs J_i and $J_{i'}$ simultaneously change the machines. Such a situation is called a *swap* of operations or jobs on machines, cf. among others [32], [83], [88] and [98]. Depending on the underlying practical or theoretical context, swaps are defined to be feasible or infeasible for the considered optimization problem. While the simultaneous movement of jobs is often regarded as feasible in production planning scenarios and in a theoretic point of view, cf. for instance [29], [34], [64] and [98], swaps may refer to collisions and are strictly infeasible in train scheduling or transport

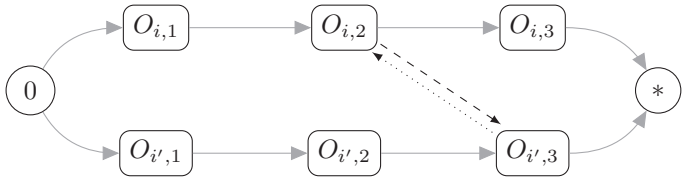


Figure 2.19: Alternative graph representation of a feasible schedule for the instance GJSP1 including a swap

applications, cf. for instance [46], [83], [84] and [99]. When being treated as infeasible, swaps are also called deadlocks or conflicts, which need to be resolved, cf. [46], [83], [84], [89], [99] and [119].

In this thesis, swaps are defined to be feasible for the BJSPT. Thus, the situation given in Figure 2.17 is considered realizable and the alternative graph shown in Figure 2.19 depicts a feasible selection of alternative arcs, even if this graph is not acyclic. Evidently, the equivalence of the feasibility of the schedule and the non-existence of cycles in the corresponding graph representation does not hold for the alternative graph and the BJSPT. In particular, the following example shows that there may simultaneously exist feasible and infeasible cycles in an alternative graph, which correspond to swaps and actual infeasibilities. Therefore, an additional property is required to conclude from an alternative graph representation that a schedule is feasible for the BJSPT.

Consider the feasible schedule for the general instance GJSP2 introduced in Section 2.3.1 and shown in Figure 2.20, which is already classified to be infeasible with regard to blocking constraints in Section 2.3.2. Assuming that the given operation sequences on the machines constitute a schedule for the BJSPT, the selected alternative arcs are depicted in Figure 2.21. As an example, regard the operation sequence $O_{3,1} \rightarrow O_{4,2} \rightarrow O_{1,1}$ on machine M_1 indicated by the dotted arcs. The decision on the processing of the operation $O_{3,1}$ prior to the operations $O_{4,2}$ and $O_{1,1}$ induces the blocking constraints $s_{3,2} \geq s_{4,2}$ and $s_{3,2} \geq s_{1,1}$, which are implemented by the arcs $(O_{3,2}, O_{4,2})$ and $(O_{3,2}, O_{1,1})$ in the alternative graph. Equivalently, the arc $(O_{4,3}, O_{1,1})$ determines the ordering $O_{4,2} \rightarrow O_{1,1}$ on this machine.

Observing the set of selected arcs in more detail, there exist three cycles in the alternative graph in Figure 2.21. Two of these cycles consist of two

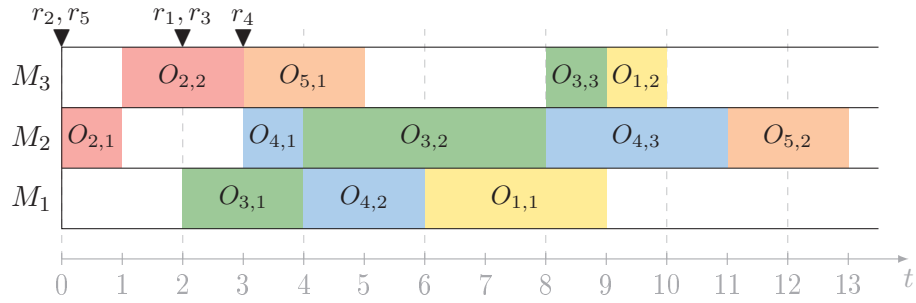


Figure 2.20: Gantt chart of the feasible schedule for the instance GJSP2 without blocking constraints

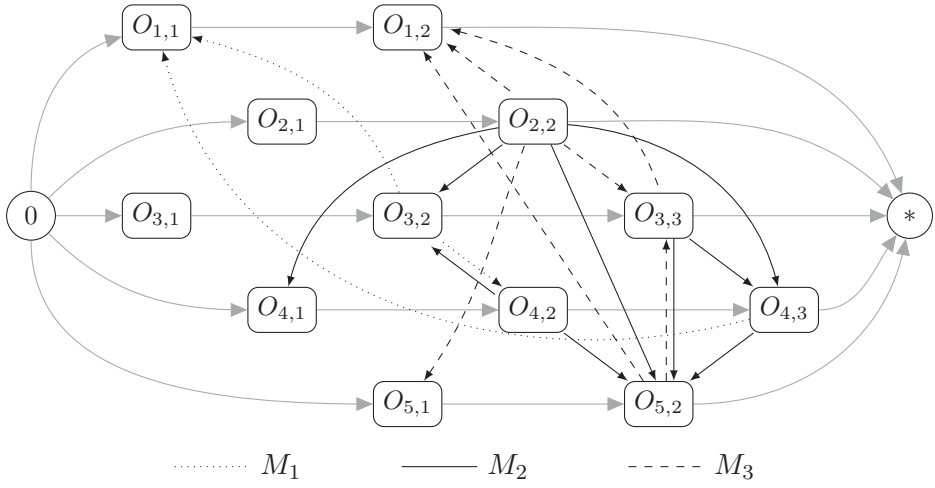


Figure 2.21: Alternative graph representation of a schedule for the instance GJSP2 of the BJSPT

arcs, while the third cycle involves three arcs. Figure 2.22 illustrates the cycles between the operations $O_{3,2}$ and $O_{4,2}$ as well as the operations $O_{3,3}$ and $O_{5,2}$ in part (a) on the left as feasible cycles, since they correspond to swaps of the jobs J_3 and J_4 on the machines M_1 and M_2 as well as the jobs J_3 and J_5 on the machines M_2 and M_3. Note that each of the arcs involved in the same cycle displays a different pattern, which refers to the fact that each operation involved in a swap requires an individual machine. The swap of the jobs J_3 and J_4 on the machines M_1 and M_2 can be observed at point $t = 4$ in the Gantt chart in Figure 2.20, while the other swap of the jobs J_3 and J_5 is not implemented. This is due to the fact that the operations $O_{3,3}$ and $O_{5,2}$ are also involved in a cycle of length three highlighted in part (b) on the right of Figure 2.22. This cycle corresponds to the infeasible operation sequences $O_{5,1} \rightarrow O_{3,3}$ on machine M_2 and $O_{3,2} \rightarrow O_{4,3} \rightarrow O_{5,2}$ on machine M_3, see Section 2.3.2. It contains two solid arcs indicating two operations $O_{4,3}$ and $O_{5,2}$ requiring the same machine, which are supposed to fulfill the precedence relations $O_{4,3} \rightarrow O_{5,2}$ and $O_{5,2} \rightarrow O_{4,3}$, simultaneously. Since this can never be realized by any pair of starting times, an observable criterion conditioning the feasibility of a schedule for the BJSPT can be derived. Note that the following Proposition 2.1 constitutes a reformulation of the definition of feasible schedules given in [32] and [87], which is based on a weighted alternative graph involving the processing times of the operations as arc weights.

Proposition 2.1. *A schedule is feasible for the BJSPT if and only if the corresponding alternative graph does not contain a cycle that involves operations requiring the same machine.*

Proof. First, let a feasible schedule be given by particular operation sequences for all machines $M_k \in \mathcal{M}$, where the earliest starting times of all operations $O_{i,j} \in \mathcal{O}$ fulfill the following restrictions based on the processing sequences, Inequality (2.21), the consecutive processing on a machine, Inequality (2.22), and blocking, Inequalities (2.23) and (2.24).

(1) Processing sequence implementation: $s_{i,j+1} \geq s_{i,j} + p_{i,j}$ for $O_{i,j}$ and $O_{i,j+1}$ on different machines

(2) Disjunctive processing constraints: $s_{i',j'} \geq s_{i,j} + p_{i,j}$ for $O_{i,j} \rightarrow O_{i',j'}$ on machine M_k

(3) Blocking constraints: $s_{i',j'} \geq s_{i,j+1}$ for $O_{i,j} \rightarrow O_{i',j'}$ on machine M_k

Since the processing times of all operations $O_{i,j} \in \mathcal{O}$ are assumed to be strictly positive, see assumption 7 in Section 2.3.3, $s_{i,j+1} > s_{i,j}$ holds for all pairs of consecutive operations of the same job J_i in (1), and (2) can be reformulated to $s_{i',j'} > s_{i,j}$ for $O_{i,j} \rightarrow O_{i',j'}$ on machine M_k. Consequently, the alternative graph incorporates arcs $(O_{i,j}, O_{i,j+1})$ and $(O_{i,j}, O_{i',j'})$ for all pairs of operations, for which a strict inequality relation of the starting times holds. These arcs can never create cycles in the graph, since there exist non-negative integers as starting times in the feasible schedule fulfilling the strict inequalities.

In contrast, the blocking constraints given in (3) might be set up in a cyclic structure, which is satisfied by equal starting times $s_{i,j+1} = s_{i',j'}$ of all involved operations in the feasible schedule. This constitutes the only possibility of the occurrence of cycles in the alternative graph. Since assumption 8 given in Section 2.3.3 holds and blocking constraints are implemented by arcs of the type $(O_{i,j+1}, O_{i',j'})$ for a pair of operations $O_{i,j} \rightarrow O_{i',j'}$ on machine M_k, these arcs do exclusively connect operations requiring different machines. Thus, cycles involving operations requiring the same machine can never occur in the alternative graph representing a feasible schedule.

Second, let the schedule-representing alternative graph contain no cycles that involve operations requiring the same machine. The starting time relations of all operations are defined by the incorporated arcs of the following three types:

(1) $(O_{i,j}, O_{i,j+1})$ connecting two consecutive operations of the same job on different machines (see assumption 8 in Section 2.3.3),

(2) $(O_{i,j}, O_{i',j'})$ connecting two operations of different jobs on the same machine and

(3) $(O_{i,j}, O_{i',j'})$ connecting two operations of different jobs requiring different machines.

The arcs given in (1) represent the processing sequence of a job and do only include operations of the same job with strictly increasing index j. Thus, these arcs are translated to the schedule as starting times of consecutive operations of the same job fulfilling $s_{i,j+1} \geq s_{i,j} + p_{i,j}$. The arcs described in (2) are transformed into starting times of the operation $O_{i,j}$ and $O_{i',j'}$ fulfilling the restriction of consecutive processing on a given machine $s_{i',j'} \geq s_{i,j} + p_{i,j}$. Regarding arcs of type (3), two situations need to be observed. If an arc of this type is not involved in a cycle, it can implemented as a starting time relation $s_{i',j'} \geq s_{i,j}$, which characterizes a feasible solution of the BJSPT for $O_{i,j-1}$ preceding $O_{i',j'}$ on a particular machine. Considering the case where a cycle exclusively formed by arcs described in (3) exists, the resulting inequalities restricting the starting times of the involved operations can be fulfilled simultaneously by setting $s_{i',j'} = s_{i,j}$ for all operations. This corresponds to a feasible swap, since every operation is processed on a different machine.

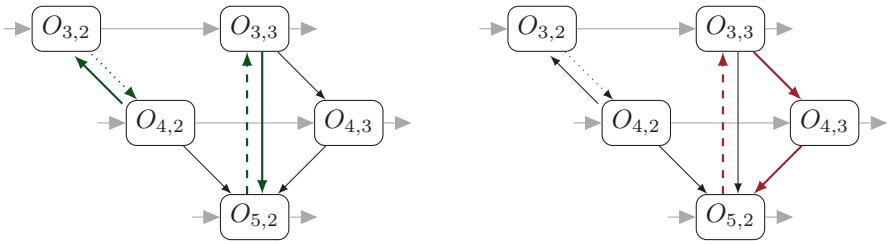
The only remaining potential property of the schedule-representing alternative graph is a cycle jointly formed by arcs of type (1) and (3). Such a cycle cannot exist due to the definition of the basic components of the graph. An arc of type (3) can only be incorporated as a choice out of a pair of alternative arcs representing a disjunctive blocking constraint. Thus, the existence of the arc $(O_{i,j}, O_{i',j'})$ in the graph indicates that the predecessor of operation $O_{i,j}$ is processed prior to the operation $O_{i',j'}$ on a particular machine. By implication, a cycle jointly including arcs of type (1) and type (3) incorporates at a specific point an arc of type (1) like $(O_{i,j-1}, O_{i,j})$ followed by an arc of type (3) such as $(O_{i,j}, O_{i',j'})$, which corresponds to involving two operations $O_{i,j-1}$ and $O_{i',j'}$ requiring the same machine. □

Infeasible cycles like the one given in Figure 2.22 (b) do only occur through the combination of arcs implementing different types of constraints. While the arcs $(O_{5,2}, O_{3,3})$ and $(O_{3,3}, O_{4,3})$ connect operations of different machines and induce the blocking constraints $s_{5,2} \geq s_{3,3}$ and $s_{3,3} \geq s_{4,3}$, the arc $(O_{4,3}, O_{5,2})$ represents a partial operation sequence on machine M_2 requiring $s_{5,2} \geq s_{4,3} + p_{4,3}$. Since the processing times of all operations are assumed to be positive, $s_{5,2} > s_{4,3}$ needs to be true. For this reason,

the red cycle in Figure 2.22 does not refer to a feasible swap implying equal starting times for all involved operations. Precisely considering the illustration, it is easy to observe potential resolutions. In order to obtain a feasible schedule for the BJSPT, one of three involved arcs needs to be reversed or substituted by its alternative opponent so that the cycle containing two operations requiring the same machine is destructed. In Section 2.3.2, a feasible schedule is derived by setting the operation sequence on machine M_3 to $O_{2,2} \rightarrow O_{3,3} \rightarrow O_{5,1} \rightarrow O_{1,2}$. While all other arcs remain unchanged, the arc $(O_{5,2}, O_{3,3})$ is excluded and replaced by its opponent of the pair of alternative arcs $(O_{3,3}, O_{5,1})$, as graphed in Figure 2.16. Another potential resolution strategy is the replacement of the arc $(O_{3,3}, O_{4,3})$ by its alternative arc opponent, which is the arc $(O_{4,3}, O_{3,2})$. Unfortunately, this change destructs the infeasible cycle given in Figure 2.22 (b) but constructs another infeasible cycle containing the operations $O_{3,2}$, $O_{4,3}$ and $O_{4,2}$. Thus, replacing the arc $(O_{3,3}, O_{4,3})$ does not represent the most efficient resolution strategy, since it may require further significant changes in the operation sequences. A third potential resolution of the given infeasibility is the reversal of the arc $(O_{4,3}, O_{5,2})$ also induced by the pair of alternative arcs. This corresponds to a change in the operation sequence on machine M_2, which does not create an infeasible cycle including other operations. The feasible schedule implementing the resulting operation sequence $O_{2,1} \rightarrow O_{4,1} \rightarrow O_{3,2} \rightarrow O_{5,2} \rightarrow O_{4,3}$ on machine M_2 is shown in Figure 2.23. Note that this schedule features both swaps indicated by the cycles given in Figure 2.22 (a) at points $t = 4$ and $t = 8$.

2.5 Instances of the Blocking Job Shop Scheduling Problem with Total Tardiness Minimization

The following section introduces the instances of the BJSPT modeled and solved in this thesis. As indicated in Section 2.1, there exist two main areas from which the motivation of considering job shop problems with blocking constraints arises. Besides, the generalizations and extensions of the considered problem explained in the Sections 2.3.1 and 2.3.2 induce specific structures and properties, whose impact shall be discussed in modeling and solving the BJSPT. Therefore, the intention is to create a set of instances



(a) Feasible cycles **(b)** Infeasible cycle

Figure 2.22: Analyzing cycles in the alternative graph representing a schedule for the instance GJSP2 of the BJSPT

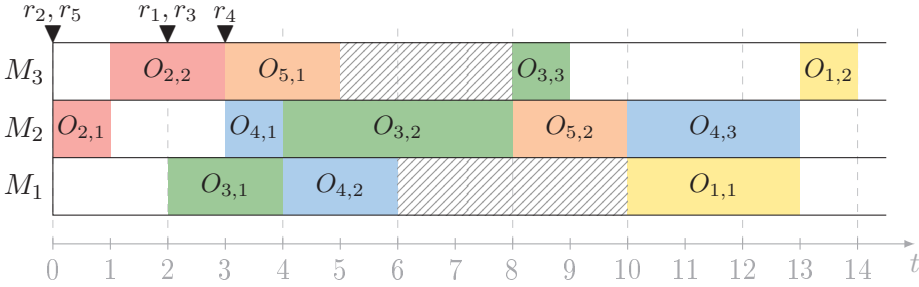


Figure 2.23: Gantt chart of the resolved feasible schedule for the instance GJSP2 of the BJSPT

that represents a production planning as well as a train scheduling environment. It shall incorporate highly and sparsely structured instances with more and less randomness. Section 2.5.1 describes benchmark instances of standard job shop scheduling problems given in the literature together with their properties and necessary adaptations. Regarding train scheduling problems, there exist no established benchmark instances to the best of the author's knowledge. Therefore, a set of 15 instances is generated based on train scheduling structures, cf. [77] and [78], and introduced in Section 2.5.2. The complete set of instances discussed throughout this thesis is precisely characterized by evaluating different property measures in Section 2.5.3.

2.5.1 Introducing and Adapting Established Job Shop Scheduling Instances from the Literature

A key element of the comparison of different solution methods proposed by individual researchers for a particular optimization problem is the application of the techniques to the same set of benchmark instances, cf. Taillard [117]. The generation and establishment of such instances covers several issues. The problems are supposed to be challenging but approachable and of a certain practical relevance, cf. [117]. Instances shall be diverse in their properties but easy to describe and to reproduce. Regarding the standard job shop scheduling problem $J \mid\mid C_{max}$ with $n_i = m$ for all $J_i \in \mathcal{J}$, which is described in Section 2.2, there exist five well-established sets of benchmark instances mainly accessible in Beasleys OR Library [14].

Fisher and Thompson [55] introduce the first set of benchmark instances in 1963. Three of these instances are available in the library, among them is the famous $(10, 10)$-instance that remained unsolved for decades. Lawrence [80] proposes a set of forty instances classifiable by size in groups of five including 5 to 15 machines and 10 to 30 jobs. In [2], Adams, Balas and Zawack test their solution method on some of the aforementioned instances and nine newly generated ones with 10 to 15 machines and 10 to 40 jobs featuring different ranges of processing times. For all of these instance sets, the technological routes of the jobs as well as the processing times of the operations are randomly generated based on uniform distributions.

In contrast, Applegate and Cook [7] provide ten $(10, 10)$-instances, where the technological routes are created by humans following the intention to construct problems that are hard to solve.

These four sets of benchmark instances are widely used by researches to compare solution approaches for the standard job shop scheduling problem, cf. for instance [1], [25], [27], [96], [97], [105], [118], [120] and [122]. Consequently, the optimal or satisfactory solutions are known for most of these instances as reported by Taillard in [117]. For this reason, the author proposes 80 larger instances involving 15 to 20 machines and 15 to 100 jobs, which are evaluated to be hardest among hundreds of randomly generated problems. This constitutes the fifth set of established benchmark instances for the standard job shop scheduling problem, cf. [13].

Considering the BJSPT, there are no specifically tailored benchmark instances existing in the literature, cf. [88]. Since the integration of blocking constraints and the evaluation of the total tardiness do not require a specific structure of the input data, the benchmark instances designed for the standard job shop problem are commonly extended and used in testing solution methods for job shop scheduling problems with blocking constraints, cf. [29], [34], [64], [88], [98] and [103], as well as for job shop scheduling problems with tardiness-based objectives, cf. [34] and [76]. In this thesis, the instances proposed by Lawrence in [80] are mathematically described and solved. This instance set provides a variety of different problem sizes and structures. With regard to the minimization of the makespan, some instances are reported to be easy to solve while others require more computational effort, cf. [7] and [120]. The entire set of instances is available in Beasleys OR Library [14]. Table 2.1 summarizes the eight groups of instances with the corresponding instance size (n, m), whereby the labels are applied in accordance to the literature. Since the instances are created while implementing the standard variant of the job shop problem, the total number of operation is determined by $n_{op} = n \cdot m$ for all instances.

As indicated above, some extensions in the input data are necessary to treat the problems given by Lawrence as instances for the BJSPT. While blocking constraints can be integrated without any changes, the calculation of the total tardiness requires predefined due dates d_i for all jobs $J_i \in \mathcal{J}$. Furthermore, release dates $r_i \in \mathbb{Z}_{\geq 0}$ are considered here as a more general

Table 2.1: Summary of the size of the considered instances by Lawrence [80]

Instance	la01 - la05	la06 - la10	la11 - la15	la16 - la20
size (n, m)	$(10, 5)$	$(15, 5)$	$(20, 5)$	$(10, 10)$
Instance	la21 - la25	la26 - la30	la31 - la35	la36 - la40
size (n, m)	$(15, 10)$	$(20, 10)$	$(30, 10)$	$(15, 15)$

problem setting. To force jobs to overlap during their processing, the release dates r_i are randomly chosen from the following interval.

$$r_i \in \left[0 \, , \, 2 \cdot \min_{J_i \in \mathcal{J}} \left\{ \sum_{j=1}^{n_i} p_{ij} \right\} \right] \qquad \text{for all } J_i \in \mathcal{J} \tag{2.25}$$

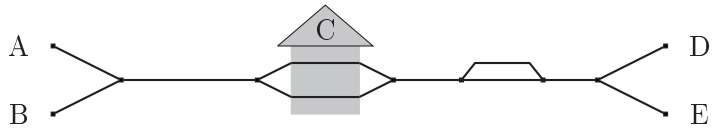
Based on these release times, the due dates d_i are determined such that the allowance $d_i - r_i$ constitutes a linear multiple of the total processing time of the considered job J_i.

$$d_i = \left\lceil r_i + \left(1.2 \cdot \sum_{j=1}^{n_i} p_{ij} \right) \right\rceil \qquad \text{for all } J_i \in \mathcal{J} \tag{2.26}$$

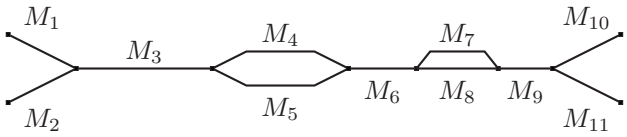
The due date factor is defined to be 1.2, which leads to the generation of tight due dates, cf. [34], [76] and [92]. Thus, the adapted instances are expected to be computationally challenging.

2.5.2 Generating Train Scheduling-Inspired Instances for the BJSPT

As described in the previous section, the benchmark instances used for the BJSPT have properties, which are true for the standard job shop problem, but do not originally incorporate features like general technological routes, release dates and due dates of jobs. Furthermore, the given technological routes and processing times are randomly determined from uniform distributions. Regarding a real-world production plant, it is certainly the case, and for train scheduling problems, it is obvious, that this dimension of



(a) A single-track railway network



(b) Machine interpretation of the railway network

Figure 2.24: Definition and interpretation of a single-track railway network

randomness does not picture production sequences and processing times or train routes and travel times occurring in practice. It can be assumed that there exists a certain amount of structure within the planning situation, so that completely arbitrary technological routes and processing times are not realistic. Moreover, there are indications given in the literature to expect that standard job shop problems with completely random technological routes are easier to solve than instances featuring structured ones, cf. [55] and [114]. Therefore, a set of train scheduling-inspired instances is generated in addition to the extended Lawrence instances explained in Section 2.5.1.

The test instances are set up based on the graph structure given in part (a) at the top of Figure 2.24, cf. [77] and [78]. The illustrated railway network consists of eight bidirectional single tracks, a station C with two parallel tracks and a siding. Trains can only enter and leave the network at the bounding nodes A, B, D and E. Furthermore, they are only allowed to pass and overtake in parallel track sections, namely the station C and the siding. There exist different methods of interpreting the given single-track network as a set of machines. In this thesis, the *Parallel-Machine Approach* is applied, where every single track, either individually or as a part of parallel tracks, constitutes one machine, cf. [77] and [78]. Figure 2.24 (b) shows the transformation of the given railway network into a set of eleven machines.

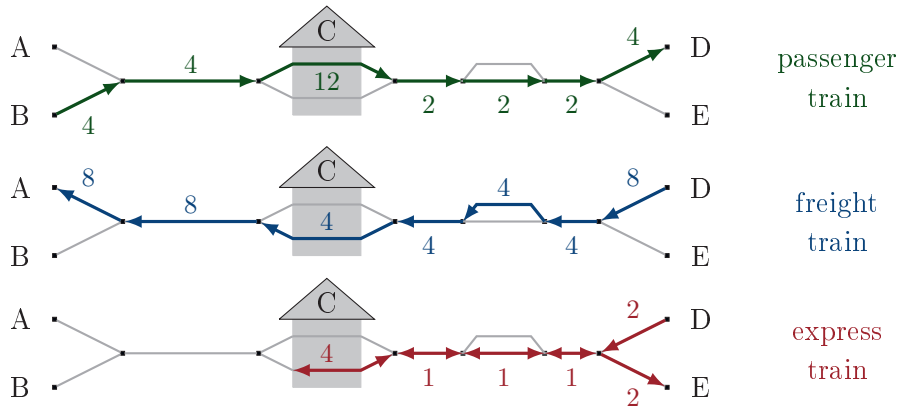


Figure 2.25: Three exemplary train routes and travel times in the single-track railway network

Since the trains are treated as jobs in the resulting job shop scheduling problem, the routes of the trains in the railway network define the technological routes of the jobs in the shop. This means that an operation is described by a train traveling through a track section resembling a job being processed on a machine. It is assumed that the origin and the destination of a train in the network are different from each other and taken from the set of bounding nodes {A, B, D, E}. Thus, there exist twelve distinct origin-destination pairs, one of which is chosen randomly for each train. The specific train route is generated correspondingly either by a shortest path route or by a recirculating route, where the station C acts as a terminus. In Figure 2.25, three origin-destination pairs are exemplarily shown. The green route between B and D as well as the blue route between D and A constitute shortest path routes, whereas the red D-E-route incorporates a revisit of the train at different tracks. Thus, the route between D and E results in a processing sequence of a job involving recirculation.

As the train routes define the technological routes of the jobs, the processing times of the operations are intended to reflect the travel times of trains in a railway network. Three characteristic train types are introduced, namely passenger train, express train and freight train, with corresponding travel times in the different parts of the network. The main differences, such as speed and expected passenger transfer time in stations,

Table 2.2: Summary of the labels and the size of the generated train scheduling-inspired instances

Instance	ts01 - ts05	ts06 - ts10	ts11 - ts15
size (n, m)	$(10, 11)$	$(15, 11)$	$(20, 11)$

are thereby considered. In Figure 2.25, the green route illustrated at the top is traveled by a passenger train, the blue route in the middle belongs to a freight train and an express train operates on the red recirculation route given at the bottom. The corresponding travel times are indicated by the colored numbers next to the track segments. A train type is randomly chosen for each train and the processing times of the operations of the represented job are determined accordingly. It is assured that every train type appears at least once in every instance. Based on the processing times of the operations, the release dates and the due dates of the jobs are generated following the rules given in (2.25) and (2.26) in Section 2.5.1. As mentioned above, this induces a temporal overlapping of the trains traveling through the network and creates tight desired arrival times.

For an appropriate testing of the modeling and solution approaches introduced in the remainder of this thesis, 15 train scheduling-inspired instances involving 10, 15 and 20 trains, respectively, are constructed. Table 2.2 presents the labels, by which the instances are marked, and the corresponding sizes (n, m).

As an example, the input data of the instance ts01 is summarized below. The first two columns state the train type and the corresponding origin-destination pair. The subsequent columns give the job shop scheduling interpretation including the job J_i, its release date r_i, due date d_i and the resulting technological route TR_i.

Instance ts01

Train type	**Route**	**Job**	r_i	d_i	**Technological route** TR_i
Express Tr.	D $\rightarrow$ E	J_1	13	30	$M_{10} \rightarrow M_9 \rightarrow M_8 \rightarrow M_6 \rightarrow M_5 \rightarrow M_6 \rightarrow M_8 \rightarrow M_9 \rightarrow M_{11}$
Passenger Tr.	A $\rightarrow$ E	J_2	2	38	$M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow M_6 \rightarrow M_8 \rightarrow M_9 \rightarrow M_{11}$
Passenger Tr.	A $\rightarrow$ B	J_3	8	37	$M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow M_3 \rightarrow M_2$
Passenger Tr.	A $\rightarrow$ E	J_4	5	41	$M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow M_6 \rightarrow M_8 \rightarrow M_9 \rightarrow M_{11}$
Passenger Tr.	E $\rightarrow$ B	J_5	4	40	$M_{11} \rightarrow M_9 \rightarrow M_7 \rightarrow M_6 \rightarrow M_4 \rightarrow M_3 \rightarrow M_2$
Passenger Tr.	D $\rightarrow$ A	J_6	6	41	$M_{10} \rightarrow M_9 \rightarrow M_7 \rightarrow M_6 \rightarrow M_5 \rightarrow M_3 \rightarrow M_1$
Passenger Tr.	E $\rightarrow$ B	J_7	11	47	$M_{11} \rightarrow M_9 \rightarrow M_7 \rightarrow M_6 \rightarrow M_4 \rightarrow M_3 \rightarrow M_2$
Passenger Tr.	E $\rightarrow$ B	J_8	2	38	$M_{11} \rightarrow M_9 \rightarrow M_7 \rightarrow M_6 \rightarrow M_4 \rightarrow M_3 \rightarrow M_2$
Express Tr.	B $\rightarrow$ D	J_9	8	26	$M_2 \rightarrow M_3 \rightarrow M_5 \rightarrow M_6 \rightarrow M_8 \rightarrow M_9 \rightarrow M_{10}$
Freight Tr.	E $\rightarrow$ A	J_{10}	5	53	$M_{11} \rightarrow M_9 \rightarrow M_8 \rightarrow M_6 \rightarrow M_4 \rightarrow M_3 \rightarrow M_1$

2.5.3 Properties of the Blocking Job Shop Scheduling Instances

In the literature, there are various resolution methods for job shop scheduling problems with and without blocking constraints featuring different objective functions. Nonetheless, the reasons for solution approaches to succeed or fail on particular instances of the same problem are barely understood, cf. [20], [91], [109] and [126]. Most attempts to experimentally answer this question rely on a search space analysis, which is a general approach but significantly based on the applied heuristic, cf. [20], [91], [109] and [126].

In this thesis, a more generic and property-focused evaluation of the instances is done to discuss a potential relationship between instance char-

acteristics and required computational effort. Such observations might be useful in designing decision support systems that incorporate the choice of the most promising algorithm. In [108], Smith-Miles et al. study the relationship between problem characteristics, such as the number of jobs, the mean processing time, the tardiness factor (here: due date factor) and the due date range, and the performance of two priority rules for a single-machine early/tardy scheduling problem. It is stated that the tardiness factor and the due date range correlate with the heuristic performance. Furthermore, the authors argue that the choice of the considered properties is a significant task, since they need to be of moderate computational effort but high differentiation power according to the applied algorithms. Mattfeld and Bierwirth [92] use instance properties, namely the mean allowance rate of the jobs (here: due date factor), the machine utilization rate and the standard deviation of the machine utilization, to demonstrate the heterogeneity of the instance set of the considered job shop problem without blocking constraints. Unfortunately, the authors do not discuss the computational results in view of these characteristics. The following basic properties are used to characterize the problems tackled in this thesis.

- the number of jobs n
- the number of machines m
- the job-machine-ratio $\frac{n}{m}$
- the number of operations n_{op}

Additionally, the concepts proposed in [92] are modified to be suitable for the problem under study. In order to display unbalanced workload and predict potential bottleneck issues, the focus of the instance key figures is changed from regarding job data exclusively to a machine-based perspective. First, an operation release time $r_{i,j}$ and an operation due time $d_{i,j}$ are defined for every operation $O_{i,j} \in \mathcal{O}$ as follows:

$$r_{i,1} = r_i \qquad \text{for } J_i \in \mathcal{J} \tag{2.27}$$

$$r_{i,j} = r_i + \sum_{j'=1}^{j-1} p_{i,j'} \qquad \text{for } O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,1}\},\ J_i \in \mathcal{J} \tag{2.28}$$

$$d_{i,n_i} = d_i \qquad \text{for } J_i \in \mathcal{J} \tag{2.29}$$

$$d_{i,j} = d_i - \sum_{j'=j+1}^{n_i} p_{i,j'} \qquad \text{for } O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,n_i}\},\ J_i \in \mathcal{J} \tag{2.30}$$

This corresponds to the earliest possible starting time of an operation as well as the latest possible completion time of an operation for the job to be completed without tardiness. Thereafter, the *inter-arrival time* λ_k of the operations on each machine $M_k \in \mathcal{M}$ is introduced by

$$\lambda_k = \frac{\max\left\{r_{i,j} \mid O_{i,j} \in \Omega^k\right\} - \min\left\{r_{i,j} \mid O_{i,j} \in \Omega^k\right\}}{\mid \Omega^k \mid}. \tag{2.31}$$

Accordingly, a *machine utilization rate* u_k is proposed and defined for each $M_k \in \mathcal{M}$ as

$$u_k = \frac{\frac{\sum_{O_{i,j} \in \Omega^k} p_{i,j}}{|\Omega^k|}}{\lambda_k} = \frac{\bar{p}_k}{\lambda_k}. \tag{2.32}$$

The mean processing time $\bar{p}_k$ of operations on the considered machine M_k is divided by the inter-arrival time λ_k. Thus, the machine usage rate will increase if either the mean processing time of the operations increases or the inter-arrival time of the operations decreases, while the corresponding other quantity is held constant.

Since the due dates are generated by applying a constant due date factor of 1.2 for all instances, the declaration of this multiplier as an instance property, like it is done in [92] and [108], is meaningless. However, due dates are expected to affect the difficulty of an instance and the observation of a due date-related characteristic seems reasonable. Therefore, the concept of slack time, cf. [8] and [123], is modified to a machine-based variant as well. The *machine slack* l_k is introduced as follows:

$$l_k = \frac{\max\left\{d_{i,j} \mid O_{i,j} \in \Omega^k\right\} - \min\left\{r_{i,j} \mid O_{i,j} \in \Omega^k\right\}}{\sum_{O_{i,j} \in \Omega^k} p_{i,j}} \tag{2.33}$$

An instance is expected to be challenging if the utilization rate of the machines is generally high, if the slack of the machines is generally low and if the workload is significantly unbalanced between the machines. As to observe and discuss potential effects, the following instance properties are evaluated.

Table 2.3: Instance properties of the train scheduling-inspired instances ts01 - ts15

Inst.	n	m	$\frac{n}{m}$	n_{op}	$\bar{u}$	$std(u_k)$	$\bar{l}$	$std(l_k)$
ts01	10	11	0.909	70	1.534	1.389	1.552	0.649
ts02	10	11	0.909	68	1.291	1.532	2.039	1.194
ts03	10	11	0.909	70	1.619	2.492	2.061	1.213
ts04	10	11	0.909	72	1.123	0.771	1.825	0.805
ts05	10	11	0.909	68	1.220	0.920	1.908	1.191
ts06	15	11	1.364	105	0.873	0.400	2.023	1.049
ts07	15	11	1.364	107	1.282	0.737	1.517	0.815
ts08	15	11	1.364	105	1.193	0.590	1.601	0.996
ts09	15	11	1.364	107	1.080	0.561	1.654	0.786
ts10	15	11	1.364	103	0.931	0.441	1.839	0.955
ts11	20	11	1.818	138	1.366	0.684	1.226	0.556
ts12	20	11	1.818	140	1.447	0.857	1.278	0.643
ts13	20	11	1.818	142	1.457	0.797	1.300	0.852
ts14	20	11	1.818	140	1.463	0.793	1.178	0.576
ts15	20	11	1.818	142	1.567	1.020	1.242	0.638

- the mean machine utilization rate $\bar{u}$
- the standard deviation of the machine utilization rate $std(u_k)$
- the mean machine slack $\bar{l}$
- the standard deviation of the machine slack $std(l_k)$

The Tables 2.3 and 2.4 display the eight characteristics mentioned above for all considered instances ts01 - ts15, introduced in Section 2.5.2, and la01 - la40, explained in Section 2.5.1. The Lawrence instances are initially designed for the standard job shop scheduling problem, so that $n_{op} = n \cdot m$ holds for all problems. In contrast, the choice of train routes in a network yields $n_{op} < n \cdot m$ for all train scheduling-inspired instances, since the number of operations per job does never reach or exceed the number of machines here. As expected, it can be observed that the Lawrence instances

are more balanced with regard to the workload of the machines than the train scheduling instances. The standard deviation of the utilization rate $std(u_k)$ and the standard deviation of the machine slack $std(l_k)$ are higher on average for the ts-instances. This indicates the existence of bottleneck machines, which show a significantly higher utilization rate u_k and machine slack l_k than others. Comparing problems of the same size (n, m), the instances la21 to la25 seem to be a rather heterogeneous group with respect to their varying mean utilization rate $\bar{u}$ and mean machine slack $\bar{l}$. Furthermore, the instance la17 appears to have a considerably high mean machine slack $\bar{l}$ combined with a high standard deviation $std(l_k)$ compared to the other $(10, 10)$-instances.

Table 2.4: Instance properties of the Lawrence instances la01 - la40

Inst.	n	m	$\frac{n}{m}$	n_{op}	$\bar{u}$	$std(u_k)$	$\bar{l}$	$std(l_k)$
la01	10	5	2.0	50	0.811	0.141	1.487	0.202
la02	10	5	2.0	50	0.644	0.072	1.823	0.207
la03	10	5	2.0	50	0.690	0.168	1.786	0.389
la04	10	5	2.0	50	1.082	0.410	1.227	0.251
la05	10	5	2.0	50	0.719	0.211	1.746	0.553
la06	15	5	3.0	75	1.134	0.246	1.083	0.225
la07	15	5	3.0	75	0.957	0.213	1.235	0.192
la08	15	5	3.0	75	0.994	0.090	1.139	0.110
la09	15	5	3.0	75	1.125	0.220	1.061	0.211
la10	15	5	3.0	75	0.955	0.180	1.219	0.218
la11	20	5	4.0	100	1.271	0.155	0.877	0.097
la12	20	5	4.0	100	1.261	0.227	0.924	0.182
la13	20	5	4.0	100	1.513	0.469	0.834	0.227
la14	20	5	4.0	100	1.242	0.177	0.902	0.122
la15	20	5	4.0	100	1.355	0.175	0.847	0.098
la16	10	10	1.0	100	0.414	0.059	2.841	0.374
la17	10	10	1.0	100	0.374	0.104	3.227	1.026
la18	10	10	1.0	100	0.412	0.103	2.910	0.739

continued on the next page

Inst.	n	m	$\frac{n}{m}$	n_{op}	$\bar{u}$	$std(u_k)$	$\bar{l}$	$std(l_k)$
la19	10	10	1.0	100	0.425	0.075	2.743	0.517
la20	10	10	1.0	100	0.423	0.081	2.717	0.470
la21	15	10	1.5	150	0.650	0.114	1.772	0.313
la22	15	10	1.5	150	0.601	0.090	1.877	0.293
la23	15	10	1.5	150	0.878	0.174	1.403	0.263
la24	15	10	1.5	150	0.514	0.056	2.175	0.251
la25	15	10	1.5	150	0.519	0.062	2.154	0.226
la26	20	10	2.0	200	0.902	0.210	1.323	0.252
la27	20	10	2.0	200	0.768	0.074	1.473	0.140
la28	20	10	2.0	200	0.617	0.061	1.789	0.181
la29	20	10	2.0	200	0.664	0.058	1.679	0.138
la30	20	10	2.0	200	0.778	0.116	1.490	0.181
la31	30	10	3.0	300	0.900	0.089	1.227	0.114
la32	30	10	3.0	300	1.014	0.102	1.110	0.107
la33	30	10	3.0	300	0.949	0.097	1.161	0.110
la34	30	10	3.0	300	0.979	0.105	1.141	0.131
la35	30	10	3.0	300	0.994	0.160	1.148	0.153
la36	15	15	1.0	225	0.415	0.075	2.720	0.429
la37	15	15	1.0	225	0.470	0.080	2.450	0.381
la38	15	15	1.0	225	0.424	0.074	2.728	0.539
la39	15	15	1.0	225	0.506	0.073	2.253	0.351
la40	15	15	1.0	225	0.440	0.081	2.594	0.486

3 Mathematical Formulations for Blocking Job Shop Scheduling Problems

This chapter focuses on the exact solvability of different mathematical formulations for the BJSPT when applying standard MIP techniques. Some of the main aspects discussed in the following sections are initially published in [77] and in [78] as a joint work with Frank Werner. At first, Section 3.1 describes three well-known types of sequence-defining variables and their characteristics. It is shown that two of these types of unknowns are based on the disjunctive graph model, while the third variable type corresponds to a flow representation of a schedule. Consequently, the disjunctive graph-based variables are chosen for the extensive study regarding the potential influence of the optimization model on the required computa-

tional effort of an exact solution method. The corresponding mathematical programming formulations of the BJSPT are presented in Sections 3.2 and 3.3. For more general descriptions on mathematical programs for generic job shop scheduling problems, the reader is referred to [23], [101] and [102] among others. As a structural aspect of the considered models that plays an important role for exact MIP techniques, the definition of lower bounds on the objective function value is reviewed in Section 3.4. Finally, comprehensive computational experiments, which include the results of a state-of-the-art MIP solver when applying both mathematical formulations on the set of benchmark instances, are performed and analyzed in Section 3.5.

3.1 Types and Characteristics of Sequence-Defining Variables

In 1997, Pan [101] summarized different types of sequence-defining variables appropriate for job shop scheduling problems, their properties and resulting mathematical formulations. However, it is not obvious, whether these kinds of variables are suitable for modeling and solving the blocking job shop scheduling problem by means of exact mathematical programming. In Section 3.1.1, three well-known types of binary variables applicable to implement logical implications in integer or mixed-integer programs are described, namely *precedence variables*, *order-position variables* and *time-indexed variables*, cf. [23], [78], [101] and [102]. For the purpose of indicating the size of the resulting mathematical formulations, the subsequent Section 3.1.2 contains approximations of the maximum number of unknowns required to entirely model the BJSPT. With regard to the implementation of sequencing decisions in the mathematical program, some important structural differences between the sequence-defining variables are highlighted in Section 3.1.3. Arguments are given for the choice of the variables included in the computational study.

3.1.1 Introducing Sequence-Defining Variables

Precedence variables of the form $y_{i,i'}$ are first applied by Manne [86] in modeling a job shop scheduling problem as a mixed-integer program. These binary variables indicate with $y_{i,i'} = 1$ that a job J_i is processed prior to a job $J_{i'}$. The reverse ordering is implemented by $y_{i,i'} = 0$. Variables expressing logical implications in this pattern are called *indicator variables*, cf. [27]. The processing sequence of all jobs is given *semi-implicitly* and can easily be derived from the pairwise precedence relations. Considering the recirculation of jobs, a precedence relation $J_i \rightarrow J_{i'}$ given by $y_{i,i'} = 1$ does not necessarily define an entire and unique sequence of all appearances of J_i and $J_{i'}$. Therefore, precedence variables are defined operation-based to be applied in modeling the BJSPT.

$$y_{i,j,i',j'} = \begin{cases} 1 & \text{if } O_{i,j} \text{ is processed prior to } O_{i',j'}, \\ 0 & \text{if } O_{i',j'} \text{ is processed prior to } O_{i,j}. \end{cases}$$

With regard to the overall goal of solving mixed-integer programs to optimality, it is beneficial to avoid meaningless variables in the formulation. The sequencing decision described above only needs to be made for operations belonging to different jobs that are processed on the same machine. Thus, introducing precedence variables $y_{i,j,i',j'}$ for all non-equivalent pairs of operations $O_{i,j}, O_{i',j'} \in \mathcal{O}$ is not reasonable. The indicator variables are only set up as follows:

$$y_{i,j,i',j'} \in \{0,1\} \qquad \text{for all } O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', M_k \in \mathcal{M}.$$

Order-position variables of the form $x^k_{i,r}$ are introduced by Wagner [124] implementing the assignment of order-positions r to jobs in flow shop and job shop problems. If a job J_i is processed as the r-th job on machine M_k, the binary variable $x^k_{i,r}$ will take the value of 1 and will equal 0 otherwise. Thus, the sequencing of the jobs is done *explicitly* by assigning an order-position r to each job. Since recirculation is involved, the order-position variables are introduced operation-based as $x^k_{i,j,r}$.

By the definition of the job shop problem, every operation $O_{i,j} \in \mathcal{O}$ is to be processed on a predefined machine M_k. Thus, there is exactly one order-position to be assigned to the considered operation $O_{i,j}$ that automatically

refers to this machine. The machine index k of the variables $x^k_{i,j,r}$ can be dropped and the order-position index is restricted to $r = 1, \ldots, R_k$, where R_k denotes the total number of operations on machine M_k. To model the BJSPT with explicit sequencing, binary variables are introduced for each $O_{i,j} \in \mathcal{O}$ as follows:

$$x^r_{i,j} = \begin{cases} 1 & \text{if } O_{i,j} \text{ is processed at order-position } r, \\ 0 & \text{if } O_{i,j} \text{ is not processed at order-position } r. \end{cases}$$

Time-indexed variables of the form $z_{i,k,t}$ are first used in a binary optimization program for job shop and flow shop scheduling problems by Bowman [28]. A variable $z_{i,k,t}$ will be set to 1, if job J_i is processed on machine M_k in time period $t = (t-1, t]$, and will equal 0 otherwise. Thus, the processing sequence of the jobs is defined *implicitly* by these variables. To assure a unique sequencing in case of recirculating jobs, the time-indexed unknowns are defined operation-based in mathematical formulations for the BJSPT. In line with the explanation given above, the machine index k can be dropped, since the required machine is fixed for every operation. Time-indexed variables implementing the processing of all operations $O_{i,j} \in \mathcal{O}$ are defined as follows:

$$z^t_{i,j} = \begin{cases} 1 & \text{if } O_{i,j} \text{ is processed in time period } t, \\ 0 & \text{if } O_{i,j} \text{ is not processed in time period } t. \end{cases}$$

3.1.2 On the Number of Sequence-Defining Variables Required in Mathematical Formulations for the BJSPT

Through binary values, each class of variables implements a specific number of order-oriented decisions that concern operations, machines and time. It is intended to set up mathematical formulations for a BJSPT with n jobs and m machines, where each job J_i consists of n_i operations and each machine M_k needs to process a set of operations Ω^k. The number of variables, the resulting optimization programs are composed of, depend on the chosen type of unknowns as well as on the instance input data. As one measure of the size of a formulation, this quantity indicates the

expected theoretical and computational effort of solving the corresponding optimization problem. Since the presence of blocking constraints does not effect the number of required variables, the following arguments are equivalently true for generic job shop scheduling problems, cf. [101].

Regarding precedence variables, there are two unknowns introduced for every pairwise ordering of operations of different jobs on the same machine. Thus, the actual number of precedence variables $\#\text{var}(y_{i,j,i',j'})$ can be approximated by the quantity of pairs of operations requiring the same machine.

$$\begin{aligned}\#\text{var}(y_{i,j,i',j'}) &\leq \sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid -1 \right) \right] = \sum_{M_k \in \mathcal{M}} \left[\left(\mid \Omega^k \mid \right)^2 - \mid \Omega^k \mid \right] \\ &= \sum_{M_k \in \mathcal{M}} \left(\mid \Omega^k \mid \right)^2 - \sum_{M_k \in \mathcal{M}} \mid \Omega^k \mid = \sum_{M_k \in \mathcal{M}} \left(\mid \Omega^k \mid \right)^2 - n_{op}.\end{aligned}$$

Remark. The maximum number of precedence variables can be reduced to $\sum_{M_k \in \mathcal{M}} \frac{1}{2} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid -1 \right) \right]$ by defining only one unknown $y_{i,j,i',j'}$ for each pair of operations of different jobs requiring the same machine and using the equation $y_{i',j',i,j} = 1 - y_{i,j,i','j}$ as a substitute for the corresponding other variable. Computational experiments give evidence for the equivalence of both modeling approaches with regard to computational effort. State-of-the-art solvers seem to easily detect this pairwise connection of indicator variables.

Order-position variables are set up for each operation $O_{i,j} \in \mathcal{O}$ and every existing position on the required machine. Since an operation is a unique processing of a specific job on a specific machine, $\mathcal{O} = \bigcup_{M_k \in \mathcal{M}} \Omega^k$ holds. By definition, the number of order-positions R_k of a machine M_k is equal to the number of operations being processed on this machine, $R_k = \mid \Omega^k \mid$. Thus, the number of order-position variables $\#\text{var}(x^r_{i,j})$ composing a mathematical formulation of the BJSPT results in

$$\#\text{var}(x^r_{i,j}) = \sum_{M_k \in \mathcal{M}} \left(\mid \Omega^k \mid \cdot R_k \right) = \sum_{M_k \in \mathcal{M}} \left(\mid \Omega^k \mid \right)^2.$$

In contrast to the first two types of unknowns, time-indexed variables do additionally refer to time periods. Let the number of time units necessary

to complete all operations be denoted by T. A time-indexed variable is introduced for each operation $O_{i,j} \in \mathcal{O}$ and each time period $t = 1, \ldots, T$. The feasibility of the resulting optimization program depends on an appropriate a priori choice of T, so that a robust estimation is required. Since $T \leq \max_{J_i \in \mathcal{J}}\{r_i\} + \sum_{O_{i,j} \in \mathcal{O}} p_{i,j}$ constitutes an upper bound for every feasible schedule,cf. [102], the number of time-indexed variables involved in a mathematical formulation can adequately be estimated by

$$\#\mathrm{var}(z_{i,j}^t) = n_{op} \cdot T \;\leq\; n_{op} \cdot \left(\max_{J_i \in \mathcal{J}}\{r_i\} + \sum_{O_{i,j} \in \mathcal{O}} p_{i,j} \right).$$

The distribution of the operations on the machines may heavily vary even for instances with equivalent numbers of jobs and machines. Thus, a general evaluation dependent on the subsets of operations Ω^k $(M_k \in \mathcal{M})$ cannot give a robust impression on the quantity of sequence-defining variables needed to model a generic job shop instance. A detailed examination of the input data of the given problem is substantial.

For a special type of job shop scheduling problems, the given estimates can be generalized based on the numbers of involved jobs and machines. Consider the standard (n, m) job shop problem introduced in Section 2.2, where every job $J_i \in \mathcal{J}$ is processed on every machine $M_k \in \mathcal{M}$ exactly once and all jobs are available at the beginning of the planning period $(r_i = 0, J_i \in \mathcal{J})$. It follows that the number of operations is equivalent for all jobs and equals the number of machines, $n_i = m,\ J_i \in \mathcal{J}$. Furthermore, the number of operations on each machine $|\Omega^k|$ is equal to the number of jobs n and the total number of operations is given by $n_{op} = n \cdot m$. In this case, the maximum quantities of precedence and order-position variables necessary to set up the mathematical formulation for the BJSPT result in

$$\begin{aligned}
\#\mathrm{var}(y_{i,j,i',j'}) &\leq \sum_{M_k \in \mathcal{M}} \Big[\mid \Omega^k \mid \cdot \Big(\mid \Omega^k \mid -1 \Big) \Big] = \sum_{M_k \in \mathcal{M}} [n \cdot (n-1)] \\
&\leq \; \big[n^2 - n \big] \cdot m, \\
\#\mathrm{var}(x_{i,j}^r) &= \sum_{M_k \in \mathcal{M}} \Big(\mid \Omega^k \mid \Big)^2 = \sum_{M_k \in \mathcal{M}} n^2 \;=\; n^2 \cdot m.
\end{aligned}$$

While applying the term given above as an estimate of the number of time-indexed variables, the sum of the processing times of all operations can be

substituted by its lower bound as follows. Since the number of operations equals $n \cdot m$ and $p_{i,j} \geq 1$ holds for all $O_{i,j} \in \mathcal{O}$ by assumption, it follows that $\sum_{O_{i,j} \in \mathcal{O}} p_{i,j} \geq n \cdot m$. The sum of the release times is equal to 0 in the considered standard case. Thus, the minimum number of required time-indexed variables in a mathematical formulation for a standard job shop problem can be estimated by

$$\#\text{var}(z^t_{i,j}) = n_{op} \cdot \sum_{O_{i,j} \in \mathcal{O}} p_{i,j} \geq n^2 \cdot m^2.$$

In general, it can be stated that the number of sequence-defining variables required to compose an optimization program for the BJSPT is polynomially bounded in the input data, particularly in the number of operations per machine $|\Omega^k|$ ($M_k \in \mathcal{M}$) and in the maximum total number of time periods $\max_{J_i \in \mathcal{J}}\{r_i\} + \sum_{O_{i,j} \in \mathcal{O}} p_{i,j}$, or for the standard problem, in the number of jobs n and the number of machines m. Contrasting the presented types of variables, the implementation of precedence variables yields a lower quantity of unknowns in the formulation compared to order-position variables. This effect will become even more remarkable with introducing only one precedence variable per pair of operations of different jobs on the same machine, cf. [86]. Assuming $p_{i,j} \geq 1$ for $O_{i,j} \in \mathcal{O}$, the application of time-indexed variables leads to a considerably larger number of sequence-defining variables compared to the other two types.

3.1.3 Characteristics of different Implementations of Sequencing Decisions

The semi-implicit and explicit implementations of sequencing decisions by precedence and order-position variables refer to the orientation of edges in the alternative graph representation of the BJSPT. Since without loss of generality the arguments given in the following hold for job shop problems with and without blocking constraints, the basic instance GJSP2 is used as an example for reasons of simplicity. Consider the three operations $O_{1,1}$, $O_{3,1}$ and $O_{4,2}$ of the jobs J_1, J_3 and J_4 that require the same machine M_1. Figure 3.1 shows the relevant subsets of nodes and edges of the corresponding disjunctive graph $G = (V, E, A)$, which represent the sequencing decisions associated with these three operations. As given in Figure 2.5,

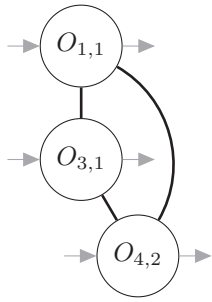


Figure 3.1: Subgraph of the disjunctive graph representation of instance GJSP2 including three operations

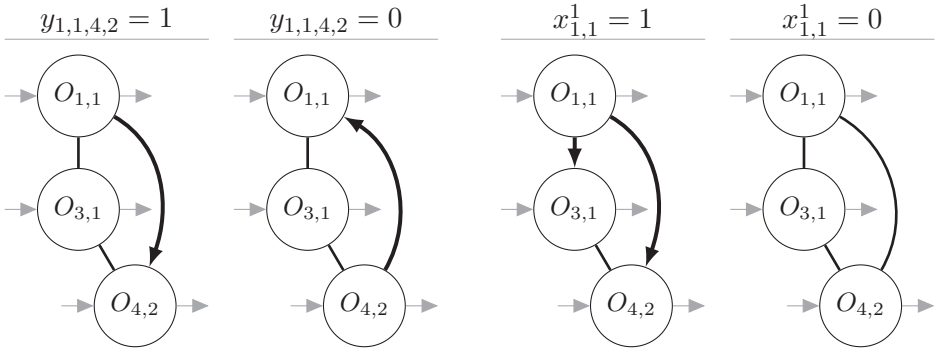


Figure 3.2: Characterizing precedence and order-position variables based on the disjunctive graph

the gray arcs indicate the processing sequences by relating the operations to their predecessor and successor operation of the same job.

Initially, there exist three undirected edges connecting the pairs of operations on machine M_1. During the scheduling process, the sequence of operations on the machine is determined and the edges are replaced by directed arcs. In Figure 3.2, the effect of fixing a sequence-defining variable to either 0 or 1 is exemplarily illustrated for the precedence variable $y_{1,1,4,2}$ on the left and for the order-position variable $x^1_{1,1}$ on the right. The resulting partial sequences on machine M_1 are represented by bold-faced arcs.

The two subgraphs on the left of Figure 3.2 show that the determination of a binary value for a precedence variable implies the orientation of

one edge in the graph. If $y_{1,1,4,2} = 1$ holds, operation $O_{1,1}$ will precede operation $O_{4,2}$ on machine M_1 and the edge $\{O_{1,1}, O_{4,2}\}$ is replaced by the arc $(O_{1,1}, O_{4,2})$ in the subgraph. In contrast, if $y_{1,1,4,2} = 0$ is true, the equality $y_{4,2,1,1} = 1$ follows from the pairwise association of the indicator variables and operation $O_{4,2}$ is processed prior to operation $O_{1,1}$. The edge $\{O_{1,1}, O_{4,2}\}$ is substituted by the arc $(O_{4,2}, O_{1,1})$. Thus, fixing a precedence variable ensures an extension of the partial solution and the effectiveness of the corresponding starting time dependencies.

The subgraphs on the right illustrate two cases of edge orientation determined by the binary value of an order-position variable. Setting $x^1_{1,1} = 1$ implies that operation $O_{1,1}$ is processed on machine M_1 at order-position $r = 1$. This establishes precedence relations between operation $O_{1,1}$ and all other operations on the considered machine, here operations $O_{3,1}$ and $O_{4,2}$. Thus, the edges $\{O_{1,1}, O_{3,1}\}$ and $\{O_{1,1}, O_{4,2}\}$ are replaced by the directed arcs $(O_{1,1}, O_{3,1})$ and $(O_{1,1}, O_{4,2})$ based on the definition of only one specified value. In contrast, the opposite statement $x^1_{1,1} = 0$ implies no precedence relation between operation $O_{1,1}$ and any other operation. Since there is no pairwise association induced by the definition of these variables, many fixings of unknowns do not result in a precise sequencing decision. Thus, it is not clear to which extent the determination of an order-position variable expands a partial solution and specifies starting time relations.

With regard to the disjunctive programming interpretation of the BJSPT, the semi-implicit implementation of sequencing decisions seems to be more effective compared to the explicit one, since the determination of every precedence variable narrows the set of potential starting times of the operations. Considering the order-position variables for all positions $r \in \mathcal{R}^k \setminus \{1\}$, there is always a set of variable fixings required to implement a sequencing decision. Detailed mathematical formulations for the BJSPT involving both types of variables and a comparison of the performance of a generic MIP solution method on these models are presented in Sections 3.2, 3.3 and 3.5, respectively.

In contrast to the two types of unknowns discussed above, time-indexed variables cannot be interpreted based on the disjunctive programming concept. The overall representation of a schedule does not result in the di-

rected graph itself, but in the determination of a flow in a network, cf. [44] and [102]. A time-expanded graph, as exemplarily given in Figure 3.3 for an instance with three machines, is needed to illustrate the meaning of the time-indexed variables $z_{i,j}^t$.

For each machine and each time period $t = 1, \ldots, T$, there are two nodes introduced to mark the start and the end of the discrete unit of time. In Figure 3.3, three time periods t, $t+1$ and $t+2$ are generally indicated. The continuity of time is represented by horizontal arcs connecting points in time on the same machine. The occupancy of a machine by an operation while processing or blocking is implemented as a positive flow on the dashed horizontal arcs of the required machine. Sequencing decisions and possible machine changes of jobs, can only be realized between two consecutive time periods and are represented by black solid arcs. With setting appropriate capacities and weights for all nodes and arcs in the network, a schedule is determined by a flow between the artificial nodes 'source' and 'sink' that are connected to all time-indicating nodes, cf. [44] and [102]. The resulting flow consists of n disjunctive source-sink-paths in the graph that separately represent the time-based processing sequence of every job. The implication of simultaneously fixing $z_{1,1}^t = 1$ and $z_{3,1}^{t+1} = 1$ is shown using bold-faced arcs in Figure 3.4. Since both operations require machine M_1, there is flow occurring on the arcs, which express the passing of the time periods t and $t+1$ on this machine. To fulfill conservation constraints, the flow is completed starting at the source node and ending at the sink node.

More detailed descriptions of the transformation of sequencing decisions into a network flow are given in [68] and [93]. The authors apply this modeling and solution approach to single-track train scheduling problems, even if the number of variables is expected to be considerably larger and the required graph-theoretic structure is more complicated compared to other sequence-defining variables. The foregoing general consideration clearly shows that a mathematical formulation involving time-indexed variables significantly differs from models constituted by precedence or order-position variables. Since this work is intended to be focused on scheduling problems in the light of disjunctive programming and to do a fair

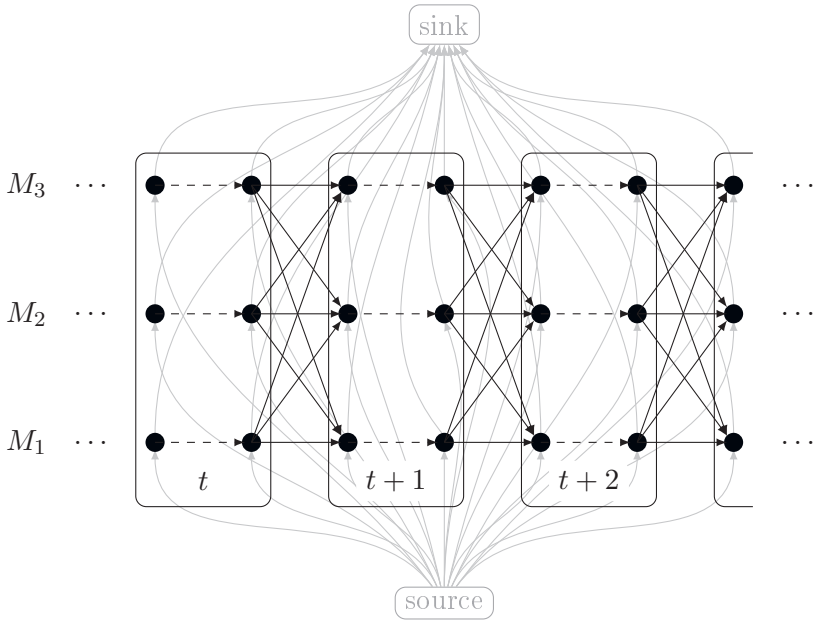


Figure 3.3: Partial time-expanded graph representation of the instance GJSP2

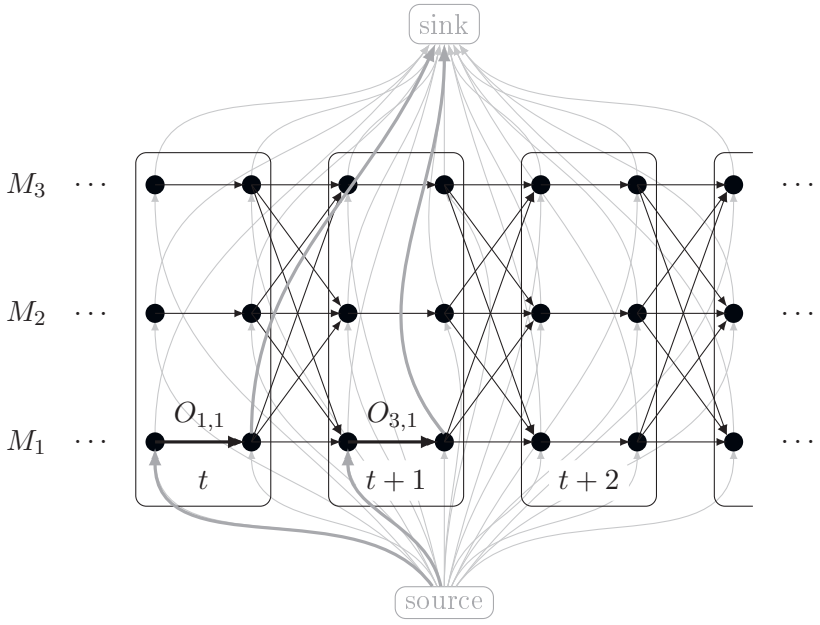


Figure 3.4: Illustration of a partial schedule in the time-expanded graph of instance GJSP2

comparison, time-indexed variables will not be included in the subsequent computational experiments.

3.2 A Mathematical Formulation for the BJSPT based on Precedence Variables

The BJSPT can be modeled by a mixed-integer linear optimization program based on the precedence variables $y_{i,j,i',j'}$ introduced in Subsection 3.1.1 as follows, cf. [78].

Mixed-Integer Linear Programming Formulation 1

$$\sum_{J_i \in \mathcal{J}} T_i \rightarrow \min! \tag{3.1}$$

subject to

$$\begin{aligned}
T_i &\geq C_i - d_i && J_i \in \mathcal{J} && (3.2)\\
T_i &\geq 0 && J_i \in \mathcal{J} && (3.3)\\
r_i &\leq s_{i,1} && J_i \in \mathcal{J} && (3.4)\\
s_{i,j} + p_{i,j} &\leq s_{i,j+1} && O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,n_i}\}, J_i \in \mathcal{J} && (3.5)\\
s_{i,n_i} + p_{i,n_i} &= C_i && J_i \in \mathcal{J} && (3.6)\\
y_{i,j,i',j'} + y_{i',j',i,j} &= 1 && O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i < i', && \\
& && M_k \in \mathcal{M} && (3.7)\\
s_{i',j'} + M(1 - y_{i,j,i',j'}) &\geq s_{i,j} + p_{i,j} && O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', && \\
& && M_k \in \mathcal{M} && (3.8)\\
s_{i',j'} + M(1 - y_{i,j,i',j'}) &\geq s_{i,j+1} && O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', j \neq n_i, && \\
& && M_k \in \mathcal{M} && (3.9)\\
y_{i,j,i',j'} &\in \{0,1\} && O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', && \\
& && M_k \in \mathcal{M} && (3.10)\\
M &\gg 0 && && (3.11)
\end{aligned}$$

Mixed-Integer Linear Programming Formulation 1 (MF1) begins with the implementation of the optimization criterion in (3.1). The objective function is constituted by the sum of the tardiness values T_i of all jobs $J_i \in \mathcal{J}$. The constraints in Inequalities (3.2) and (3.3) ensure the correct determination of the tardiness of all jobs J_i as the maximum of the difference between completion time and due date $(C_i - d_i)$ and the lower bound of 0. The Inequalities (3.4) restrict the starting times of the first operations of the jobs not to be earlier than the given release dates. In accordance with assumption 4 in Section 2.3.3, the processing sequence of all operations of a particular job is assured by the constraints (3.5) for all jobs $J_i \in \mathcal{J}$. In Equation (3.6), the completion time C_i of each job is defined as the end of the processing time of its last operation O_{i,n_i}.

The following set of constraints (3.7), (3.8) and (3.9) is related to the sequencing decisions that are implemented by the precedence variables. Equation (3.7) assures the uniqueness, completeness and exclusiveness of the indicator variables by the definition of the pairwise connection of variables, both corresponding to the operations $O_{i,j}$ and $O_{i',j'}$. Thereby, an exact ordering is determined for each pair of operations of different jobs that requires the same machine. According to these sequencing decisions, the starting time relations of the operations are set to be effective or redundant. Inequality (3.8) determines the starting time of the succeeding operation $O_{i',j'}$ to be greater than or equal to the sum of the starting time and the processing time of the preceding operation $O_{i,j}$, if $y_{i,j,i',j'} = 1$ holds. Otherwise, the sufficiently large positive constant M appears as a summand on the left-hand side of these so-called indicator or Big-M constraints, cf. [16] and [27], and allows $s_{i',j'}$ to take an arbitrary value. This formulation constitutes a well-known linearization of the basic disjunction given in (2.22), cf. for instance [16], [27] and [32]. The blocking constraints are modeled by Inequality (3.9) applying the same linearization. The starting time of a succeeding operation $O_{i',j'}$ is restricted to be greater than or equal to the starting time of the job successor of the preceding operation $O_{i,j}$, if the ordering $O_{i,j} \rightarrow O_{i',j'}$ is defined on a particular machine. Thus, it is guaranteed that the preceding operation has left the machine or is leaving the machine at the starting time of the succeeding operation.

Table 3.1: Number of constraints in Mathematical Formulation 1 based on precedence variables

Type	Number of Constraints
(3.2) Calculation of delay	n
(3.3) Non-negativity of tardiness	n
(3.4) Release date fulfillment	n
(3.5) Processing sequences	$n_{op} - n$
(3.6) Completion time calculation	n
(3.7) Pairwise relation of precedence variables	$\sum_{M_k \in \mathcal{M}} \left[\frac{1}{2} \cdot \mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right]$
(3.8) Disjunctive starting time constraints	$\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right]$
(3.9) Blocking Constraints	$\left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) - n$

The indicator variables are set to be binary in constraint (3.10) and the positive constant M is described as required by constraint (3.11). For the remaining variables, in particular the starting times $s_{i,j}$ of the operations as well as the completion times C_i and the tardiness times T_i of the jobs $J_i \in \mathcal{J}$, there is no necessity to introduced non-negativity or integrality restrictions. Both characteristics automatically result from the structure of the model, namely the restrictions (3.2) to (3.6) and the optimization criterion, and the input data given as non-negative and strictly positive integers, respectively, (assumptions 6 and 7 in Section 2.3.3).

Besides the number of variables involved in a mathematical formulation, the number of constraints is an indicator to the required amount of computational effort for solving an instance. Table 3.1 summarizes the quantities of constraints in terms of general problem size measures. Consequently, the maximum total number of restrictive constraints involved in MF1 for BJSPT can be determined as the summation of the given expressions.

$$\begin{aligned}
\#\text{con(MF1)} &= 2 \cdot n + n_{op} + \sum_{M_k \in \mathcal{M}} \left[\frac{1}{2} \cdot \mid \Omega^k \mid \cdot \Big(\mid \Omega^k \mid - 1 \Big) \right] + \\
&\quad 2 \cdot \sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \Big(\mid \Omega^k \mid - 1 \Big) \right] \\
&= 2 \cdot n + n_{op} + \frac{5}{2} \cdot \sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \Big(\mid \Omega^k \mid - 1 \Big) \right]
\end{aligned}$$

As a special case, consider the standard job shop scheduling problem with blocking constraints, where every job needs to be processed on every machine exactly once. The maximum quantity of restrictive constraints can be expressed in polynomial terms of the number of jobs n and the number of machines m.

$$\begin{aligned}
\#\text{con(MF1)} &= 2 \cdot n + n \cdot m + \frac{5}{2} \cdot m \cdot n \cdot (n-1) \\
&= \frac{5}{2} \cdot n^2 \cdot m - \frac{3}{2} \cdot n \cdot m + 2 \cdot n \\
&= O(n^2 \cdot m)
\end{aligned}$$

Note that these terms represent an upper bound on the total number of restrictive constraints in the mathematical formulation, since the disjunctive starting time constraints given in (3.8) and the blocking constraints set up by (3.9) constitute redundant constraints when operations $O_{i,j}$ with $j \neq n_i,\ J_i \in \mathcal{J}$ are involved, see the explanations on the redundancy of arcs in the disjunctive and the alternative graph in Section 2.3.2.

3.3 A Mathematical Formulation for the BJSPT based on Order-Position Variables

The BJSPT can alternatively be modeled as a mixed-integer linear optimization program based on the order-position variables $x_{i,j}^r$ introduced in Subsection 3.1.1, cf. [78].

Mixed-Integer Linear Programming Formulation 2

$$\sum_{J_i \in \mathcal{J}} T_i \rightarrow \min! \tag{3.12}$$

subject to

$$T_i \geq C_i - d_i \qquad J_i \in \mathcal{J} \tag{3.13}$$

$$T_i \geq 0 \qquad J_i \in \mathcal{J} \tag{3.14}$$

$$r_i \leq s_{i,1} \qquad J_i \in \mathcal{J} \tag{3.15}$$

$$s_{i,j} + p_{i,j} \leq s_{i,j+1} \qquad O_{i,j} \in \mathcal{O}^i \setminus \{O_{i,n_i}\}, J_i \in \mathcal{J} \tag{3.16}$$

$$s_{i,n_i} + p_{i,n_i} = C_i \qquad J_i \in \mathcal{J} \tag{3.17}$$

$$\sum_{r \in \mathcal{R}^k} x_{i,j}^r = 1 \qquad O_{i,j} \in \Omega^k, M_k \in \mathcal{M} \tag{3.18}$$

$$\sum_{O_{i,j} \in \Omega^k} x_{i,j}^r \leq 1 \qquad r \in \mathcal{R}^k, M_k \in \mathcal{M} \tag{3.19}$$

$$s_{i',j'} + M(2 - x_{i,j}^r - x_{i',j'}^{r+1}) \geq s_{i,j} + p_{i,j} \qquad O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', r \in \mathcal{R}^k \setminus \{R_k\}, M_k \in \mathcal{M} \tag{3.20}$$

$$s_{i',j'} + M(2 - x_{i,j}^r - x_{i',j'}^{r+1}) \geq s_{i,j+1} \qquad O_{i,j}, O_{i',j'} \in \Omega^k, \text{ with } i \neq i', j \neq n_i, r \in \mathcal{R}^k \setminus \{R_k\}, M_k \in \mathcal{M} \tag{3.21}$$

$$x_{i,j}^r \in \{0,1\} \qquad O_{i,j} \in \Omega^k, r \in \mathcal{R}^k, M_k \in \mathcal{M} \tag{3.22}$$

$$M \gg 0 \tag{3.23}$$

In Mixed-Integer Linear Programming Formulation 2 (MF2), the optimization criterion (3.12) as well as the constraints (3.13) to (3.17) are equivalent to those presented in MF1 in the previous section. Completion time and tardiness time calculations, release time fulfillment and the processing sequences are implemented identically.

The relations given in (3.18) to (3.21) model the sequencing decisions by applying order-position variables. Equality (3.18) assures that every operation that requires a particular machine M_k is assigned to exactly one order-position $r \in \mathcal{R}^k$. Reversely, Inequality (3.19) imposes the occupancy of every available order-position by at most one operation. Considering the fact that $R_k = |\Omega^k|$ holds for every machine $M_k \in \mathcal{M}$, the second set of assignment constraints is fulfilled with equality by any feasible solution. Starting time restrictions of operations consecutively processed on the same machine are set up by Inequality (3.20). Similar to MF1, the disjunctive constraints are linearized applying a Big-M formulation. If an operation $O_{i,j}$ is assigned to an order-position r directly prior to an operation $O_{i',j'}$ at order-position $r+1$, the starting time of the succeeding operation $O_{i',j'}$ is determined not to be earlier than the end of processing of the preceding operation $O_{i,j}$. An inequality of this type will only be active for the starting times $s_{i',j'}$ and $s_{i,j}$, when both considered operations are assigned to the regarded pair of adjacent order-positions. In all other cases, the large positive constant M will appear with coefficient 1 or 2 as a summand on the left-hand side of this Big-M constraint and, thus, it will be redundant. Following the same linearized structure, the blocking constraints are given by Inequality (3.21). Let a pair of operations $O_{i,j}$ and $O_{i',j'}$ be assigned to the order-positions r and $r+1$, respectively, the processing of the succeeding operation $O_{i'j'}$ is allowed to begin simultaneously to or after operation $O_{i,j+1}$ of job J_i being processed on another machine.

Constraint (3.22) defines the order-position variables to be binary and constraint (3.23) states the requirement for the constant parameter M. As explained in Section 3.2, the non-negativity and the integrality of the remaining variables is automatically implied by the formulation.

The resulting quantities of constraints that are included in the optimization program are summarized in Table 3.2. The maximum total number of

Table 3.2: Number of constraints in Mathematical Formulation 2 based on order-position variables

Type	**Number of Constraints**
(3.13) Calculation of delay	n
(3.14) Non-negativity of tardiness	n
(3.15) Release date fulfillment	n
(3.16) Processing sequences	$n_{op} - n$
(3.17) Completion time calculation	n
(3.18) Order-position assignment	n_{op}
(3.19) Operation assignment	$\sum_{M_k \in \mathcal{M}} R_k = \sum_{M_k \in \mathcal{M}} \mid \Omega^k \mid = n_{op}$
(3.20) Disjunctive starting time constraints	$\left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) \cdot \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 2 \right) \right] \right)$
(3.21) Blocking constraints	$\left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) - n$

restrictive constraints involved in MF2 for the BJSPT can be estimated by the summation of these terms as follows:

$$
\begin{aligned}
&\#\text{con(MF2)} \\
&= 2 \cdot n + 3 \cdot n_{op} + \\
&\left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) \cdot \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) - \mid \Omega^k \mid \right] \right) \\
&\quad + \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) \\
&= \; 2 \cdot n + 3 \cdot n_{op} + \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right)^2 - \\
&\quad n_{op} \cdot \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) + \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right) \\
&= \; 2 \cdot n + 3 \cdot n_{op} + \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right)^2 + \\
&\quad (1 - n_{op}) \cdot \left(\sum_{M_k \in \mathcal{M}} \left[\mid \Omega^k \mid \cdot \left(\mid \Omega^k \mid - 1 \right) \right] \right)
\end{aligned}
$$

With regard to the standard variant of a job shop scheduling problem with blocking constraints, the maximum total number of restrictive constraints occurring in MF2 can be expressed in polynomial dependence on n, the number of jobs, and m, the number of machines.

$$
\begin{aligned}
\#\text{con(MF2)} &= 2 \cdot n + 3 \cdot n_{op} + \left[m \cdot \left(n^2 - n \right) \right]^2 + (1 - n_{op}) \cdot m \cdot \left(n^2 - n \right) \\
&= n^4 \cdot m^2 - 2 \cdot n^3 \cdot m^2 + n^2 \cdot m^2 + (1 - n \cdot m) \cdot (n^2 - n) + \\
&\quad 3 \cdot n \cdot m + 2n \\
&= n^4 \cdot m^2 - 2 \cdot n^3 \cdot m^2 + n^2 \cdot m^2 - n^3 \cdot m + n^2 \cdot m + \\
&\quad 3 \cdot n \cdot m + n^2 + n \\
&= O(n^4 \cdot m^2)
\end{aligned}
$$

Note again that these terms state upper bounds on the quantity of restrictive constraints in the model, since the redundancy of disjunctive starting time constraints (3.20) and blocking constraints (3.21) equivalently appears in MF2.

3.4 Lower Bounds as a Key Feature of the Exact Solution Approach

The Branch & Bound procedure constitutes the best known and widely utilized exact algorithm to solve job shop scheduling problems and their applications based on mathematical formulations, cf. for instance [46], [31] and [134]. A key feature of this method is the definition of a lower bound on the objective function value of any completion of a partial schedule to reason the continuation or termination of the considered branch, cf. for instance [25] and [102]. The most popular definition of lower bounds for the mixed-integer programs under study is based on the linear relaxation of the binary constraints, cf. for instance [102]. With regard to the mathematical formulations for the BJSPT given in Sections 3.2 and 3.3, this implies replacing the binary restrictions in (3.10) and (3.22) by

$$0 \leq y_{i,j,i',j'} \leq 1 \qquad O_{i,j}, O_{i',j'} \in \Omega^k \text{ with } i \neq i', M_k \in \mathcal{M} \tag{3.24}$$

and

$$0 \leq x^r_{i,j} \leq 1 \qquad O_{i,j} \in \Omega^k, r \in \mathcal{R}^k, M_k \in \mathcal{M}, \tag{3.25}$$

respectively. The objective function value obtained by solving the resulting linear program constitutes a lower bound on the objective function value of the underlying BJSPT. Unfortunately, it is well-known that indicator and Big-M constraints lead to weak linear relaxations and accordingly unsatisfactory lower bounds, cf. [16] and [27].

To observe this issue in more detail, consider the instance GJSP1 represented by the disjunctive graph in Figure 2.2 in Section 2.3.1 as an example. The problem consists of two jobs J_i and $J_{i'}$, where two pairs of operations $O_{i,2}$ and $O_{i',2}$ as well as $O_{i,1}$ and $O_{i',3}$ require the same machines M_{k_1} and M_{k_2}, respectively. Without loss of generality, $r_i = r_{i'} = 0$ and $p_{i,j} = 1$ for $O_{i,j} \in \mathcal{O}$ is assumed. Furthermore, $d_i = d_{i'} = 0$ is considered as a special

case of the problem under study and consequently $C_i + C_{i'}$ is minimized. Since the processing times are constant and predefined, the minimization of the completion time $C_i = s_{i,3} + p_{i,3}$ of a job J_i corresponds to the minimization of the starting time of its last operation. Thus, the objective function can be reformulated as $s_{i,3} + s_{i',3}$ for the instance GJSP1 here.

To set up the concrete optimization program, the parameter M, which is used to impose or exclude constraints, needs to be defined as a sufficiently large constant. A commonly applied estimate in the literature is the largest possible appearing opponent of the Big-M terms in the set of constraints, cf. [27]. This can be implemented by the upper bound on the makespan for the BJSPT and defines $M = \max_{J_i \in \mathcal{J}}\{r_i\} + \sum_{O_{i,j} \in \mathcal{O}} p_{i,j}$ which equals 6 for the considered instance. The following model constitutes the relaxed formulation of MF1 applying precedence variables for GJSP1.

Linear Relaxation of MF1 applying Precedence Variables for GJSP1

$$s_{i,3} + s_{i',3} \rightarrow \min! \tag{3.26}$$

subject to

$$0 \leq s_{i,1} \tag{3.27}$$
$$0 \leq s_{i',1} \tag{3.28}$$

$$s_{i,1} + 1 \leq s_{i,2} \tag{3.29}$$
$$s_{i,2} + 1 \leq s_{i,3} \tag{3.30}$$
$$s_{i',1} + 1 \leq s_{i',2} \tag{3.31}$$
$$s_{i',2} + 1 \leq s_{i',3} \tag{3.32}$$
$$y_{i,2,i',2} + y_{i',2,i,2} = 1 \tag{3.33}$$
$$y_{i,1,i',3} + y_{i',3,i,1} = 1 \tag{3.34}$$
$$s_{i',2} + 6 \cdot (1 - y_{i,2,i',2}) \geq s_{i,2} + 1 \tag{3.35}$$
$$s_{i,2} + 6 \cdot (1 - y_{i',2,i,2}) \geq s_{i',2} + 1 \tag{3.36}$$
$$s_{i',3} + 6 \cdot (1 - y_{i,1,i',3}) \geq s_{i,1} + 1 \tag{3.37}$$
$$s_{i,1} + 6 \cdot (1 - y_{i',3,i,1}) \geq s_{i',3} + 1 \tag{3.38}$$

$$s_{i',2} + 6 \cdot (1 - y_{i,2,i',2}) \geq s_{i,3} \tag{3.39}$$

$$s_{i,2} + 6 \cdot (1 - y_{i',2,i,2}) \geq s_{i',3} \tag{3.40}$$

$$s_{i',3} + 6 \cdot (1 - y_{i,1,i',3}) \geq s_{i,2} \tag{3.41}$$

$$y_{i,2,i',2},\ y_{i',2,i,2},\ y_{i,1,i',3},\ y_{i',3,i,1} \geq 0 \tag{3.42}$$

$$y_{i,2,i',2},\ y_{i',2,i,2},\ y_{i,1,i',3},\ y_{i',3,i,1} \leq 1 \tag{3.43}$$

The relaxed formulation incorporates release date and processing sequence restrictions in (3.27) to (3.32), interdependencies of the precedence variables in (3.33) and (3.34), disjunctive constraints implementing the consecutive processing of operations on machines in (3.35) to (3.38), blocking constraints in (3.39) to (3.41) and the linear relaxation of the binary constraints in (3.42) and (3.43). Note that the model does only include three explicit blocking constraints, since the operation $O_{i',3}$ involved in one of the sequencing decisions constitutes the last operation of job $J_{i'}$ and the opponent restriction of inequality (3.41) determines $s_{i,1} \geq 0$ already given in (3.27). The relaxed model of GJSP1 can easily be solved by setting all indicator variables as follows:

$$y_{i,2,i',2} = 0.5,\ y_{i',2,i,2} = 0.5,\ y_{i,1,i',3} = 0.5,\ y_{i',3,i,1} = 0.5.$$

These values satisfy the pairwise summation to 1 as well as the relaxed binary constraints. Furthermore, a constant summand of 3 appears on the left-hand-side of each Big-M inequality. This enables the starting times of all operations to take their minimum values as to minimize the objective function. Accounting for the release dates and processing sequences of the jobs in (3.27) to (3.32), the following starting times complete the solution of the linear relaxation of the model.

$$s_{i,1} = 0,\ s_{i,2} = 1,\ s_{i,3} = 2, \quad \text{and} \quad s_{i',1} = 0,\ s_{i',2} = 1,\ s_{i',3} = 2$$

Since these values determine the smallest possible objective function value $s_{i,3}+s_{i',3} = 2+2 = 4$ for the given input data, the solution is optimal for the relaxed model of the BJSPT. Figure 3.5 illustrates the resulting schedule in a Gantt chart. Both jobs J_i and $J_{i'}$ are processed from the beginning of the planning horizon $t = 0$ and completed after their total processing time at $t = 3$ without any waiting times. Thus, $C_i + C_{i'} = 3+3 = 6$ constitutes

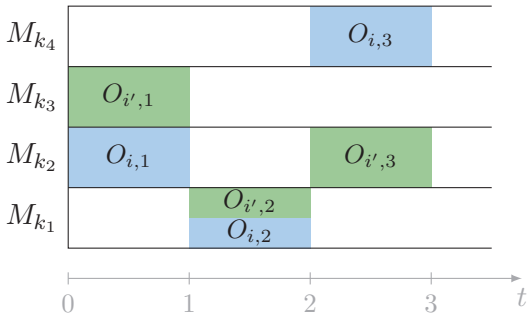


Figure 3.5: Gantt chart representation of the schedule resulting from the linear relaxation of the mathematical formulation for the instance GJSP1

a lower bound on the total completion time of the initial GJSP1. In the relaxed solution, the indicator variables $y_{i,2,i',2} = y_{i',2,i,2} = 0.5$ allow the simultaneous processing of the operations $O_{i,2}$ and $O_{i',2}$ on machine M_{k_1}. Since this is strictly contradicting a basic assumption in machine scheduling, see Section 1.2, a total completion time equal to this lower bound can never be realized by any feasible schedule for the instance.

Generally, the example shows that the lower bound on the total completion time of every job J_i will always approximate $r_i + \sum_{O_{i,j} \in \mathcal{O}^i} p_{i,j}$. Regarding the optimization criterion considered in this thesis, the lower bound on the tardiness of each job $J_i \in \mathcal{J}$ is consequently determined by $T_i = 0$, since $d_i > r_i + \sum_{O_{i,j} \in \mathcal{O}^i} p_{i,j}$ holds by the definition of practically relevant the due date factors. Instances of the BJSPT with an actual optimal objective function value of $\sum_{J_i \in \mathcal{J}} T_i = 0$ are not likely to appear, especially, with increasing job-machine-ratio $\frac{n}{m}$ and tight due dates. Thus, the linear relaxation of MF1 provides a fairly unsatisfactory and constant lower bound for the considered optimization criterion, where the main reason is constituted by the Big-M constraints, cf. [27].

Considering the alternative formulation of the BJSPT using order-position variables and taking the assumptions given above, the linear relaxation of the optimization program for the instance GJSP1 can be stated as follows:

Linear Relaxation of MF2 applying Order-Position Variables for GJSP1

$$s_{i,3} + s_{i',3} \rightarrow \min! \tag{3.44}$$

subject to

$$0 \leq s_{i,1} \tag{3.45}$$
$$0 \leq s_{i',1} \tag{3.46}$$
$$s_{i,1} + 1 \leq s_{i,2} \tag{3.47}$$
$$s_{i,j} + 1 \leq s_{i,3} \tag{3.48}$$
$$s_{i',1} + 1 \leq s_{i',2} \tag{3.49}$$
$$s_{i',j'} + 1 \leq s_{i',3} \tag{3.50}$$
$$x^1_{i,3} = 1 \tag{3.51}$$
$$x^1_{i',1} = 1 \tag{3.52}$$
$$x^1_{i,1} + x^2_{i,1} = 1 \tag{3.53}$$
$$x^1_{i',3} + x^2_{i',3} = 1 \tag{3.54}$$
$$x^1_{i,2} + x^2_{i,2} = 1 \tag{3.55}$$
$$x^1_{i',2} + x^2_{i',2} = 1 \tag{3.56}$$
$$x^1_{i,3} \leq 1 \tag{3.57}$$
$$x^1_{i',1} \leq 1 \tag{3.58}$$
$$x^1_{i,1} + x^1_{i',3} \leq 1 \tag{3.59}$$
$$x^2_{i,1} + x^2_{i',3} \leq 1 \tag{3.60}$$
$$x^1_{i,2} + x^1_{i',2} \leq 1 \tag{3.61}$$
$$x^2_{i,2} + x^2_{i',2} \leq 1 \tag{3.62}$$

$$s_{i',3} + 6 \cdot (2 - x^1_{i,1} - x^2_{i',3}) \geq s_{i,1} + 1 \tag{3.63}$$
$$s_{i,1} + 6 \cdot (2 - x^1_{i',3} - x^2_{i,1}) \geq s_{i',3} + 1 \tag{3.64}$$
$$s_{i',2} + 6 \cdot (2 - x^1_{i,2} - x^2_{i',2}) \geq s_{i,2} + 1 \tag{3.65}$$
$$s_{i,2} + 6 \cdot (2 - x^1_{i',2} - x^2_{i,2}) \geq s_{i',2} + 1 \tag{3.66}$$
$$s_{i',3} + 6 \cdot (2 - x^1_{i,1} - x^2_{i',3}) \geq s_{i,2} \tag{3.67}$$
$$s_{i',2} + 6 \cdot (2 - x^1_{i,2} - x^2_{i',2}) \geq s_{i,3} \tag{3.68}$$

$$s_{i,2} + 6 \cdot (2 - x^1_{i',2} - x^2_{i,2}) \geq s_{i',3} \tag{3.69}$$

$$x^1_{i,1}, x^2_{i,1}, x^1_{i,2}, x^2_{i,2}, x^1_{i,3}, x^1_{i',1}, x^1_{i',2}, x^2_{i',2}, x^1_{i',3}, x^2_{i',3} \geq 0 \tag{3.70}$$

$$x^1_{i,1}, x^2_{i,1}, x^1_{i,2}, x^2_{i,2}, x^1_{i,3}, x^1_{i',1}, x^1_{i',2}, x^2_{i',2}, x^1_{i',3}, x^2_{i',3} \leq 1 \tag{3.71}$$

The model consists of a set of release date and processing sequence constraints (3.45) to (3.50), restrictions assuring the order-position assignment for each operation in (3.51) to (3.56) and the operation assignment for each order-position in (3.57) to (3.62), linearized disjunctive constraints implementing the consecutive processing of the operations on the machines by (3.63) to (3.66) and blocking by (3.67) to (3.69) as well as the relaxed binary constraints (3.70) and (3.71) next to the optimization criterion (3.44). With regard to the solution of this relaxed program, the set of order-variables is divided into two subsets. First, there exist variables indicating the order-position of operations, which are solely processed on a particular machine, here $x^1_{i',1}$ and $x^1_{i,3}$. Such variables are directly determined by the assignment constraints with $x^1_{i',1} = x^1_{i,3} = 1$ in any feasible solution. The remaining variables implement actual sequencing decisions and similar to the solution derived for the linear relaxation of MF1, the following values satisfy the given set of constraints.

$$x^1_{i,1} = x^2_{i,1} = x^1_{i,2} = x^2_{i,2} = x^1_{i',2} = x^2_{i',2} = x^1_{i',3} = x^2_{i',3} = 0.5$$

The starting times of all operations are equivalently driven to take smallest possible values in accounting for release dates and processing sequences. Accordingly,

$$s_{i,1} = 0,\ s_{i,2} = 1,\ s_{i,3} = 2, \quad \text{and} \quad s_{i',1} = 0,\ s_{i',2} = 1,\ s_{i',3} = 2$$

determine the optimal solution given in Figure 3.5 from the linear relaxation of MF2 for the BJSPT. As a consequence, the mathematical formulation based on order-position variables is expected to produce the same unsatisfactory lower bound on the total tardiness for the BJSPT.

The idea of defining tighter lower bounds for different types of job shop scheduling problems is pursued by many researchers, whereby the standard job shop problem with makespan minimization constitutes the most intensively studied issue. The best known approach introduced by Carlier in 1982 relies on the capacity relaxation of all machines except one, which

leads to the separate optimization of the sequences on the single machines while considering operation release times and due times. These so-called heads and remaining tails of operations are derived from the disjunctive graph representation of a partial schedule as longest paths. The maximal completion time determined over all machines facilitates as the lower bound on the overall makespan of the problem. An enhanced version of this method is presented for instance by Carlier and Pinson in [38]. Even if the computation of the single-machine bound itself is NP-hard, it can be done efficiently for makespan minimization and the method is widely used and adapted, cf. [120]. Masics and Pacciarelli [88], applied the idea based on the alternative graph to a blocking job shop problem with makespan minimization. Furthermore, an extended version of the capacity relaxation method is derived by Carballo et al. to solve different variants of multiprocessor job shop problems, cf. [37].

As an alternative approach, Taillard [117] presents lower bounds on the makespan of flow shop and job shop scheduling problems, where heads and tails are determined machine-based and added to the total processing time of all operations on the considered machine. The maximum value of the resulting makespan estimates is stated as the lower bound on the objective function value of the problem. This bound is not expected to outperform the lower bound defined by single-machine optimization but there is less computational effort required. Brucker and Jurisch [31] test the resolution of the two-job relaxation of a given job shop problem for the purpose of determining good lower bounds on the makespan. Experimental evidence shows that even a job shop scheduling problem involving two jobs demands for a considerable amount of computational effort and the bounds obtained do not significantly improve compared to the single-machine relaxation. Applegate and Cook [7] apply the idea of Balas [11] in adding valid inequalities, so-called cutting planes, to the problem to reduce the search space and steadily improve the lower bounds during the execution of a Branch & Bound procedure. This technique is not exclusively applied to job shop scheduling, but constitutes a main module in general MIP solvers up to the present.

Nonetheless, the achieved results remain unsatisfactory compared to what can be obtained for other NP-hard combinatorial problems, cf. [120], and

the general Branch & Bound mechanism is not expected to efficiently solve job shop instances of sizes relevant in practice. Recently, Bonami et al. [27] derive a nonlinear reformulation of optimization programs with logical implications to overcome the weak relaxations discussed above. The authors show that the computationally challenging construction of the reformulation is possible for particular classes of disjunctive optimization problems, among them the standard job shop scheduling problem with makespan minimization. Unfortunately, the achieved improvement in the lower bounds, which is reported for a large set of benchmark instances, is not significant enough to believe in general-purpose solvers becoming competitive against scheduling-tailored methods, cf. [27]. Fischetti and Monaci [54] indicate the applicability of MIP solvers when combined with fitted preprocessing techniques. A real-world train rescheduling problem with total delay minimization is solved by determining an upper bound on the objective function value through a structured trial and error method prior to the actual resolution. The authors emphasize the important role of this bound for the following variable tightening and fixing mechanisms incorporated in the Branch & Bound method. Zhou and Zhong [134] present a commonly known Lagrangian relaxation dual bound, which requires considerable computational effort. In addition, they define lower bounds for partial schedules in the considered train timetabling problem with total travel time minimization based on delay estimates for single conflict resolutions. The influence of this simple bounding technique increases with the size of the existing partial schedule, so that it needs to be combined with another more general method, which is applied in the early stages of the solving process.

Overall, there exist various attempts to obtain lower bounds on the makespan for the job shop scheduling problem as well as more or less tailored MIP techniques. However, none of these approaches can easily be adapted to compute satisfactory lower bounds on the total tardiness for a considered scheduling problem. Thus, it can be expected that even a state-of-the-art MIP solver will not be able to compute optimal solutions for all benchmark instances given in Section 2.5 in reasonable runtime. Accordingly, the computational study presented in the following section is intended to indicate the boundaries of resolvability of the BJSPT by a general exact method

and to provide benchmark results for the heuristic methods derived later in this thesis.

3.5 Analysis of Solving the BJSPT as a Mixed-Integer Program

In the following, the key features of the MIP formulations, which are set up in Sections 3.2 and 3.3, are summarized referring to the benchmark instances introduced in Section 2.5. The quantities of involved variables and constraints are compared and the resolvability of the BJSPT modeled by MF1 and MF2, respectively, is tested with the use of a well-known general optimization software. The instance properties, which are presented in Section 2.5.3, are additionally discussed in relation to the performance of the state-of-the-art MIP solver. ZIMPL, the open source modeling language introduced by Thorsten Koch [73] at Zuse Institute Berlin, is chosen to implement the mathematical formulations. Applying a highly sophisticated Branch & Bound procedure, the computational experiments are performed by IBM ILOG CPLEX 12.8.0 (simply referred to as CPLEX in the following) on a notebook operating an Intel Dual Core i5 processor (2.20 GHz) with 8 GB RAM.

The Tables 3.3 and 3.4 depict the descriptive figures of the mathematical formulations and three quantities characterizing the computational results for the train scheduling-inspired instances modeled by MF1 and MF2, respectively. The first two columns display the instance label and its size (n, m). The columns #var(MF) and #con(MF) contain the number of variables and the number of constraints of the mathematical formulation of the given instance. Furthermore, the last three columns indicate the best found total tardiness value $\sum T_i$, the percentage gap between the objective function value of the best found feasible solution and the lower bound as well as the runtime of the solution procedure in seconds. A time limit of two hours is set to the execution of the optimization software. Solutions with proven optimality are denoted by an asterisk (*). Considering the last three columns, the entry 'none' as the total tardiness value expresses that there is no integer feasible solution found for the given instance and the gap

is denoted by infinity (inf) accordingly. Furthermore, the percentage gap is stated with a maximum value of 100.00%, which is shown for differences greater than or equal to this measure. In the same way, the computational results are summarized for the Lawrence instances modeled by MF1 with precedence variables in Table 3.5 and stated by MF2 with order-position variables in Table 3.6.

Comparison of the quantities of variables and constraints in the mathematical formulations. Due to the structural variation of the train scheduling-inspired instances, where most of the train routes do not visit all of the tracks, compared to the Lawrence instances, which are created as standard job shop problems with $n_i = m$ for all $J_i \in \mathcal{J}$, a difference appears in the descriptive measures of the models. Each of the 15 ts-instances features a distinct quantity of variables and constraints involved, whereas these measures are equal for all Lawrence instances of the same size (n, m). Nonetheless, the numbers of variables and constraints do naturally increase for all benchmark problems and formulations with an increasing size of the corresponding instance. Comparing the amounts of variables and constraints involved in different formulations of the same instance, the models set up by MF1 with precedence variables include less unknowns and significantly less constraints than the programs created by MF2 with order-position variables. This effect can be observed for both train scheduling-inspired and Lawrence instances. In particular, it is remarkable that the number of constraints increases over-proportionally in MF2 when the number of considered jobs and machines increases, while in MF1, the ratio between the number of variables and the number of constraints is almost constant for all benchmark problems. These observations empirically support the theoretical findings of the Sections 3.1.2, 3.2 and 3.3.

On the resolvability of the BJSPT by applying MF1 versus MF2. With regard to the train scheduling-inspired instances, Table 3.3 shows that nine of the 15 problems given as optimization programs with precedence variables are solved to optimality within the time limit of two hours. In contrast, no feasible solution is found by CPLEX for the instances when implementing the train scheduling problems with order-position variables, see Table 3.4. A similar behavior can be observed for the Lawrence instances. As indicated in Table 3.5, 16 out of the 40 benchmark instances

are solved to optimality while being described by MF1. In comparison, Table 3.6 displays that there is no optimal solution found by CPLEX for the instances given by MF2. These results signalize that a mathematical formulation using precedence variables should clearly be preferred against a model based on order-position variables for the purpose of solving the BJSPT by a general optimization software. Evidently, the structural issues of implementing sequencing decisions discussed in Section 3.1.3 seem to effect the performance of a MIP solver.

Aspects of solving the BJSPT by MIP techniques. In Section 3.4, difficulties in the definition of good lower bounds for the BJSPT are highlighted and explained. The numerical results obtained by CPLEX on the entire set of benchmark instances demonstrate the appearance and significance of this issue. Consider the results for the Lawrence instances implemented by MF2 in Table 3.6, there exists a feasible solution for each of the 40 instances, but for 38 of them, the percentage gap is given with 100.00 %. This implies that the lower bound on the total tardiness of these problems is estimated by 0 although a feasible solution is known and the Branch & Bound procedure has been run for two hours. Considering the mathematical formulation based on precedence variables, the results are generally improved compared to order-position variables but there are still remarkably large percentage gaps remaining. Based on the results in Tables 3.3 and 3.5, it is not clear whether the best found feasible solutions of the instances are actually far from the optimal schedules or whether it is just a matter of proving optimality. ts09 gives evidence to the latter idea, since the total tardiness of 153 can be proven to be optimal after two more hours of computation time. Thus, an enhancement in lower bounds for mathematical formulations of scheduling problems is expected to improve the performance of Branch & Bound methods on these types of instances. In addition, general MIP solvers use the solution of the linear relaxation of a model to obtain a feasible solution for the initial mixed-integer problem. Table 3.4 shows that for the structured ts-instances implemented based on order-position variables, CPLEX does not succeed in finding any feasible solution. A similar behavior is also reported by Fischetti and Monaci [54] while solving real-world train rescheduling problems. This gives rise to the conjecture that the linear relaxation incorporates an exceptionally

Table 3.3: Computational results of MF1 for the train scheduling-inspired instances solved by CPLEX 12.8.0

Inst.	(n, m)	#var(MF1)	#con(MF1)	$\sum T_i$	Gap	Time
ts01	$(10, 11)$	530	1146	138*	0	5
ts02	$(10, 11)$	498	1072	90*	0	1
ts03	$(10, 11)$	518	1120	72*	0	1
ts04	$(10, 11)$	558	1215	41*	0	1
ts05	$(10, 11)$	490	1052	71*	0	2
ts06	$(15, 11)$	1145	2564	88*	0	390
ts07	$(15, 11)$	1199	2689	172*	0	4865
ts08	$(15, 11)$	1149	2565	163*	0	5514
ts09	$(15, 11)$	1189	2668	153	34.90	7200
ts10	$(15, 11)$	1115	2488	97*	0	435
ts11	$(20, 11)$	1994	4532	366	74.57	7200
ts12	$(20, 11)$	2038	4643	419	75.26	7200
ts13	$(20, 11)$	2106	4802	452	76.72	7200
ts14	$(20, 11)$	2036	4628	459	78.67	7200
ts15	$(20, 11)$	2084	4755	418	57.65	7200

Table 3.4: Computational results of MF2 for the train scheduling-inspired instances solved by CPLEX 12.8.0

Inst.	(n, m)	#var(MF2)	#con(MF2)	$\sum T_i$	Gap	Time
ts01	$(10, 11)$	608	7014	none	inf	7200
ts02	$(10, 11)$	568	6283	none	inf	7200
ts03	$(10, 11)$	596	6730	none	inf	7200
ts04	$(10, 11)$	644	7868	none	inf	7200
ts05	$(10, 11)$	568	6235	none	inf	7200
ts06	$(15, 11)$	1266	23505	none	inf	7200
ts07	$(15, 11)$	1328	25652	none	inf	7200
ts08	$(15, 11)$	1270	23552	none	inf	7200
ts09	$(15, 11)$	1318	25292	none	inf	7200
ts10	$(15, 11)$	1228	22455	none	inf	7200
ts11	$(20, 11)$	2142	55316	none	inf	7200
ts12	$(20, 11)$	2186	57140	none	inf	7200
ts13	$(20, 11)$	2262	60674	none	inf	7200
ts14	$(20, 11)$	2200	57400	none	inf	7200
ts15	$(20, 11)$	2256	60252	none	inf	7200

Table 3.5: Computational results of MF1 for the Lawrence instances solved by CPLEX 12.8.0

Inst.	(n, m)	#var(MF1)	#con(MF1)	$\sum T_i$	Gap	Time
la01	$(10, 5)$	520	1005	762*	0	5
la02	$(10, 5)$	520	1005	266*	0	1
la03	$(10, 5)$	520	1005	357*	0	1
la04	$(10, 5)$	520	1005	1165*	0	33
la05	$(10, 5)$	520	1005	557*	0	2
la06	$(15, 5)$	1155	2520	2516	23.68	7200
la07	$(15, 5)$	1155	2520	1677*	0	2886
la08	$(15, 5)$	1155	2520	1829*	0	4622
la09	$(15, 5)$	1155	2520	2851	32.20	7200
la10	$(15, 5)$	1155	2520	1841*	0	4213
la11	$(20, 5)$	2040	4510	6534	80.89	7200
la12	$(20, 5)$	2040	4510	5286	81.85	7200
la13	$(20, 5)$	2040	4510	7737	85.59	7200
la14	$(20, 5)$	2040	4510	6038	81.08	7200
la15	$(20, 5)$	2040	4510	7082	83.69	7200
la16	$(10, 10)$	1020	2280	330*	0	4
la17	$(10, 10)$	1020	2280	118*	0	1
la18	$(10, 10)$	1020	2280	159*	0	3
la19	$(10, 10)$	1020	2280	243*	0	2
la20	$(10, 10)$	1020	2280	42*	0	1
la21	$(15, 10)$	2280	5220	1956	67.64	7200
la22	$(15, 10)$	2280	5220	1455	58.68	7200
la23	$(15, 10)$	2280	5220	3436	90.24	7200
la24	$(15, 10)$	2280	5220	560*	0	155
la25	$(15, 10)$	2280	5220	1002	34.79	7200

continued on the next page

Inst.	(n, m)	#var(MF1)	#con(MF1)	$\sum T_i$	Gap	Time
la26	$(20, 10)$	4040	9360	7961	96.30	7200
la27	$(20, 10)$	4040	9360	8915	96.49	7200
la28	$(20, 10)$	4040	9360	2226	78.58	7200
la29	$(20, 10)$	4040	9360	2018	86.97	7200
la30	$(20, 10)$	4040	9360	6655	94.94	7200
la31	$(30, 10)$	9060	21240	20957	99.16	7200
la32	$(30, 10)$	9060	21240	23150	98.92	7200
la33	$(30, 10)$	9060	21240	none	inf	7200
la34	$(30, 10)$	9060	21240	none	inf	7200
la35	$(30, 10)$	9060	21240	none	inf	7200
la36	$(15, 15)$	3405	7920	675	42.19	7200
la37	$(15, 15)$	3405	7920	1070	82.52	7200
la38	$(15, 15)$	3405	7920	489*	0	467
la39	$(15, 15)$	3405	7920	754	72.24	7200
la40	$(15, 15)$	3405	7920	407*	0	1082

Table 3.6: Computational results of MF2 for the Lawrence instances solved by CPLEX 12.8.0

Inst.	(n, m)	#var(MF2)	#con(MF2)	$\sum T_i$	Gap	Time
la01	(10, 5)	570	7910	2109	94.40	7200
la02	(10, 5)	570	7910	3862	100.00	7200
la03	(10, 5)	570	7910	4494	100.00	7200
la04	(10, 5)	570	7910	3750	100.00	7200
la05	(10, 5)	570	7910	1558	82.35	7200
la06	(15, 5)	1230	27765	9796	100.00	7200
la07	(15, 5)	1230	27765	8068	100.00	7200
la08	(15, 5)	1230	27765	9204	100.00	7200
la09	(15, 5)	1230	27765	11403	100.00	7200
la10	(15, 5)	1230	27765	11931	100.00	7200
la11	(20, 5)	2140	67220	24443	100.00	7200
la12	(20, 5)	2140	67220	23608	100.00	7200
la13	(20, 5)	2140	67220	20978	100.00	7200
la14	(20, 5)	2140	67220	24133	100.00	7200
la15	(20, 5)	2140	67220	29618	100.00	7200
la16	(10, 10)	1120	16610	3314	100.00	7200
la17	(10, 10)	1120	16610	7170	100.00	7200
la18	(10, 10)	1120	16610	6039	100.00	7200
la19	(10, 10)	1120	16610	3397	100.00	7200
la20	(10, 10)	1120	16610	7266	100.00	7200
la21	(15, 10)	2430	58440	8928	100.00	7200
la22	(15, 10)	2430	58440	18522	100.00	7200
la23	(15, 10)	2430	58440	19444	100.00	7200
la24	(15, 10)	2430	58440	16911	100.00	7200
la25	(15, 10)	2430	58440	20807	100.00	7200

continued on the next page

Inst.	(n, m)	#var(MF1)	#con(MF1)	$\sum T_i$	Gap	Time
la26	$(20, 10)$	4240	141620	44186	100.00	7200
la27	$(20, 10)$	4240	141620	59038	100.00	7200
la28	$(20, 10)$	4240	141620	50300	100.00	7200
la29	$(20, 10)$	4240	141620	50545	100.00	7200
la30	$(20, 10)$	4240	141620	62194	100.00	7200
la31	$(30, 10)$	9360	489030	140756	100.00	7200
la32	$(30, 10)$	9360	489030	169308	100.00	7200
la33	$(30, 10)$	9360	489030	168525	100.00	7200
la34	$(30, 10)$	9360	489030	187311	100.00	7200
la35	$(30, 10)$	9360	489030	144064	100.00	7200
la36	$(15, 15)$	3630	89115	34571	100.00	7200
la37	$(15, 15)$	3630	89115	37565	100.00	7200
la38	$(15, 15)$	3630	89115	36463	100.00	7200
la39	$(15, 15)$	3630	89115	29359	100.00	7200
la40	$(15, 15)$	3630	89115	35286	100.00	7200

low amount of restrictiveness on the sequence-defining variables as to letting MIP-based techniques fail to construct a feasible solution from the continuous values.

On the general resolvability of the BJSPT. In general, the computational results obtained by the state-of-the-art MIP solver emphasize that the job shop scheduling problem remains one of the hardest combinatorial optimization problems. Equivalently testing on instances involving structured patterns against pure randomness, the BJSPT can be expected to be hard to solve when incorporating more than 150 operations, $n_{op} \geq 150$, or including significantly more jobs than machines, $\frac{n}{m} \geq 3$. Certainly, a guaranteed statement on the resolvability of a problem cannot be given based on these trivial properties. Nonetheless, the ts-instances and the Lawrence instances, both show their boundaries of resolvability with instance sizes of $(15, 11)$ as well as $(15, 5)$, $(15, 10)$ and $(15, 15)$, respectively. This observation implies that blocking constraints and total tardiness minimization complicate the problem under study, since the transition between easy and hard instances occurs at higher job-machine-ratios for the standard job shop scheduling problem, cf. [118]. Considering real-world instances of the BJSPT, it can be expected that more jobs and machines are involved while less computation time is demanded compared to the presented experiments. Thus, the computational results give evidence to the fact that even a state-of-the-art general-purpose optimization software is not a reasonable choice to tackle occurring instances of the BJSPT in practice. Therefore, a heuristic approach, which is tailored to the structure of the considered scheduling problem, is introduced and tested in the subsequent chapter of this thesis.

Relating instance properties and computational results. To properly discuss the computational experiments with regard to the instance properties presented in Section 2.5.3, the results obtained by CPLEX solving MF1 are considered exclusively, see Tables 3.3 and 3.5. Among the ts-instances of size $(15, 11)$, two instances, namely ts06 and ts10, are solved with a remarkably less amount of computation time compared to the others. It can be seen in Table 2.3, that these instances feature the smallest mean machine utilization rate $\bar{u}$ combined with the highest mean machine slack $\bar{l}$. In line with this observation, the instance la28 and la29 show the lowest

percentage gaps after two hours of computation as against the other problems involving ten machines and 20 jobs. Furthermore, the resolvability of la24 and the relatively small percentage gap shown by la25 could have been expected according to low mean machine utilization rates and high mean machine slacks. Supporting the same observation with a reverse behavior, the instance la04 requires more computational effort than the other Lawrence instances of the same size while showing the highest mean utilization rate and the lowest mean machine slack. The same holds for the problems la06 and la09 in comparison to the other instances with five machines and 15 jobs as well as for problem la23 among the instances of size $(15, 10)$. Moreover, the Lawrence instances of size $(15, 10)$ seem to be rather diverse with regard to the ranges of their mean machine utilization rates and mean machine slacks. This is reflected by the computational results, since la24 is solved to optimality in less than three minutes, whereas la23 shows a percentage gap of more than 90 % after two hours of computation. Thus, the interdependency of the two averaged instance properties seems to constitute an indicator for blocking job shop instances to be relatively easy or hard to solve by MIP methods.

According to these observations, it is to be mentioned that blocking constraints and the considered objective function seem to shift the relative difficulty of problems. The instances la21, la27, la29 and la38 are reported to be exceptionally challenging as standard job shop scheduling problems with makespan minimization, cf. [7], [13] and [120], while they do not show any indication for this property here. On the contrary, la38 constitutes one of the two instances, which are optimally solved within minutes among the five $(15, 15)$-instances.

Nonetheless, there are some drawbacks of the proposed relation between the instance properties and the computational results to be taken into account. Since the magnitude of the measures of utilization and slack is dependent on the job-machine-ratio $\frac{n}{m}$, an indication can only be derived comparing instances of the same size (n, m). Additionally, the considered properties do only take effect at the boundary of resolvability. Regarding the instances of size $(10, 10)$ in Table 3.5, the problem la17 features the lowest mean machine utilization rate $\bar{u}$ together with the highest mean machine slack $\bar{l}$. However, these properties do not project to the compu-

tational results, since the $(10, 10)$-instances are easily solvable in general. Furthermore, the correlation between $\bar{u}$ and $\bar{l}$ does not form a comprehensively reliable reference for all benchmark instances tested here. The resolvability of la38 and la40 among the $(15, 15)$-instances and the complexity of ts09 cannot be explained by these measures.

A conclusion, which can be drawn from the reported experiments, is that the numbers of variables and constraints do not apply as a sensitive indicator of resolvability. Comparing these generic quantities for the instances la06 to la10 and la16 to la20, it can observed that the formulations are of the same magnitude while the computational results are significantly different. It seems that a more complex property measure is required to cover occurring interdependencies and compensating effects. Observing the instance ts15 as an example, this problem shows a significantly smaller percentage gap compared to the other ts-instances involving 20 trains. In contrast to the arguments given above, ts15 features the highest mean machine utilization rate and a medium mean machine slack among all $(20, 11)$-instances. Here, the standard deviation of the machine utilization rate $std(u_k)$ can be observed to be remarkably high compared to the other problems and, thus, there might occur certain bottleneck machines, which are recognized and structurally used by the MIP technique.

Overall, the instance properties presented in Section 2.5.3 can be used as an indication for the expected performance of a general resolution method. However, precise conclusions on characteristics, which lead to instances that are easy or hard to be solved by mixed-integer programming, cannot be drawn based on the given results. Further studies are required to derive and test more complex property measures combined with additional observations on the performance variability of MIP solvers on blocking job shop problems, cf. [85], which is not considered here.

4 Heuristic Methods for the Blocking Job Shop Problem with Total Tardiness Minimization

Since many real-world planning situations demand for the development of good solutions in reasonable runtime, the computational experiments reported in the previous chapter indicate the necessity of applying scheduling-tailored methods to solve BJSPT instances in practice. Therefore, the main purpose of the following sections is the introduction, the theoretical analysis and the empirical testing of heuristic techniques for the problem under study. Initial results obtained by applying a simulated annealing metaheuristic, which is addressed here in detail, are published as a joint work with Frank Werner in [79].

At first, Section 4.1 reviews existing metaheuristic methods applied to blocking job shop instances and related problems. Section 4.2 describes distinct representations of a schedule by permutations of operations, cf. [79], since these lists constitute the basis of the heuristic approaches derived in this thesis. A distance between two schedules is defined in Section 4.3 as an important independent measure for the ongoing analysis. Occurring issues of feasibility and redundancy of permutations are discussed in Section 4.4. A procedure, which constructs a feasible schedule from any given permutation in polynomial time, is presented in Section 4.5, cf. [79]. Based on this, solution techniques of different complexity are discussed and tested on the BJSPT. Section 4.6 contains the description of various popular priority rules, which constitute a basic module of the proposed heuristic procedures. Three neighborhood structures are introduced and analyzed in Section 4.7, where an emphasis is given to significant feasibility issues occurring in the generation of neighboring solutions, the connectivity and further empirically observable properties. Sections 4.8 and 4.9 combine and apply the presented priority rules and neighborhoods in a Local Neighborhood Search (LS) and a Simulated Annealing (SA) framework, respectively. The computational results obtained by both methods are compared to the outcome of the state-of-the-art MIP solver presented in Section 3.5.

4.1 Existing Heuristic Methods for Job Shop Problems with Blocking Constraints and Tardiness-Based Objectives

This section is meant to summarize the most popular metaheuristic approaches tackling job shop scheduling problems for the purpose of indicating applications and highlighting heuristic techniques, which are related to the key issues discussed in this thesis. For comprehensive reviews of the literature presenting heuristic methods on job shop scheduling problems in general, the reader is for instance referred to Anderson, Glass and Potts [5], Blazewicz, Domschke and Pesch [22] and Blazewicz et al. [25]. The metaheuristics involved in the following survey are

- *Priority Rules* (also called dispatching rules),
- *Local Neighborhood Search (LS)* (meaning local descent or iterative improvement schemes in a strict sense),
- *Simulated Annealing (SA)*,
- *Tabu Search* and
- *Evolutionary Algorithms* (including genetic algorithms as a specific type).

General explanations on how to apply these procedures to scheduling problems are given for example by Pinedo [102]. Most of the existing methods are not proposed for the BJSPT, but approach a more or less extended version of the standard job shop problem introduced in Section 2.2. In the following, special emphasis is given to the incorporation of blocking constraints and the consideration of tardiness-based objectives to classify the metaheuristics. Further characteristics of the job shop problem under study, such as the general technological routes including recirculation and the tolerance of swaps, are additionally mentioned to highlight the contribution of the study conducted here.

Priority rules are set up to guide the decision on which operation to be scheduled next on a specific machine in the shop. Most of the dispatching rules proposed in the literature are created to control dynamic job shop environments, where jobs appear dynamically throughout the planning horizon. Gere [59] applies and compares several slack-based priority rules for the static and dynamic job shop problem with total tardiness minimization. The experimental results show that the enhancement of basic dispatching rules by additional mechanisms, which suspend the general regulation in urgent situations, improves the performance. Baker and Kanet [9] introduce a priority rule based on modified operation due dates and report promising results on dynamic job shop scheduling instances involving recirculation and tardiness-based objectives. A priority rule tailored to handle weighted tardiness costs is presented by Vepsalainen and Morton [123] for a dynamic job shop problem facing different utilization levels and due date factors. Anderson and Nyirenda [6] show that two rules perform well on dynamic job shop problems with recirculation and tardiness-based objectives, while combining existing priority measures. Supporting this idea,

further studies are conducted, which highlight positive effects obtained by linking priority rules to benefit from different properties, cf. [70], [71] and [104]. However, many authors point out that the performance of a certain priority rule is not robust but depends on the workload in the shop, the due date tightness and the considered objective function, cf. for instance [6], [71] and [123]. These findings imply that different dispatching rules should be comparatively applied and more complex procedures are required to appropriately tackle complicated settings like the BJSPT.

In contrast to the methods summarized above, the following metaheuristics require an initial solution, from which neighboring solutions are derived and the search space is explored. In various implementations, priority rules are used to set up the initial feasible schedule, cf. for instance [4], [13] and [64]. Thereafter, the guidance of the search is a key issue in almost every heuristic method. Special attention is paid to the combination of intensification to exploit locally optimal solutions and diversification, so that an early entrapment in a local optimum is avoided, cf. for instance [13], [67] and [97].

With regard to *local neighborhood search* techniques, an observation identical to the statement on priority rules can be made. Most of the methods do not only include a simple local descent scheme but apply certain enhancements to tackle the more complex problem structure. Balas and Vazacopoulos [13] present a guided local search procedure for the standard job shop problem, where neighboring solutions are constructed based on interchanges of operations in the critical path in the disjunctive graph. To diversify the search, the graph-based neighborhood is combined with a shifting bottleneck-based scheme. Additionally, the search is guided by a neighborhood tree, in which considered solutions are stored and linked. The procedure obtains good solutions to all known benchmark instances in reasonable time. Similarly, Rego and Duarte [105] apply a filter and fan algorithm to the standard job shop problem likewise incorporating a local descent scheme and a diversification technique.

Considering the blocking job shop scheduling problem, Oddi et al. [98] apply an iterative improvement algorithm, which incorporates a constraint-based search procedure with relaxation and reconstruction steps. A schedule with minimal makespan is to be found, while swaps are allowed in fea-

sible solutions. Pranzo and Pacciarelli [103] introduce an iterated greedy algorithm based on the alternative graph representation for the same problem setting. The authors show that the proposed method, which loops a destruction and a construction phase, is well applicable to instances with and without swaps. Both heuristics are tested on the Lawrence instances without recirculation. The results imply that an operator forcing neighboring solutions to be more diverse is favorable to solve job shop problems with blocking constraints.

Simulated annealing constitutes a basic metaheuristic involving the acceptance of deteriorated solutions to overcome the entrapment in local optima. The most popular application of an SA to a standard job shop scheduling problem with makespan minimization is published by van Laarhoven, Aarts and Lenstra [122]. The authors introduce the reversion of a critical arc in the longest path in the disjunctive graph of an existing schedule as the transition scheme and this represents a basic makespan-tailored neighborhood structure to the present. Matsuo, Suh and Sullivan [90] show that this neighborhood can be reduced by pairs of operations, for which the predecessor of the first and the successor of the second operation in the corresponding processing sequences do also belong to the critical path. Steinhöfel, Albrecht and Wong [112] extend the transition scheme through permitting the reversion of more than one arc and apply it in an SA with two different cooling schemes to the standard job shop problem. In addition, the authors implement the original neighborhood of van Laarhoven, Aarts and Lenstra in a parallelized version of an SA algorithm, cf. [113]. Kolonko [74] shows that the SA as a stochastic process does not feature the convergence property when applied to job shop scheduling problems. Therefore, the execution of multiple runs starting from different initial solutions is reasonable when the performance of the algorithm or its components is evaluated.

Minimizing the total tardiness of standard job shop instances, Wang and Wu [125] indicate that a generic neighborhood interchanging a pair of adjacent operations on a machine leads to better results compared to a critical path-based transition scheme when applied in an SA. However, Zhang and Wu [133] implement the idea of combining intensification and diversification by operating the neighborhood of van Laarhoven, Aarts

and Lenstra together with a block reversion operator working on the dual maximum flow problem. The authors emphasize that the combination of the neighborhoods is beneficial against the application of one separate operator when minimizing the total weighted tardiness. Kuhpfahl and Bierwirth [76] use a local descent scheme and an SA algorithm as basic frameworks to compare different neighborhood structures for a job shop scheduling problem with release dates and total weighted tardiness minimization. The experimental results show that transition schemes vary in their performance when implemented in different metaheuristics and with different initial solutions. In line with the observation in [133], more complex neighborhood structures with critical path-based components yield convincing results with regard to solution quality and computational effort.

Indicated as the most promising metaheuristic for job shop scheduling problems, cf. [5] and [120], *tabu search* is widely applied in the literature. Taillard [118] implements the transition scheme of van Laarhoven, Aarts and Lenstra mentioned above in a sequential and a parallel tabu search algorithm. Nowicki and Smutnicki [96] introduce a second critical path-based neighborhood for the standard job shop problem, which is reduced to interchanges of adjacent pairs of operations on the same machine at specific positions in critical blocks. The presented tabu search algorithm additionally includes a procedure to collect and restore elite schedules when the search is trapped in a local optimum. The authors provide a significantly enhanced version of their method in [97], which further benefits from a fast a priori makespan evaluation of neighboring solutions and a stronger focus on the set of elite schedules.

Incorporating blocking constraints in the job shop problem causes considerable feasibility issues, as briefly outlined in Section 2.4. This aspect becomes even more challenging during the construction of neighboring solutions when relying on generic operators. Occurring issues and their resolutions are comprehensively discussed in Sections 4.4, 4.5 and 4.7. Mati, Rezg and Xie applied the transition scheme given in [122] and a job insertion-based rescheduling procedure to create feasible neighbors. The authors operate their tabu search approach to minimize the makespan of blocking job shop problems without swaps and recirculation based on the

alternative graph. Gröflin and Klinkert [64] present a connected neighborhood, which is set up by interchanges of adjacent operations and job reinsertion, for the blocking job shop problem with makespan minimization while involving setup and transfer times. In this study, a critical path-dependent and accordingly reduced neighborhood is applied in a tabu search algorithm additionally including a list of elite solutions to be restored after a certain amount of non-improving iterations. Bürgy [34] provides computational experiments on a tabu search approach to the blocking job shop problem with regular objective functions including total tardiness. Swaps are allowed in the schedule and neighboring solutions are obtained by a job reinsertion technique initially proposed in [63]. The metaheuristic is applied to benchmark instances without release dates and with a moderate due date factor of 1.8.

The most popular class of *evolutionary algorithms* is constituted by genetic algorithms. Comprehensive descriptions of components and operators as well as their application to different kinds of shop scheduling problems are given by Werner in [128]. A more specific survey on hybrid genetic search strategies for job shop scheduling problems can be found in [41]. Considering the standard job shop problem, Yamada and Nakano [131] propose a genetic algorithm where the completion times of the operations represent a solution and the crossover operator is based on the well-known construction scheme by Giffler and Thompson [60]. Contrastingly, Della Croce, Tadei and Volta [48] apply a machine-based preference list representation of a schedule together with a general crossover. Their genetic algorithm is extended by a lookahead evaluation scheme and an updating mechanism to reduce redundancy in the population. Both heuristic techniques are tested on benchmark instances showing good results. Dorndorf and Pesch [50] used the genetic algorithm framework to obtain the most effective construction scheme with flexible priority rule application as well as to identify a beneficial machine ordering for a shifting bottleneck procedure introduced in [2]. It turns out that the shifting bottleneck-based genetic algorithm produces promising results for standard job shop scheduling instances with general technological routes.

Tackling a job shop problem, which involves recirculation and release dates while minimizing tardiness-based objective functions, Mattfeld and Bier-

wirth [92] apply a hybrid genetic algorithm with a permutation-based encoding. The list representation is interpreted as priority relations to decide about the ordering of conflicting operations. The procedure is enhanced by a flexible encoding scheme and a decomposition approach to reduce computation time. Nonetheless, the results do not significantly support genetic algorithms as a favorable solution method. Considering blocking constraints, tolerating swaps in a feasible schedule and minimizing the makespan, Brizuela, Shao and Sannomiya [29] propose a genetic algorithm with a permutation-based encoding, which actually acts as a scheduling order of the operations. The authors derive a procedure to construct a feasible schedule from any given permutation and test their method on benchmark instances of standard job shop problems. Regarding the same problem but following another evolutionary approach, AitZai and Boudhar [4] apply a parallel particle swarm optimization, which turns out not to be competitive to the methods proposed in [64] and [103]. All these results indicate that evolutionary techniques are basically applicable to the BJSPT, but the effort required to adapt the encoding and the transition operators to the scheduling problem is not profitable against solution quality and runtime.

In line with this observation, *scheduling-tailored heuristics* accounting for the specific structure of the problems are developed. The most popular and successfully applied solution technique for job shop scheduling problems is the shifting bottleneck procedure initially proposed by Adams, Balas and Zawack [2]. The main idea incorporates the sequential resolution of single machine problems and the repeated adaptation of the operation sequences on the machines. As mentioned above, this method is used in several hybridized metaheuristics. Solving a practical job shop problem with release dates and due dates, the shifting bottleneck strategy is applied by Balas et al. [12] as a modified variant of the guided local search heuristic given in [13]. Another scheduling-specific procedure based on the sequential insertion of jobs in the schedule is proposed by Werner and Winkler [129]. This work constitutes the fundamental module of job reinsertion techniques implemented in some metaheuristics given above and a job insertion-based neighborhood applied in [34], [63] and [64]. Such heuristic methods yield

promising results in reasonable runtime while being integrated in a metaheuristic framework.

Analyzing the advantages of scheduling-tailored structures in lean heuristic mechanisms, this thesis is intended to shed light upon the applicability of permutation-based representations of schedules, basic list scheduling techniques and generic transition schemes in the construction of neighborhoods for blocking job shop schedules. In contrast to the work by Oddi et al. [98] and Pranzo and Pacciarelli [103], simple ordering-based operators are observed for the blocking job shop problem, so that the magnitude of destruction is more precisely controllable and a moderate runtime for the transition can be expected. The remarkable results obtained by van Laarhoven, Aarts and Lenstra [122], Nowicki and Smutnicki [96] and Bürgy [34] give evidence for focusing on the development of a sophisticated neighborhood instead of creating complex resolution procedures. The studies presented by Gröflin and Klinkert [64], Brizuela, Shao and Sannomiya [29] as well as Mati, Rezg and Xie [89] motivate the following work on a broader discussion and evaluation of permutations and generic interchange operators in metaheuristics while optimizing a practice-driven objective function. Furthermore, the popular idea of reducing the number of considered neighbors by specific characteristics, as applied for makespan minimization, cf. [96] and [122], is observed with regard to its effectiveness for the minimization of total tardiness.

4.2 Permutation-Based Representations of a Schedule

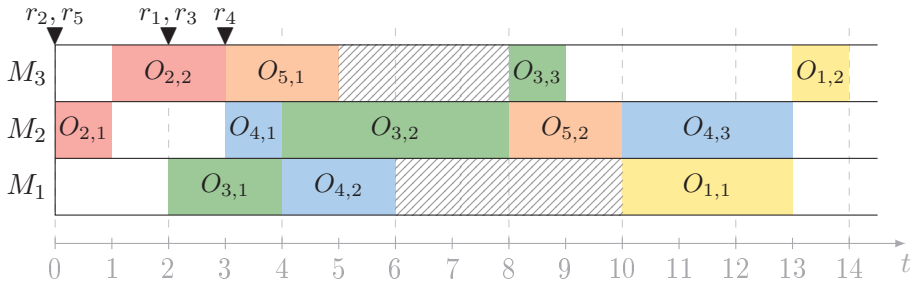
There exist various structures to represent a schedule in the job shop scheduling framework, cf. for instance [5], [40] and [128]. Since it constitutes one of the key components of every heuristic method, the encoding is supposed to be as simple as possible, while containing all information required for the solution process. The representation by a permutation or list of operations is a well-known strategy in scheduling research, cf. [5] and [128], which is applied in the following. This is straightforward,

since a permutation is technically easy to handle and the schedule-defining operation sequences on the machines can directly be derived.

The two basic representations of a schedule, which are used in the ongoing study, cf. [79], are defined as follows: The *operation-based representation* $s^{op} = [\ldots]$ states the schedule as a permutation (a single list) of all operations $O_{i,j} \in \mathcal{O}$. Likewise, the *machine-based representation* $s^{ma} = [[\ldots],[\ldots],\ldots]$ displays the schedule as the sequences of operations on the machines (a nested list) starting from M_1 up to M_m. The machine-based representation and the starting times of the operations can be derived from the operation-based representation by applying a simple list scheduling technique to the permutation, which is illustrated in detail in Section 4.4.1. Hence, every operation-based representation uniquely defines a machine-based representation. Since a machine-based schedule only incorporates precedence relations between operations requiring the same machine, this implication does not hold in the opposite direction, cf. [5]. The encoding s^{ma} does not specify the ordering of operations in the permutation, which are processed on different machines.

Figure 4.1 shows three different representations of the schedule for the instance GJSP2 of the BJSPT constructed in Section 2.4. Based on the permutation s^{op} given in part (b) of the figure, the Gantt chart can be set up by consecutively assigning the starting times to the operations from the first, here operation $O_{2,1}$, to the last, namely operation $O_{1,2}$. In the same way, the machine-based representation s^{ma} outlined in part (c) at the bottom of Figure 4.1 can be derived from the permutation. Contrarily, the representation s^{ma} can be transformed into several different but redundant operation-based representations. This issue is discussed in more detail in the subsequent Section 4.4.1.

In order to examine and describe modifications of a schedule in a formal way, two quantities are defined, which specify the positions of the operations in the permutation-based representations. A list index $lidx(O_{i,j}) \in \{1,2,\ldots,n_{op}\}$ is assigned to every operation $O_{i,j}$ in the operation-based representation. More precisely, the list indices $lidx(O_{i,j})$ for $O_{i,j} \in \mathcal{O}$ are denoted as the positions of the operations in the permutation. Similarly, a machine index $midx(O_{i,j}) \in \{1,2,\ldots,R_k\}$ is determined for every operation $O_{i,j}$ in the machine-based representation, where R_k defines the



(a) Gantt-chart representation

$$s^{op} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}]$$

(b) Operation-based representation

$$s^{ma} = [[O_{3,1}, O_{4,2}, O_{1,1}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{5,2}, O_{4,3}], [O_{2,2}, O_{5,1}, O_{3,3}, O_{1,2}]]$$

(c) Machine-based representation

Figure 4.1: Different representations of a schedule for the instance GJSP2 of the BJSPT

number of operations requiring machine M_k. These indices refer to the order-positions of the operations on the corresponding machines. Regarding the schedule depicted in Figure 4.1, the list indices and machine indices are defined as follows:

$$\begin{array}{lcccccccc} s^{op} = & [O_{2,1}, & O_{3,1}, & O_{2,2}, & O_{4,1}, & O_{5,1}, & O_{4,2}, & O_{3,2}, & \dots \\ lidx(O_{i,j}) & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ & O_{4,3}, & O_{1,1}, & O_{1,2}] & & & & & \\ & 10 & 11 & 12 & & & & & \end{array}$$

$$\begin{array}{lcccccccc} s^{ma} = & [[O_{3,1}, & O_{4,2}, & O_{1,1}], & [O_{2,1}, & O_{4,1}, & O_{3,2}, & O_{5,2}, & O_{4,3}], \\ midx(O_{i,j}) & 1 & 2 & 3 & 1 & 2 & 3 & 4 & 5 \\ & [O_{2,2}, & \dots\,]] & & & & & & \\ & 1 & \dots & & & & & & \end{array}$$

4.3 Measuring the Distance of Schedules

When characterizing the set of solutions to a scheduling problem or the neighborhood structure, the distances between the incorporated schedules act as a well-known figure, cf. for instance [20] and [107]. Generally, the distance $\delta(s, s')$ of two solutions s and s' is defined by the minimal number of basic operators required to construct one schedule from the other, cf. [107] and [127]. A popular operator, which is used in this thesis, is the interchange of a pair of adjacent operations in the operation sequence of a machine. Here, this transition is denoted as an *adjacent pairwise interchange (API)*, while it is also referred to as a swap or a transpose in the literature, cf. for instance [5], [91], [107], [127] and [132]. Accordingly, the distance determination is related to the machine-based representations of the considered solutions. To count the number of required APIs, a binary indicator variable is introduced referring to two schedules s and s' as follows:

$$h_{i,j,i',j'} = \begin{cases} 1 & \text{if an ordering } O_{i,j} \to O_{i',j'} \text{ or } O_{i',j'} \to O_{i,j} \text{ in } s \\ & \text{is reverse in } s', \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

These variables are defined for pairs of operations $O_{i,j}$ and $O_{i',j'}$ requiring the same machine, for which $i < i'$ holds. This condition assures that there is only one indicator introduced for each pair of operations of different jobs belonging to the same set Ω^k. Thus, every API is counted exactly once. The distance of two schedules s and s' is determined by

$$\delta(s, s') = \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i<i'}} h_{i,j,i',j'} \ . \tag{4.2}$$

Note that the maximum distance of two schedules equals the total number of possible adjacent pairwise interchanges in the operation sequences on all machines $M_k \in \mathcal{M}$. Thus, the distance $\delta(s, s')$ is bounded by $\sum_{M_k \in \mathcal{M}} \binom{|\Omega^k|}{2}$.

With regard to the optimization program MF1 presented in Section 3.2, the statements $O_{i,j} \to O_{i',j'}$ in s and $O_{i',j'} \to O_{i,j}$ in s' can be expressed

by means of the precedence variables with $y^{s}_{i,j,i',j'} = 1$ and $y^{s}_{i',j',i,j} = 0$ as well as $y^{s'}_{i,j,i',j'} = 0$ and $y^{s'}_{i',j',i,j} = 1$. Based on this, the distance $\delta(s, s')$ can be related to the well-known Hamming distance, which is defined for binary bit strings in [66]. The Hamming distance $H(s, s')$ of two solutions s and s' corresponds to the number of list entries, here the number of precedence variables, showing different values in the compared permutations. Dependent on the implementation of MF1, two cases are to be observed.

- **Case 1.** If the solution vector consists of binary variables $y_{i,j,i',j'}$ for all operations $O_{i,j}, O_{i',j'} \in \Omega^k$ with $i \neq i'$ and all machines $M_k \in \mathcal{M}$, every API refers to *two* precedence variables changing their values. Thus,

$$\begin{aligned} H(s,s') &= \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i \neq i'}} \mid y^{s}_{i,j,i',j'} - y^{s'}_{i,j,i',j'} \mid \\ &= \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i < i'}} \left(\mid y^{s}_{i,j,i',j'} - y^{s'}_{i,j,i',j'} \mid \right) \\ &\quad + \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i < i'}} \left(\mid y^{s}_{i',j',i,j} - y^{s'}_{i',j',i,j} \mid \right) \\ &= \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i < i'}} 2 \cdot h_{i,j,i',j'} \\ &= 2 \cdot \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i < i'}} h_{i,j,i',j'} \;=\; 2 \cdot \delta(s,s'). \end{aligned}$$

- **Case 2.** If the solution vector consists of binary variables $y_{i,j,i',j'}$ for all operations $O_{i,j}, O_{i',j'} \in \Omega^k$ with $i < i'$ and all machines $M_k \in \mathcal{M}$, every API refers to *one* precedence variable changing its value, cf. [91] and [107]. Thus,

$$\begin{aligned} H(s,s') &= \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i < i'}} \mid y^{s}_{i,j,i',j'} - y^{s'}_{i,j,i',j'} \mid \\ &= \sum_{M_k \in \mathcal{M}} \sum_{\substack{O_{i,j}, O_{i',j'} \in \Omega^k \\ \text{with } i < i'}} h_{i,j,i',j'} \;=\; \delta(s,s'). \end{aligned}$$

4.4 Feasibility and Redundancy of Permutation-Based Schedules

In the following, redundancy and feasibility issues are addressed, which occur when working with permutations as solutions to combinatorial optimization problems. Particularly, it is shown that the feasibility of a schedule s for the BJSPT can only be checked on its operation-based representation s^{op} in Section 4.4.2, while the uniqueness can only be assured by its machine-based representation s^{ma}, see Section 4.4.1. Thus, both types of encoding are of equal importance to a permutation-based heuristic method and a reliable transformation scheme, as proposed in Section 4.4.1, needs to be applied.

4.4.1 The Redundancy of Permutations Representing Job Shop Schedules

As indicated in Section 4.2, a schedule uniquely given by a machine-based representation or by a Gantt chart with earliest operation starting times $s_{i,j}$ for all $O_{i,j} \in \mathcal{O}$ may result from several redundant permutations s^{op}. Every two adjacent operations in the permutation, which fulfill the following conditions, can be interchanged without effecting the represented schedule.

(a) The operations belong to different jobs.

(b) The operations require different machines.

(c) The operations are not connected by a blocking constraint.

(d) None of the operations is involved in a swap.

Recall the operation-based representation of the schedule shown in Figure 4.1.

$$\begin{aligned} s^{op} &= s_1^{op} \\ &= [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \end{aligned}$$

The adjacent pairs of operations $O_{2,1}$ and $O_{3,1}$, $O_{3,1}$ and $O_{2,2}$ as well as $O_{4,1}$ and $O_{5,1}$ feature the given properties and their interchanges can be applied

separately as well as simultaneously to construct different redundant permutations. Furthermore, the lists obtained from these interchanges need to be checked for newly appearing swappable pairs, since some redundancies are not visible in the initially considered permutation. Altogether, the schedule given in Figure 4.1 is redundantly expressible by ten permutations.

$$
\begin{aligned}
s_2^{op} &= [O_{3,1}, O_{2,1}, O_{2,2}, O_{4,1}, O_{5,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_3^{op} &= [O_{3,1}, O_{2,1}, O_{2,2}, O_{5,1}, O_{4,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_4^{op} &= [O_{2,1}, O_{2,2}, O_{3,1}, O_{4,1}, O_{5,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_5^{op} &= [O_{2,1}, O_{2,2}, O_{4,1}, O_{3,1}, O_{5,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_6^{op} &= [O_{2,1}, O_{2,2}, O_{4,1}, O_{5,1}, O_{3,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_7^{op} &= [O_{2,1}, O_{2,2}, O_{3,1}, O_{5,1}, O_{4,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_8^{op} &= [O_{2,1}, O_{2,2}, O_{5,1}, O_{3,1}, O_{4,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_9^{op} &= [O_{2,1}, O_{2,2}, O_{5,1}, O_{4,1}, O_{3,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}] \\
s_{10}^{op} &= [O_{2,1}, O_{3,1}, O_{2,2}, O_{5,1}, O_{4,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}]
\end{aligned}
$$

With regard to computation time, any heuristic method should be set up so that redundant permutations are not classified as individual solutions to a scheduling problem. Therefore, schedules are exclusively distinguished by their machine-based representations in the solution approaches proposed in this thesis. Nonetheless, since the discussion of blocking-related feasibility issues requires the operation-based representation of a schedule, there exists a necessity to transform one encoding into the other. Without loss of generality, it can be assumed that both, the given operation-based as well as the machine-based representation, depict feasible schedules with regard to the processing sequences and the technological routes of the jobs $J_i \in \mathcal{J}$, see details in the following Section 4.4.2.

Case 1: $s^{op} \rightarrow s^{ma}$**.** Starting from a permutation s^{op}, the machine-based representation s^{ma} can be derived by applying a list scheduling procedure, cf. [32]. The operations in the permutation s^{op} are considered with increasing list index and added to the operation sequences on the machines successively. Figure 4.2 illustrates the transformation process of the permutation s_1^{op} given above into the corresponding machine-based represen-

tation s^{ma} of the schedule, whereby the gray numbers indicate the ordering of the construction steps.

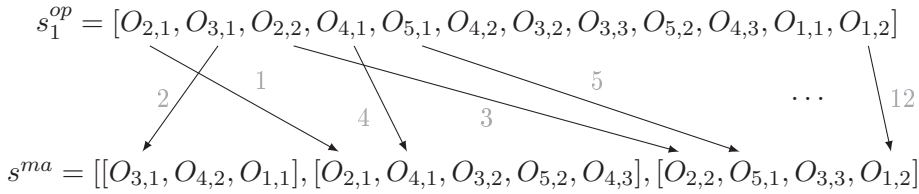


Figure 4.2: Transforming the operation-based representation into the machine-based representation of a schedule

Since the number of operations is finitely defined by n_{op} and every operation is read from the permutation and assigned to the machine-based encoding exactly once, this transformation can be operated in $O(n_{op})$ iterations.

Case 2: $s^{ma} \rightarrow s^{op}$**.** On the contrary, transforming a uniquely defined machine-based representation s^{ma} into a permutation s^{op} leads to decisions, with which different but redundant lists of operations can be created for the same schedule, cf. [5]. When considering the machine-based encoding of the schedule given in Case 1, every operation $O_{i,j}$ with $midx(O_{i,j}) = 1$ that has no predecessor in the processing sequence of its job qualifies for being at the first position in the permutation, here the operations $O_{3,1}$ and $O_{2,1}$. For general purposes, the decision on which operation to place first can be made arbitrarily, since the underlying schedules are equivalent. However, the application of a structured solving method aims at restoring equal permutations and constructing neighboring solutions without incorporating unnecessary changes. Therefore, a transformation scheme based on list index priorities is implemented.

Priority-Guided Transformation Scheme $s^{ma} \rightarrow s^{op}$

Let s^{op} be the operation-based representation of a schedule s with list indices $lidx(O_{i,j})$ and let $s^{ma'}$ be the machine-based encoding of a schedule s' with machine-indices $midx'(O_{i,j})$ for all operations $O_{i,j} \in \mathcal{O}$. Both schedules s and s' are assumed to be feasible with regard to the processing sequences and technological routes of all jobs. A permutation $s^{op'}$ representing the schedule s' with list indices $lidx'(O_{i,j})$ is to be derived, so that

as all entries of the list s^{op}, which do not define the difference between the schedule s and s', keep their absolute and relative positions when being transferred to $s^{op'}$. Therefore, the construction of the permutation $s^{op'}$ is guided by dynamic priorities $prio(O_{i,j})$ that are based on the a priori list indexes $lidx(O_{i,j})$ of all operations $O_{i,j} \in \mathcal{O}$. In case that several operations can redundantly be assigned to the currently considered position $lidx'(*)$ in the new permutation $s^{op'}$, the decision is made in favor of the operation featuring the maximum priority among the candidate operations, whereby

$$prio(O_{i,j}) = \begin{cases} \frac{1}{lidx(O_{i,j}) - lidx'(*)} & \text{if } lidx'(*) < lidx(O_{i,j}), \\ 2 & \text{if } lidx'(*) = lidx(O_{i,j}), \\ lidx'(*) - lidx(O_{i,j}) + 2 & \text{if } lidx(O_{i,j}) < lidx'(*). \end{cases} \tag{4.3}$$

Algorithm 1 describes the priority-guided transformation scheme. Technically, all operations $O_{i,j} \in \mathcal{O}$ are taken from the machine-based representation $s^{ma'}$ and inserted in the permutation $s^{op'}$, correspondingly. The a priori list indices of the operations in permutation s^{op} are managed in a dictionary LI, while the machine indices $midx'(O_{i,j})$ defining the schedule s' are stored in the MI dictionary. Every iteration of the **while**-loop creates a list $Cand$ of candidate operations from the machine-based encoding $s^{ma'}$. An operation $O_{i,j}$ constitutes a candidate of being assigned to the currently considered list index $lidx'(*)$, if it is first on its machine, $midx'(O_{i,j}) = 1$, and its predecessor $O_{i,j-1}$ does not exist or is already part of the permutation $s^{op'}$. Subsequently, the function PRIORITY(...) called within the **for**-loop in the lines 12 to 14 determines and stores the value $prio(O_{i,j})$ as defined in (4.3) for every operation in the candidate list. Based on this, the operation with maximal priority is added to the permutation $s^{op'}$ and deleted from the machine-based representation $s^{ma'}$ in lines 16 and 17. The currently regarded list index $lidx'(*)$ and the machine index dictionary are adapted in the lines 18 and 19. Finally, the candidate list is cleared in line 20 and the procedure continues until all operations are transferred to the permutation $s^{op'}$, which is returned.

Since the machine-based representation $s^{ma'}$ is feasible with regard to processing sequences and technological routes, it is guaranteed that exactly

Algorithm 1 Priority-Guided Transformation Scheme from s^{ma} to s^{op}

Input: (feasible) permutation s^{op}, (feasible) machine-based representation $s^{ma'}$
Output: (feasible) operation-based representation $s^{op'}$

initialize LI as list index dictionary with $lidx(O_{i,j})$ for $O_{i,j} \in \mathcal{O}$ from s^{op}
initialize MI as machine index dictionary with $midx'(O_{i,j})$ for $O_{i,j} \in \mathcal{O}$ from $s^{ma'}$

$s^{op'} \leftarrow$ empty list
$lidx'(*) \leftarrow 1$
$Cand \leftarrow$ empty list
while $s^{ma'}$ is not empty **do**
    for $O_{i,j}$ with $midx'(O_{i,j}) == 1$ **do**
        if $O_{i,j-1}$ does not exist **or** $O_{i,j-1} \in s^{op'}$ **then**
            add $O_{i,j}$ to $Cand$
        end if
    end for
    for $O_{i,j} \in Cand$ **do**
        PRIORITY($O_{i,j}$, LI, $lidx'(*)$)
    end for
    $O^* \leftarrow O_{i,j} \in Cand$ with maximum priority
    add O^* to $s^{op'}$
    delete O^* from s^{ma}
    $lidx'(*) \leftarrow lidx'(*) + 1$
    update MI
    clear $Cand$
end while
return $s^{op'}$

one operation is transferred from $s^{ma'}$ to $s^{op'}$ in every execution of the **while**-loop. Thus, the number of operations in the machine-based encoding decreases by one in every iteration and Algorithm 1 terminates due to the finite number of operations in the problem. With regard to the runtime of the transformation scheme, it is assumed that basic operations like reading and writing an element from or to a list or a dictionary, checking the existence of a specific element in a list or a dictionary as well as assigning a value to a variable can be done in constant time. The initialization, lines 1 and 2, is executed in $O(n_{op})$ steps. For correctly implemented processing sequences in the list $s^{ma'}$, the **while**-loop in the lines 6 to 21 is operated exactly n_{op} times. The function PRIORITY(...) in line 13 only involves dictionary calls and basic mathematical operations. Thus, the **for**-loops require $O(m)$ steps each, since there may be at most m operations added to the candidate list. Overall, this leads to a polynomial runtime of the priority-guided transformation scheme. Algorithm 1 constructs a permutation $s^{op'}$ from a given permutation s^{op} and a machine-based encoding $s^{ma'}$ in $O(n_{op} \cdot m)$ time.

4.4.2 The Infeasibility of Permutations Representing Job Shop Schedules

When using permutation-based representations of solutions in constrained optimization, the feasibility of the encoded solution is not intrinsically given but requires special attention, cf. [19]. The BJSPT constitutes a highly constrained scheduling problem, for which feasibility issues of permutations arise from the processing sequences of the jobs, the technological routes of the jobs and the blocking constraints.

The simplest form of infeasibility, which occurs in operation-based as well as in machine-based representations of the schedule, violates the processing sequences and the technological routes of the jobs. For many other scheduling problems like certain single machine, flow shop and open shop problems, every permutation consisting of all operations or jobs involved encodes a feasible schedule. Considering the instance GJSP2 of the BJSPT

introduced in Section 2.3.2, a random sequence of all operations $O_{i,j} \in \mathcal{O}$ may result in

$$s^{op} = [O_{2,2}, O_{3,3}, O_{4,2}, O_{3,2}, O_{3,1}, O_{4,1}, O_{5,1}, O_{2,1}, O_{1,2}, O_{5,2}, O_{4,3}, O_{1,1}],$$

where the corresponding machine-based representation is determined by

$$s^{ma} = [[O_{4,2}, O_{3,1}, O_{1,1}], [O_{3,2}, O_{4,1}, O_{2,1}, O_{5,2}, O_{4,3}], [O_{2,2}, O_{3,3}, O_{5,1}, O_{1,2}].$$

Obviously, the permutation s^{op} does not encode a feasible schedule, since several processing sequences are violated by for instance $O_{2,2} \rightarrow O_{2,1}$ concerning job J_2 and $O_{3,3} \rightarrow O_{3,2}$ for job J_3. Furthermore, the earliest starting times of the operations cannot be determined from the machine-based solution s^{ma}, since the technological routes of the jobs forbid the operations $O_{4,2}$, $O_{3,2}$ and $O_{2,2}$ to be processed first on the machines M_1, M_2 and M_3, respectively. Due to the predefined technological route TR_4: $M_2 \rightarrow M_1 \rightarrow M_2$, the processing of job J_4 on machine M_1 can only be performed as the second production step of this job after its processing on machine M_2. Consequently, the machine-based encoding depicts a schedule that is infeasible with regard to the technological routes of the jobs. Bierwirth [18] proposed a simple adaptation scheme for permutation-based representations to generate lists, which satisfy processing sequences and technological routes, as follows: The given operation-based representation is generalized to a job-based permutation

$$s^{job} = [J_2, J_3, J_4, J_3, J_3, J_4, J_5, J_2, J_1, J_5, J_4, J_1],$$

from which a feasible permutation of operations is derived by assigning the processing steps of the jobs according to the jobs appearances in the list. The adapted operation-based representation results in

$$s^{op} = [O_{2,1}, O_{3,1}, O_{4,1}, O_{3,2}, O_{3,3}, O_{4,2}, O_{5,1}, O_{2,2}, O_{1,1}, O_{5,2}, O_{4,3}, O_{1,2}]$$

together with the corresponding machine-based representation

$$s^{ma} = [[O_{3,1}, O_{4,2}, O_{1,1}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{5,2}, O_{4,3}], [O_{3,3}, O_{5,1}, O_{2,2}, O_{1,2}].$$

Since both encodings fulfill the restrictions on the processing sequences and the technological routes of all jobs, the depicted schedule is guaranteed

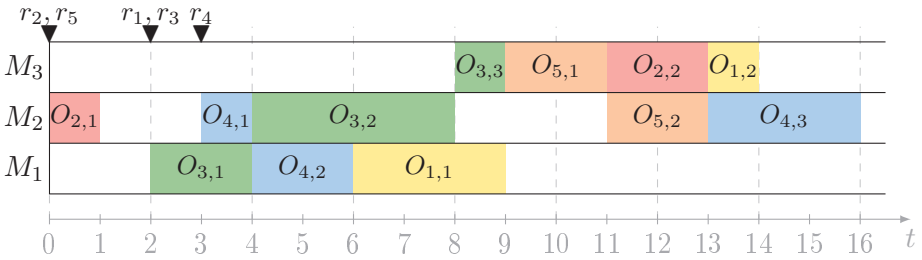


Figure 4.3: A schedule for the instance GJSP2 of the BJSPT violating blocking constraints

feasible for the job shop scheduling problem without blocking constraints. The foregoing explanations show that due to the structure of the job shop problem, a large set of permutations, which express feasible solutions to other scheduling instances such as single machine, flow shop and open shop problems, are infeasible even with regard to basic constraints here, cf. [19].

Figure 4.3 illustrates the implementation of the schedule given above by the machine-based encoding s^{ma}. From the Gantt chart, it can be seen that this schedule is not feasible with regard to blocking constraints, since the jobs J_1, J_2 and J_4 require intermediate storage. Here, a drawback of such simple permutation-based representations is revealed, cf. [19]. The conclusion on the feasibility of the expressed schedule for the BJSPT cannot be drawn based on the machine-based encoding, but requires the calculation of the starting times of the operations. In order to tackle this issue, a formal analysis of the corresponding operation-based representation is beneficial.

Every permutation s^{op} constitutes a total ordering of all operations $O_{i,j} \in \mathcal{O}$ and specifies the corresponding decisions on the disjunctive constraints. Thus, the permutation implies particular starting time relations of pairs of operations resulting from blocking constraints, which need to be reflected by the list indices of the operations. Consequently, the feasibility of a schedule with regard to blocking constraints can be checked by considering an iteratively growing partial permutation of s^{op} that initially consists of the first two operations. For every two operations occurring in the permutation, the resulting blocking constraints are set up in accordance to the given ordering and their implementation in the posterior list in-

dices is examined. Checking the feasibility of the schedule given above, the consideration starts with the partial permutation $s^{op} = [O_{2,1}, O_{3,1}]$ of the operation-based encoding. Since the operations $O_{2,1}$ and $O_{3,1}$ require different machines, the ordering $O_{2,1} \rightarrow O_{3,1}$ does not imply a blocking constraint. In the next iteration, the permutation is extended by the subsequent operation to $s^{op} = [O_{2,1}, O_{3,1}, O_{4,1}]$ and the orderings $O_{2,1} \rightarrow O_{4,1}$ as well as $O_{3,1} \rightarrow O_{4,1}$ need to be investigated. Regarding the former pair, the operations $O_{2,1}$ and $O_{4,1}$ are processed on the same machine M_2 and the sequence implies the blocking constraint $s_{2,2} \leq s_{4,1}$. In order to fulfill this starting time relation, the list indices of the involved operations are restricted to $lidx(O_{2,2}) < lidx(O_{4,1})$. Since $lidx(O_{2,2}) = 8 > 3 = lidx(O_{4,1})$ holds for the given permutation s^{op}, it is classified as infeasible with regard to blocking constraints. Applying this procedure, the feasibility of an encoded schedule for the BJSPT can be checked by means of its operation-based representation.

Furthermore, the observation indicates a possible change in the permutation to obtain a feasible partial schedule. With implementing $lidx(O_{2,2}) < lidx(O_{4,1})$ in the permutation, the blocking constraint can be satisfied. Therefore, the operation $O_{2,2}$ is taken out of the list and reinserted at the position between $O_{3,1}$ and $O_{4,1}$. The resulting partial permutation $s^{op} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}]$ is feasible for the BJSPT, since the intermediate storage of job J_2 illustrated in Figure 4.3 is avoided. As a consequence, the machine-based representation needs to be modified and operation $O_{2,2}$ is shifted to order-position $r = 1$ with $midx(O_{2,2}) = 1$ on machine M_3. The examples in Sections 2.3.2 and 2.4 indicate that there may exist several different adaptations, which resolve an infeasible permutation. For the purpose of displaying all operations available to be set at position three following operation $O_{3,1}$ in the given permutation s^{op}, the alternative graph representation of the problem is considered.

Extending the operation-based representation s^{op} by the artificial source node 0 at the beginning and by the sink node $*$ at the end, it can be interpreted as a Hamiltonian path in a complete directed graph that involves all operations $O_{i,j} \in \mathcal{O}$ together with the source and the sink in the set of nodes. Implementing the total ordering given in s^{op} step by step, a set of operations that are feasible to be added next can be determined for every

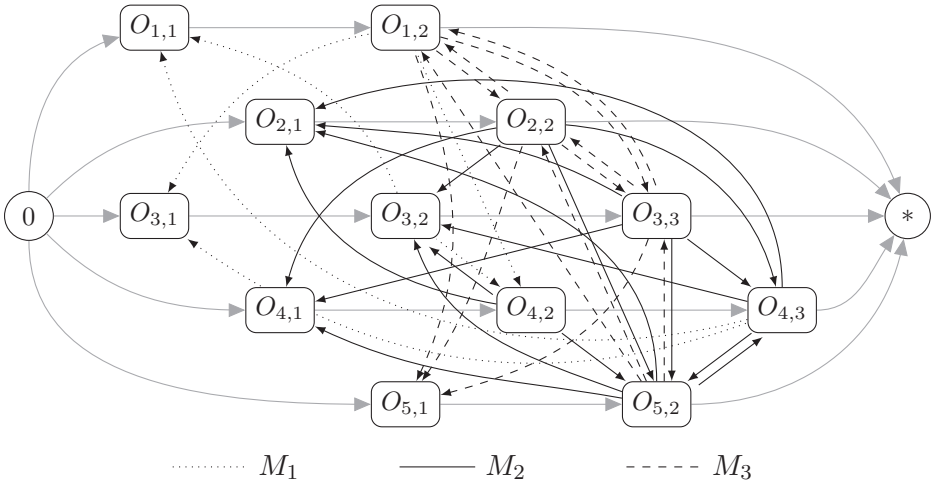


Figure 4.4: Alternative graph representation of the instance GJSP2

partial permutation. Figure 4.4 recalls the alternative graph representation of the instance GJSP2 of the BJSPT introduced in Section 2.3.2. This graph constitutes a subgraph of the complete directed graph with the same set of nodes. For reasons of clarity, the complete graph is not plotted in the figure so that the Hamiltonian path given by the permutation

$$s^{op} = [O_{2,1}, O_{3,1}, O_{4,1}, O_{3,2}, O_{3,3}, O_{4,2}, O_{5,1}, O_{2,2}, O_{1,1}, O_{5,2}, O_{4,3}, O_{1,2}]$$

constructed above might use invisible arcs.

The permutation is implemented in the graph operation-wise and indicated by a complete $0-*-$path, if the encoded schedule is feasible for the BJSPT. The infeasibility of the given ordering is detected by the appearance of an operation $O_{i,j}$ at the subsequently considered position in the list, which does not belong to the set $\mathcal{A}$ of operations available to be visited by the path in the next iteration. If the path visits a node $O_{i,j}$, all undecided disjunctive precedence relations referring to this operation are determined by one of the following cases.

Case 1. If the operation $O_{i,j}$ constitutes the tail of an alternative arc $(O_{i,j}, O_{i',j'})$ with $O_{i',j'} \in \mathcal{O} \setminus \{O_{i,j}\}$, this arc is fixed as the partial ordering $O_{i,j} \rightarrow O_{i',j'}$ in the schedule and its disjunctive substitute arc is deleted from the graph.

Case 2. If the operation $O_{i,j}$ constitutes the head of an alternative arc $(O_{i',j'}, O_{i,j})$ with $O_{i',j'} \in \mathcal{O} \setminus \{O_{i,j}\}$, this arc is deleted from the graph and its disjunctive substitute is fixed as a partial ordering in the schedule.

Consequently, a node $O_{i,j}$ can be visited by the path in the subsequent iteration, $O_{i,j} \in \mathcal{A}$, if the following conditions are fulfilled, cf. [87].

(a) The operation $O_{i,j}$ is the first operation of its job J_i or its predecessor $O_{i,j-1}$ is already part of the path.

(b) The operation $O_{i,j}$ does not constitute the head of a fixed ordering indicated by an arc $(O_{i',j'}, O_{i,j})$ with $O_{i',j'} \in \mathcal{O} \setminus \{O_{i,j}\}$ except the operation $O_{i',j'}$ is already part of the path or the arc is an element of a fixed feasible cycle.

According to the conditions (a) and (b), an empty permutation $s^{op} = [\,]$ can initially be extended by the corresponding first operations of the jobs $J_i \in \mathcal{J}$, here $\mathcal{A} = \{O_{1,1}, O_{2,1}, O_{3,1}, O_{4,1}, O_{5,1}\}$. Figure 4.5 depicts the implementation of the partial permutation $s^{op} = [O_{2,1}]$ as Iteration 1 in the alternative graph. The first arc of the Hamiltonian path $(0, O_{2,1})$, indicated in red, mirrors a processing sequence arc of job J_2. Since the operation $O_{2,1}$ requires machine M_2, four pairs of solid alternative arcs are effected by the sequencing decision $lidx(O_{2,1}) = 1$. Operation $O_{2,1}$ does not constitute the tail of any alternative arc, so that Case 1 is not applied. Following Case 2, the alternative arcs $(O_{4,3}, O_{2,1})$, $(O_{3,3}, O_{2,1})$, $(O_{5,2}, O_{2,1})$ and $(O_{4,2}, O_{2,1})$ are deleted and the disjunctive substitutes $(O_{2,2}, O_{4,3})$, $(O_{2,2}, O_{5,2})$, $(O_{2,2}, O_{3,2})$ and $(O_{2,2}, O_{4,1})$ are fixed, respectively. The determined precedence relations are indicated by blue arcs in the graph. According to the conditions (a) and (b), the set of available operations is adapted to $\mathcal{A} = \{O_{1,1}, O_{2,2}, O_{3,1}, O_{5,1}\}$. Following the permutation s^{op}, the Hamiltonian path is extended by the arc $(O_{2,1}, O_{3,1})$.

The second iteration with the extended sequence $s^{op} = [O_{2,1}, O_{3,1}]$ is shown in Figure 4.6. The operation $O_{3,1}$ does not constitute the tail of any alternative arc, so that only Case 2 is applied to two pairs of dotted arcs referring to machine M_1. Accordingly, the arcs $(O_{1,2}, O_{3,1})$ and $(O_{4,3}, O_{3,1})$ are deleted and their disjunctive substitutes $(O_{3,2}, O_{1,1})$ and $(O_{3,2}, O_{4,2})$ are fixed. The set of determined precedence relations is extended and $\mathcal{A} = \{O_{2,2}, O_{5,1}\}$ appears. Consequently, these are the only two operations

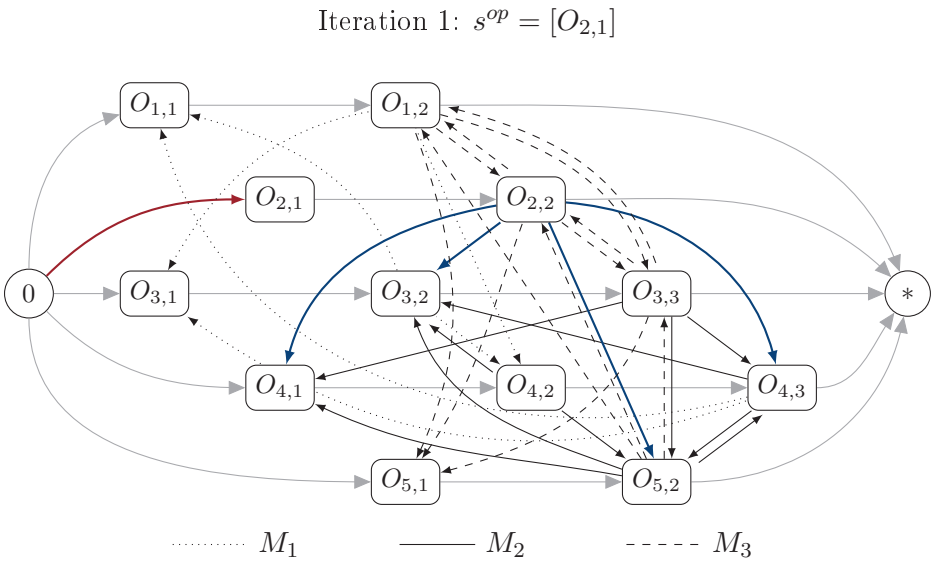


Figure 4.5: Implementing a permutation for the instance GJSP2 of the BJSPT in the alternative graph - Iteration 1

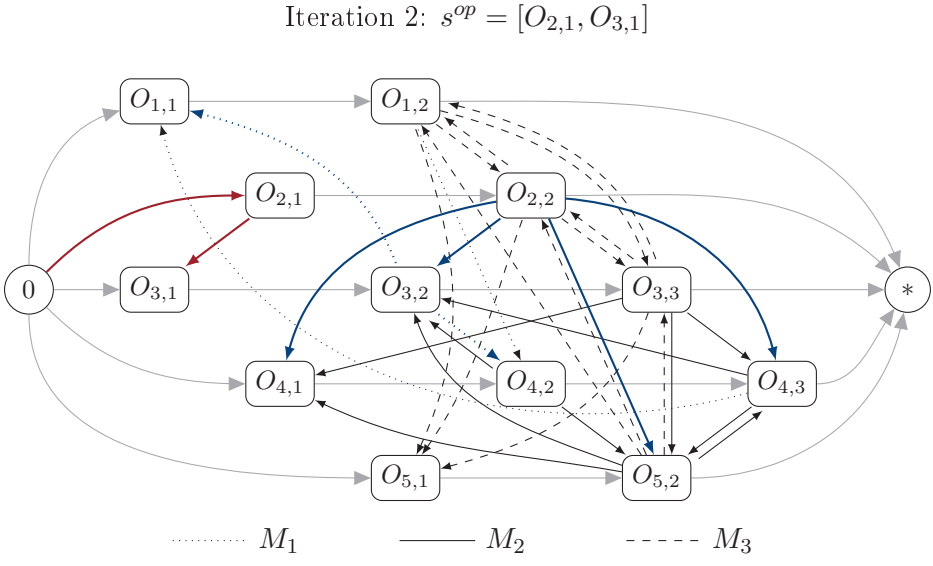


Figure 4.6: Implementing a permutation for the instance GJSP2 of the BJSPT in the alternative graph - Iteration 2

that may pursue the operations $O_{2,1}$ and $O_{3,1}$ at list index three in a feasible permutation. Evidently, the operation being indicated as required due to the blocking constraint, here $lidx(O_{2,2}) < lidx(O_{4,1})$, will always be part of the set of available operations. Therefore, the choice of an operation from $\mathcal{A}$ can be guided by the blocking constraint to stay close to the initially given permutation.

Successively applying the procedure to the entire permutation, a feasible schedule is determined by

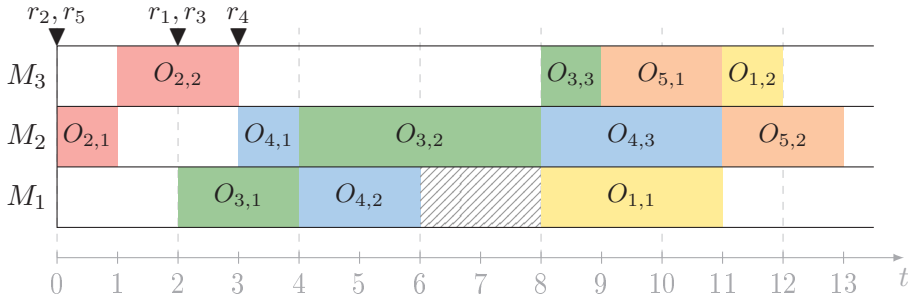
$$s^{op} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,1}, O_{4,3}, O_{1,1}, O_{5,2}, O_{1,2}],$$

$$s^{ma} = [[O_{3,1}, O_{4,2}, O_{1,1}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{4,3}, O_{5,2}], [O_{2,2}, O_{3,3}, O_{5,1}, O_{1,2}]].$$

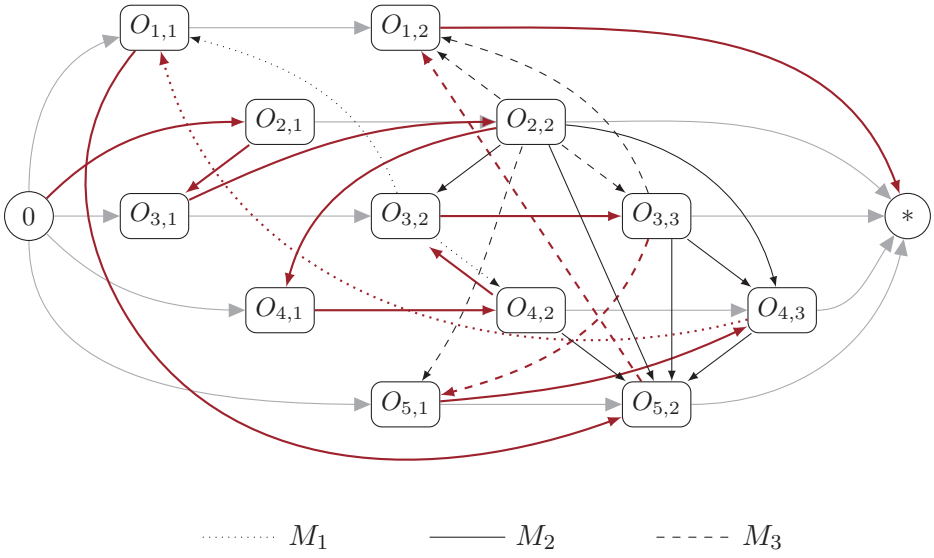
Besides the shift of operation $O_{2,2}$, the operations $O_{4,2}$ and $O_{4,3}$ are reinserted at an earlier position in the list. As a consequence, the operation sequences on the machines M_2 and M_3 are changed during the reordering process. Figure 4.7 shows the Gantt chart and the alternative graph representation of the resulting schedule for the BJSPT. The feasible total ordering given in s^{op} is indicated as the red Hamiltonian $0 - *-$path in the graph at the bottom of the figure. The corresponding selection of alternative arcs is generally plotted in black, where the arcs $(O_{2,2}, O_{4,1})$, $(O_{4,2}, O_{3,2})$, $(O_{4,3}, O_{1,1})$, $(O_{3,3}, O_{5,1})$ and $(O_{5,2}, O_{1,2})$ are overlaid by the red path. The schedule can additionally be classified as feasible for the BJSPT based on the criterion mentioned earlier, since there exists no cycle involving operations requiring different machines among the fixed precedence relations.

Mascis and Pacciarelli [87] observe the expandability of a feasible partial schedule for job shop problems with blocking constraints in a similar way. Based on a weighted alternative graph, the authors show that any feasible partial permutation is expendable. Since the procedure described above starts from an empty permutation and assures feasibility in every iteration, this implies that the set of operations available to be assigned to the next list index can never be empty. Thus, by following the arguments given above, a feasible solution for the BJSPT can always be found based on any permutation.

To summarize, the feasibility of operation-based and machine-based encodings with regard to processing sequences and technological routes of



(a) Gantt chart



(b) Alternative graph representation with the Hamiltonian path

Figure 4.7: A repaired schedule for the instance GJSP2 of the BJSPT

the jobs can easily be assured. On the contrary, both permutation-based representations may depict a schedule, which is infeasible with regard to blocking constraints. The operation-based encoding is required to detect the infeasibility of a schedule for the BJSPT.

Obviously, the incorporation of blocking constraints in a job shop problem leads to even more infeasible permutations that appear during a heuristic search process. There exist four main strategies on how to deal with infeasible solutions, namely exclusion, penalization, repair and translation, cf. [5]. Since initial computational experiments have shown that the majority of generated permutations comes out to be infeasible with regard to at least one class of constraints and in line with the existing literature, cf. [29], [89] and [64], a repair strategy is implemented in this thesis. Therefore, the arguments outlined above form an excellent foundation of the repair procedure derived in the subsequent section.

4.5 The Basic Repair Technique

The proposed repairing strategy relies on the stepwise consideration of an arbitrary permutation of all operations $O_{i,j} \in \mathcal{O}$, which represents a feasible schedule for the job shop problem without blocking constraints. The method constructs the operation-based representation of a feasible schedule for the BJSPT while treating the given sequence as a template. The starting times $s_{i,j}$ of the operations $O_{i,j} \in \mathcal{O}$ are directly determined with the operation sequences on the machines, so that the resulting schedule can easily be evaluated according to any objective function. In resolving the BJSPT by a simple SA metaheuristic, a brief description of this repairing strategy is given in [79]. In the following, the initially existing infeasible permutation is denoted by $perm$, where the finally determined feasible encoding is labeled by s^{op}. The entire method proposed here to construct the representation of a feasible schedule from any given permutation, which accounts for processing sequences and technological routes of all jobs $J_i \in \mathcal{J}$, is denoted as the *Basic Repair Technique (BRT)*. A schematic outline of the BRT is presented in Figure 4.8.

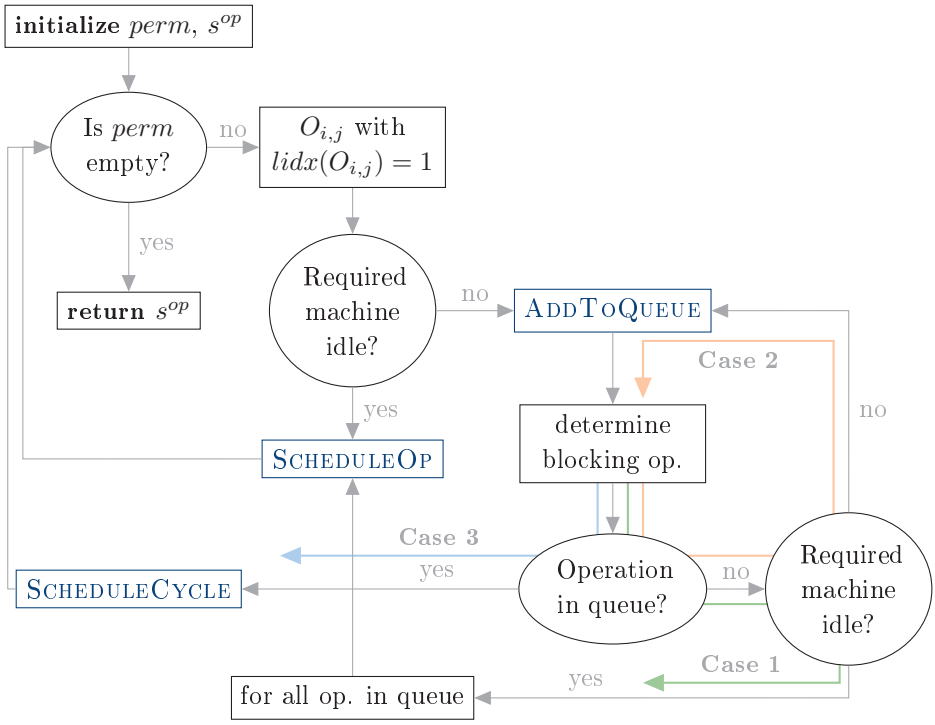


Figure 4.8: Schematic outline of the Basic Repair Technique (BRT)

The process is initialized by a total ordering in $perm$ with the list indices $lidx(O_{i,j})$ for all operations $O_{i,j} \in \mathcal{O}$ and a blank list s^{op}, where the currently considered list index for an operation to be added is denoted as $lidx'(*)$ starting with the first position. While the permutation $perm$ is not empty, the operation $O_{i,j}$ featuring the first list index with $lidx(O_{i,j}) = 1$ is chosen to be assigned to the currently considered list index $lidx'(*)$ in the feasible encoding s^{op}. If the required machine is idle, there is no unsatisfied blocking constraint related to operation $O_{i,j}$ and the operation can be added to the feasible sequence at position $lidx'(*)$. The function ScheduleOp schedules the operation $O_{i,j}$, whereby this subroutine involves

- the expansion of the list s^{op} by operation $O_{i,j}$,
- the increase of the currently considered list index $lidx'(*) \leftarrow lidx'(*) + 1$
- the deletion of operation $O_{i,j}$ from the permutation $perm$,
- the determination and storing of the starting time $s_{i,j}$ of operation $O_{i,j}$ and
- the branding of the required machine as blocked, provided that a job successor $O_{i,j+1}$ exists.

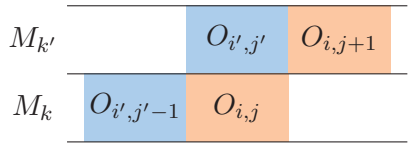
In case that the required machine is not idle, there exists an unsatisfied blocking constraint related to the currently treated operation $O_{i,j}$. As a general example, let the machine be blocked by an operation $O_{i',j'-1}$ and let the permutation determine the ordering $O_{i',j'-1} \rightarrow O_{i,j}$ to be implemented. This causes the starting time relation $s_{i',j'} \leq s_{i,j}$ for the operation $O_{i',j'}$, which is so far assigned to a posterior position in the permutation $perm$. Since it is not clear, whether the operations $O_{i,j}$ and $O_{i',j'}$ are involved in a swap, the required list index relation is generally introduced by $lidx(O_{i',j'}) \leq lidx(O_{i,j})$ and the operation $O_{i,j}$ cannot be scheduled before operation $O_{i',j'}$ is scheduled. The function AddToQueue adds $O_{i,j}$ to the queue Q and $O_{i',j'}$ constitutes the blocking operation. At this point, the procedure may continue according to one out of three possible cases, which are indicated in Figure 4.8 by a green, an orange and a blue arrow, respectively.

Case 1. If operation $O_{i',j'} \notin Q$ and its required machine is idle, the current blocking operation $O_{i',j'}$ and all operations in the queue are scheduled by the subroutine ScheduleOp according to a last in first out strategy.

Case 2. If operation $O_{i',j'} \notin Q$ and its required machine is not idle, there exists an unsatisfied blocking constraint $lidx(O_{i'',j''}) \leq lidx(O_{i',j'})$ related to the considered operation $O_{i',j'}$ and another operation $O_{i'',j''}$ at a posterior position in $perm$. The queue Q is expanded by operation $O_{i',j'}$ by the function AddToQueue and operation $O_{i'',j''}$ is set to be the blocking operation. This loop is repeatedly executed until Case 1 or Case 3 come to pass. Note that the queue Q can include at most m operations, since every newly detected blocking operation $O_{i'',j''}$ needs to require a machine different from the machines of the other operations in the queue for the procedure to expand the list Q. Once there appears a blocking operation $O_{i'',j''}$ requiring the same machine as an operation in the queue Q, $O_{i'',j''} \in Q$ is implied and the procedure continues according to Case 3.

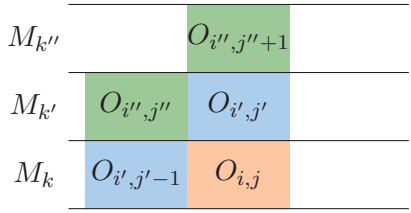
Case 3. If operation $O_{i',j'} \in Q$, a cycle is detected, which is implemented as a swap in the schedule. The function ScheduleCycle determines all operations involved in the swap and forms a swap group $W \subseteq Q$. The operations in the swap group W are pairwise connected by list index relations such as $lidx(O_{i',j'}) \leq lidx(O_{i,j})$, which can only be fulfilled simultaneously by introducing a shared list index in s^{op}. Thus, the operations in W are scheduled as performed by ScheduleOp featuring equal starting times and $lidx'(*)$ as a common list index. If $W \subset Q$ holds and the queue Q involves operations, which do not take part in the swap, these operations remain in the permutation $perm$ keeping their initial ordering.

Figure 4.9 illustrates the three cases by small general examples involving two or three jobs J_i, $J_{i'}$ and, if necessary, $J_{i''}$ in $\mathcal{J}$ and two to three machines M_k, $M_{k'}$ (and $M_{k''}$) in $\mathcal{M}$. According to the infeasible permutations $perm_1$, $perm_2$ and $perm_3$, the operation $O_{i',j'-1}$ of the blue job $J_{i'}$ blocks the machine M_k, when the orange operation $O_{i,j}$ is considered to be scheduled next. Thus, in all three cases the blue operation $O_{i',j'}$ constitutes the first blocking operation with $lidx'(O_{i',j'}) \leq lidx'(O_{i,j})$ and requires to be added to the feasible operation-based representation prior to operation $O_{i,j}$.



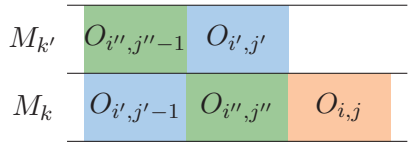
$$perm_1 = [O_{i',j'-1}, O_{i,j}, O_{i,j+1}, O_{i',j'}] \qquad Q_1 = [O_{i,j}]$$
$$s_1^{op} = [O_{i',j'-1}, O_{i',j'}, O_{i,j}, O_{i,j+1}]$$

(a) Case 1 with $| Q | = 1$



$$perm_2 = [O_{i'',j''}, O_{i',j'-1}, O_{i,j}, O_{i',j'}, O_{i'',j''+1}] \qquad Q_2 = [O_{i,j}, O_{i',j'}]$$
$$s_2^{op} = [O_{i'',j''}, O_{i',j'-1}, O_{i'',j''+1}, O_{i',j'}, O_{i,j}]$$

(b) Case 2 → Case 1 with $| Q | > 1$



$$perm_3 = [O_{i',j'-1}, O_{i'',j''-1}, O_{i,j}, O_{i',j'}, O_{i'',j''}] \qquad Q_3 = [O_{i,j}, O_{i',j'}, O_{i'',j''}],$$
$$s_3^{op} = [O_{i',j'-1}, O_{i'',j''-1}, (O_{i',j'}, O_{i'',j''}), O_{i,j}] \quad W = \{O_{i',j'}, O_{i'',j''}\}$$

(c) Case 2 → Case 3 with $W \subset Q$

Figure 4.9: Exemplary resolutions of an infeasible permutation for the BJSPT

In part (a) of Figure 4.9, the machine $M_{k'}$, on which the blocking operation $O_{i',j'}$ is to be processed, is idle. Hence, the blocking operation can directly be scheduled at $lidx'(*) = 2$ in the list s_1^{op} according to Case 1 and the operation $O_{i,j}$ stored in queue Q_1 is assigned to the subsequent list index. Considering part (b) of Figure 4.9, the machine $M_{k'}$ is blocked by operation $O_{i'',j''}$ of the green job $J_{i''}$. Thus, operation $O_{i',j'}$ is following operation $O_{i,j}$ in the queue Q_2 and operation $O_{i'',j''+1}$ is set to be the blocking operation during the execution of the loop described by Case 2. Since the machine $M_{k''}$, on which the new blocking operation $O_{i'',j''+1}$ is supposed to be processed, is idle, the function ScheduleOp can be called according to Case 1. The operations $O_{i'',j''+1}$, $O_{i',j'}$ and $O_{i,j}$ are scheduled in the feasible ordering s_2^{op} starting from the one considered last. Part (c) of Figure 4.9 depicts the execution of Case 3. The operation $O_{i,j}$ is stored in the queue Q_3, while the machine $M_{k'}$ is blocked by the green operation $O_{i'',j''-1}$. Therefore, operation $O_{i',j'}$ is likewise assigned to the queue Q_3 and the green operation $O_{i'',j''}$ requiring machine M'_k as well constitutes the new blocking operation. After one more operation of the Case 2 loop, the next blocking operation is determined by $O_{i',j'}$, which is already contained in the queue Q_3. Thus, a swap group $W = \{O_{i',j'}, O_{i'',j''}\}$ is detected by the function ScheduleCycle and both operations are assigned to the currently considered list index $lidx'(*) = 3$ in the feasible permutation s_3^{op}. Since $W \subset Q$ holds, the remaining operation $O_{i,j}$ is scheduled posterior to the swap group during the following iteration of the procedure.

These steps are repeatedly executed until there is no operation left in the permutation *perm* and all operations are consequently sequenced to build an operation-based representation s^{op} of a feasible schedule for the BJSPT. Algorithm 2 technically describes the strategies explained above. The execution of the **while**-loop in the lines 4 to 26 of the algorithm characterizes one iteration of the BRT. The correctness and the termination of the BRT are shown by the following proposition.

Proposition 4.1. *The BRT terminates and returns a permutation s^{op} encoding a feasible schedule for the BJSPT.*

Algorithm 2 Basic Repair Technique (BRT)

Input: permutation (list) of operations *perm*
[feasible with regard to processing sequences and technological routes of $J_i \in \mathcal{J}$]

Output: operation-based representation s^{op} of a feasible schedule
[feasible for the BJSPT]

initialize s^{op}, Q and W as empty lists, ST as an empty starting times dictionary
initialize MaS as a dictionary including the status of all machines $M_k \in \mathcal{M}$ (= idle)

$lidx'(*) \leftarrow 1$
while *perm* is not empty **do**
    $O_{i,j} \leftarrow$ operation with $lidx(O_{i,j}) = 1$ in *perm*
    if $Machine(O_{i,j})$ is idle **then**
        ScheduleOp($O_{i,j}$, *perm*, s^{op}, ST, $lidx'(*)$, MaS)
    else
        AddToQueue(Q, $O_{i,j}$)
        $O_{i',j'} \leftarrow$ blocking operation requiring $lidx'(O_{i',j'}) \leq lidx'(O_{i,j})$
        while $O_{i',j'} \notin Q$ **do**
            if $Machine(O_{i',j'})$ is not idle **then**
                AddToQueue(Q, $O_{i',j'}$) [**Case 2**]
                $O_{i',j'} \leftarrow$ blocking operation $O_{i'',j''}$ requiring $lidx'(O_{i'',j''}) \leq lidx'(O_{i',j'})$
            else
                ScheduleOp($O_{i',j'}$, *perm*, s^{op}, ST, $lidx'(*)$, MaS) [**Case 1**]
                for $O_{i,j} \in Q$ **do**
                    ScheduleOp($O_{i,j}$, *perm*, s^{op}, ST, $lidx'(*)$, MaS)
                end for
                break
            end if
        end while
    end if
    if Q is not empty **then**
        ScheduleCycle(Q, *perm*, s^{op}, ST, $lidx'(*)$, MaS) [**Case 3**]
    end if
end while
return s^{op} (**return** ST)

Proof. It has to be shown that

(1) the resulting permutation s^{op} is feasible with regard to the processing sequences and the technological routes of all jobs $J_i \in \mathcal{J}$,

(2) the resulting permutation s^{op} is feasible with regard to blocking constraints and

(3) every operation $O_{i,j} \in \mathcal{O}$ is assigned to a position in the feasible permutation s^{op} exactly once.

The BRT takes an arbitrary permutation $perm$ with the list indices $lidx(O_{i,j})$ for all $O_{i,j} \in \mathcal{O}$, which is feasible with regard to the processing sequences and the technological routes of all jobs $J_i \in \mathcal{J}$, as its input. Let the ordering $lidx(O_{i',j'-1}) < lidx(O_{i,j}) < lidx(O_{i',j'})$, where both operations $O_{i',j'-1}$ and $O_{i,j}$ are processed on the same machine M_k, cause the blocking constraint $s_{i',j'} \leq s_{i,j}$ requiring $lidx(O_{i',j'}) \leq lidx(O_{i,j})$. This unsatisfied blocking constraint is detected while operation $O_{i',j'-1}$ is already scheduled in the feasible partial permutation s^{op} and $lidx'(*) = lidx(O_{i,j})$ holds for the currently considered list index for the next operation to be assigned to. Thus, the BRT shifts required operation(s), here only operation $O_{i',j'}$, to the position $lidx'(*) = lidx(O_{i,j}) > lidx(O_{i',j'-1})$ and will never affect list indices prior or equal to $lidx'(O_{i',j'-1})$. Hence, a given feasible ordering accounting for processing sequences and technological routes, such as $lidx(O_{i',j'-1}) < lidx(O_{i',j'})$, can never be violated by changes in the operation sequences made to fulfill blocking constraints. (1) is true.

The permutation s^{op} is constructed by the stepwise expansion of an empty list. Every time an operation $O_{i,j}$ is assigned to the currently regarded list index $lidx'(*)$, unsatisfied blocking constraints that require $lidx'(O_{i',j'}) \leq lidx'(O_{i,j})$ for an arbitrary operation $O_{i',j'}$ placed at a posterior position in the permutation $perm$, are detected and fulfilled. Accordingly assigning the operation $O_{i',j'}$ to the list index $lidx'(*)$ in s^{op} prior to its initially given index $lidx(O_{i',j'})$ in $perm$ may implement a change in the operation sequence on the concerned machine. This may only cause new blocking constraints referring to the positions of the job successor $O_{i',j'+1}$ and the machine successor of operation $O_{i',j'}$. Due to feasible processing sequences and technological route orderings, both affected operations cannot be part of the current partial permutation s^{op} and unsatisfied blocking constraints

do only arise in the remainder of the permutation $perm$. Since this is still to be considered by the BRT, it is always assured that the existing partial permutation s^{op} is feasible with regard to blocking constraints in every iteration. Resulting from the stepwise construction pattern of s^{op}, (2) is shown.

In every iteration of the BRT, the permutation s^{op} is expanded by at least one operation $O_{i,j} \in \mathcal{O}$, where $\mathcal{O}$ is a finite set with $|\mathcal{O}| = n_{op}$. The adding of operations to the feasible permutation s^{op} follows the ordering given in the initial list $perm$ starting from the first position. Since the assignment of an operation $O_{i,j}$ to the list index $lidx'(*)$ in s^{op} may only affect constraints that relate succeeding operations at posterior positions in the initial list $perm$, the necessity of a repeated considering of an operation can never occur, once it is added to the feasible ordering s^{op}. Therefore, (3) is true. □

Remark. In order to specify the number of iterations of the BRT required to obtain the operation-based representation s^{op} of a feasible schedule for the BJSPT, the transformation of the initially given permutation $perm$, especially its length, is observed. In every iteration, the operation $O_{i,j}$ with $lidx(O_{i,j}) = 1$ in $perm$ is added to the feasible ordering s^{op} and deleted from the list $perm$. If the machine required by operation $O_{i,j}$ is not idle, there will be further operations assigned to positions in the permutation s^{op}. The number of operations being added to s^{op} can be determined by

- $|Q| + 1 = 2$ with $|Q| = 1$ according to Case 1,
- $|Q| + 1 > 2$ with $|Q| > 1$ according to Case 2 followed by Case 1 and
- $|Q| \geq 2$ with $W \subseteq Q$ and $|W| \geq 2$ according to Case 2 followed by Case 3.

Thus, the length of the list $perm$ decreases by at least one element in every iteration of the BRT. If the given ordering in $perm$ by chance constitutes a feasible schedule for the BJSPT, the condition in line 6 of Algorithm 2 will be true in every execution of the **while**-loop and the BRT will terminate after exactly n_{op} iterations with $s^{op} = perm$. If the given ordering in $perm$ encodes an infeasible schedule for the BJSPT, the number of iterations

of the BRT will be strictly less than n_{op}, whereby the amount of steps executed within the iterations will increase, significantly.

Proposition 4.1 shows that the BRT terminates after a finite number of iterations. In the following, the runtime of the BRT according to the implementation of Algorithm 2 in Python 3 is examined in detail. First, the execution time of the subroutines ScheduleOp, AddToQueue and ScheduleCycle is determined based on the instance input parameters. For the entire runtime observation, it is assumed that Python 3 built-in operators on lists and dictionaries, such as detecting elements by their indices or keys, inserting and removing elements, appending a list as well as returning the list index of an element, can be executed in constant time $O(const)$.

- The tasks involved in the function ScheduleOp($O_{i,j}$, $perm$, s^{op}, ST, $lidx'(*)$, MaS) are listed above. Expanding the list s^{op}, redeclaring the variable $lidx'(*)$ and deleting an element from list $perm$ can be done in constant time by assumption. The determination of the starting time of the scheduled operation involves the detection of two earliest possible starting times from two lists and their comparison. The change of the status of a machine incorporates a dictionary request to check the existence of a job successor and the renewal of a list entry. Thus, the entire function ScheduleOp takes $O(const)$ operations.
- AddToQueue(Q, $O_{i,j}$) is a small subroutine, which expands the list Q by an element. This is done in $O(const)$ steps by assumption.
- The function ScheduleCycle(Q, $perm$, s^{op}, ST, $lidx'(*)$, MaS) involves the detection of the cycle in the queue Q, its transformation into the swap group W and the call of the function ScheduleOp for all operations in the list W. The list Q may involve at most m operations, for each of which a decision about its incorporation in the swap group W is made in constant time. As a result, $|W| \leq m$ holds. A common starting time for all elements in the swap group W is determined by first executing a dictionary request for each of the involved operations and second applying a built-in maximum operator. According to the resulting starting time, the operations

in the list W are scheduled one after another using the function ScheduleOp. Thus, the subroutine ScheduleCycle operates in $O(m)$ time.

The initialization of the BRT, see the lines 1 and 2 of Algorithm 2, is executed in $O(m)$. Thereafter, the **while**-loop involving the lines 4 to 26 is repeated at most n_{op} times. If the idleness condition in line 6, which relies on a dictionary request $Machine(O_{i,j})$, is true, the operation $O_{i,j}$ will be scheduled by the function ScheduleOp. Thus, the lines 6 to 22 are operated in $O(const)$ and the function in line 24 is not called, since Q constitutes an empty list. If the machine is not idle and the condition in line 6 is false, the queue Q is expanded and a blocking operation is detected by a dictionary request in constant time (see the lines 9 and 10). Subsequently, the **while**-loop in the lines 11 to 21 operates in $O(m)$. The condition in line 12 may be true for at most m consecutive blocking operations, resulting in a repeated execution of the **while**-loop. Alternatively, the **else** condition may come to pass, with which at most m operations are scheduled sequentially and the **while**-loop is immediately stopped. In the worst case, $m-1$ operations are put into the list Q by AddToQueue in line 13, the m-th blocking operation $O_{i',j'}$ features an idle required machine and, consequently, m operations are scheduled by ScheduleOp during the execution of **else**. Considering the case of the **while**-loop in line 11 being left due to $O_{i',j'} \in Q$, the condition in line 23 is automatically true and the function ScheduleCycle is executed in $O(m)$.

Taking the assumption that $m \leq n_{op}$ holds, the BRT is operated in polynomial time

$$O(m + n_{op} \cdot (m + m)) \quad = \quad O(n_{op} \cdot m).$$

4.6 Priority Rules as Techniques to Generate Initial Solutions

Priority rules constitute well-known generic solution techniques for all types of scheduling problems. In line with the literature mentioned in Section 4.1, a set of such decision support mechanisms is applied to quickly

generate schedules as the starting points for metaheuristic search procedures. Since several studies indicate a dependence of the performance of priority rules on the specific optimization criterion and the shop load level, cf. for instance [92] and [106], there are five basic rules chosen for parallel implementation together with a random list scheduling mechanism (RANDOM) in this thesis. All of these dispatching rules are proposed for or at least tested on job shop scheduling problems with tardiness-based objective functions and show promising features, cf. [9], [59], [92], [106] and [123].

By applying a priority rule, decisions about which operation to insert next into the schedule are made repeatedly. Since the job shop scheduling problem constitutes a constraint optimization problem, the priority rule operates on a steadily updated set of available operations $\mathcal{A}$, where availability is defined by satisfied processing sequences and technological routes of the jobs. This process refers to the determination of an ordering of the operations involved in the problem similar to the operation-based representation of a schedule described above. Therefore, the feasibility issues of permutations caused by the incorporation of blocking constraints need to be taken into account while solving the BJSPT by a priority rule. Since the existing dispatching techniques do not account for blocking constraints and accordingly generate infeasible permutations with a high probability, the BRT proposed in Section 4.5 is applied subsequently. Figure 4.10 illustrates the priority rule-based procedure to generate a feasible schedule for the BJSPT. The set of rules highlighted in blue in the left upper corner of the figure is specified in the following. Note that according to all of these dispatching rules except ATC, the operation $O_{i,j} \in \mathcal{A}$ with the *minimal priority measure* is chosen to be inserted next. Let t denote the point in time, at which the processing of operation $O_{i,j}$ can be started earliest.

The probably most popular priority rule is the *Shortest Processing Time Rule (SPT)*, which schedules jobs or operations according to increasing processing times, cf. among others [69], [92] and [104]. This strategy is motivated by the idea of reducing the queue of unscheduled operations as fast as possible.

$$\text{SPT:} \quad prio(O_{i,j}) = p_{i,j}$$

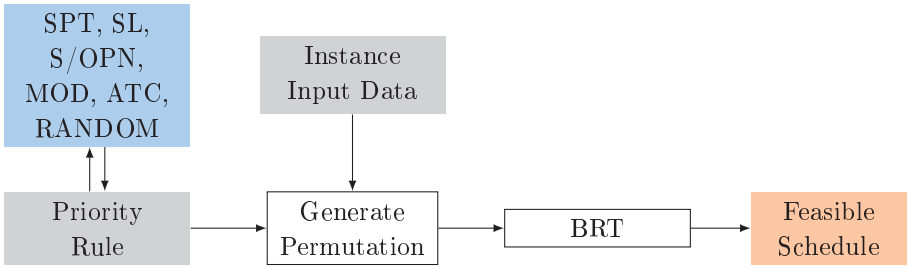


Figure 4.10: The application of priority rules to generate a feasible schedule for the BJSPT

The *Slack Rule (SL)* gives priority to operations of jobs, which are most urgent according to their remaining slack time, cf. among others [69] and [59].

$$\text{SL:} \quad prio(O_{i,j}) = d_i - t - \sum_{q=j+1}^{n_i} p_{i,q}.$$

Following the *Slack per Operation Rule (S/OPN)*, the remaining slack time of the corresponding job J_i is equally split among all processing steps succeeding operation $O_{i,j}$, cf. among others [59], [69] and [92]. Priority is given to the operation featuring the minimal remaining slack time per successor operation.

$$\text{S/OPN:} \quad prio(O_{i,j}) = \frac{d_i - t - \sum_{q=j+1}^{n_i} p_{i,q}}{n_i - j}$$

The *Modified Operation Due Date Rule (MOD)* is proposed in [9] and also denoted by ODD in later publications, cf. [8] and [69]. Prior to the application of the priority rule, operation due dates $d_{i,j}$ are determined based on a desired equal allocation of the allowance $d_i - r_i$ of job J_i over all n_i operations.

$$\text{MOD:} \quad prio(O_{i,j}) = \max\{d_{i,j},\ t + p_{i,j}\}.$$

Accounting for more complex aspects of the problem, the *Apparent Tardiness Cost Rule (ATC)* is introduced in [123] and proves to be robust against the choice of the objective function and the shop load level. Since

the priority rules are intended to generate an initial solution for a consecutively applied metaheuristic procedure, the ATC is slightly simplified by setting the involved waiting time estimates of the operations to zero. Let the considered operation $O_{i,j} \in \mathcal{A}$ require machine M_k and let $\tau = 3$ as the look-ahead parameter according to the findings in [123]. $\bar{p}_k$ denotes the average processing time of the operations pending to be processed on machine M_k. The operation with the maximal measure is chosen to be inserted into the schedule based on the following priorities.

$$\text{ATC:} \quad prio(O_{i,j}) = \frac{1}{p_{i,j}} \cdot \exp\left(- \max\left\{ \frac{d_i - t - \sum_{q=j}^{n_i} p_{i,q}}{\tau \cdot \bar{p}_k} \ , \ 0 \right\} \right)$$

Since these priority rules have not been tested comparatively on job shop instances with blocking constraints, the permutation generating mechanisms are implemented in parallel. Furthermore, all of the presented rules still incorporate a randomized component. In case that several operations $O_{i,j} \in \mathcal{A}$ feature the same minimal (or maximal) value $prio(O_{i,j})$, the choice of an operation is done arbitrarily among them.

4.7 Neighborhood Structures and their Characteristics

Three neighborhoods are proposed for the BJSPT and discussed with regard to occurring feasibility issues, connectivity and further characteristics. To highlight the foundation and the motivation of the choice of the proposed transition schemes, existing neighborhood structures for generic job shop scheduling problems with and without blocking constraints are briefly summarized in the subsequent Section 4.7.1. The herein applied basic operators and their implementation in different representations of a schedule are described in Section 4.7.2. In order to overcome occurring feasibility issues, it is shown in Section 4.7.3 that the BRT proposed in Section 4.5 requires an extensive adaptation to be applicable in the construction of feasible neighboring schedules. In Section 4.7.4, the *Adjacent Pairwise Interchange (API) neighborhood*, which operates on the machine-based representation of the schedule and is strictly controllable by the chosen

pair of adjacent operations on a certain machine, is defined together with the *Tardy Adjacent Pairwise Interchange (TAPI) neighborhood*, which is derived for the purpose of operating on a reduced search space. Both API-based neighborhoods and the proposed procedure to regain feasibility can be found briefly described in [79]. In order to add a randomized component to the search method, which features a significantly different structure compared to the API, the *Tardy Job (TJ) neighborhood* is introduced in Section 4.7.5. As one of the main characteristics, the connectivity of the neighborhoods is examined in Section 4.7.6. Finally, further properties of the transition structures are empirically analyzed in Section 4.7.7 to shed light onto the complexion of the underlying search space for the metaheuristics and give evidence for the effectiveness of specific parts of the construction schemes.

4.7.1 Permutation-Based Neighborhoods for Job Shop Scheduling Problems

Since the job shop scheduling problem constitutes a special type of sequencing problems, there exist four generic operators applicable to its permutation-based solutions, cf. for instance [5] and [127]. Considering an arbitrary permutation of operations

$$perm_1 = [O_{1,1}, O_{3,1}, O_{4,2}, O_{2,3}, O_{5,4}],$$

the following four neighboring sequences can exemplarily be determined.

- $perm_2 = [O_{1,1}, \underline{O_{4,2}}, \underline{O_{3,1}}, O_{2,3}, O_{5,4}]$ is constructed by applying a swap of the two adjacent operations $O_{3,1}$ and $O_{4,2}$ to the list, while the positions of the remaining operations are kept and there appears exactly one change in the total ordering. This transition scheme is denoted as an *adjacent pairwise interchange (API)* in this thesis, cf. [128].
- $perm_3 = [O_{1,1}, \underline{O_{2,3}}, O_{4,2}, \underline{O_{3,1}}, O_{5,4}]$ features a general *interchange* of the two operations $O_{3,1}$ and $O_{2,3}$, where the positions and the orderings of the remaining operations among each other are kept.

- $perm_4 = [O_{1,1}, \underline{O_{5,4}}, O_{3,1}, O_{4,2}, O_{2,3}]$ is obtained by *shifting* operation $O_{5,4}$ by three positions to the left, while the orderings of the remaining operations among each other are kept.
- $perm_5 = [O_{4,2}, O_{2,3}, \underline{O_{1,1}, O_{3,1}}, O_{5,4}]$ results from performing a *block shift* of the block of operations $(O_{1,1}, O_{3,1})$ by two positions to the right, whereby the orderings of the remaining operations among each other are kept.

With regard to the BJSPT, each transition can generally be applied to the operation-based as well as to the machine-based representation of a schedule. Due to the redundancy of permutations of operations s^{op}, which is discussed in Section 4.4.1, it is reasonable to operate a neighborhood on machine-based representations. Furthermore, the previously described feasibility issues tend to increase with the amount of change caused by the transition. Therefore, two of the three proposed neighborhoods mainly rely on the API, which constitutes the smallest move by means of the distance measure introduced in Section 4.3. To add a diversifying component to the search, a third shift-based neighborhood is applied to the operation-based representation of the schedule causing a higher distance between neighboring solutions.

Considering the standard $J \mid\mid C_{max}$, a well-known neighborhood is proposed by van Laarhoven, Aarts and Lenstra [122]. Neighboring solutions are derived by reversing a given ordering on the critical path in the disjunctive graph representation of the initial schedule. This corresponds to the implementation of an API of a pair of specific operations on a certain machine, where the choice of the machine and the operations is guided by the optimization criterion. Since this neighborhood is successfully applied to the standard job shop problem, it is tested in a basic and several advanced versions on the problem $J \mid r_i \mid \sum w_i T_i$ in [76]. This study gives evidence to the fact that the modification of the neighborhood according to the considered optimization criterion might be similarly beneficial for tardiness-based objective functions, since the search space is reduced while promising solutions are pursued. Moreover, this idea of reducing the number of neighbors with regard to the optimization criterion is successfully applied to the blocking job shop problem with makespan minimization, cf. [64], [98] and [103]. Therefore, a tardiness- and API-based neighborhood is

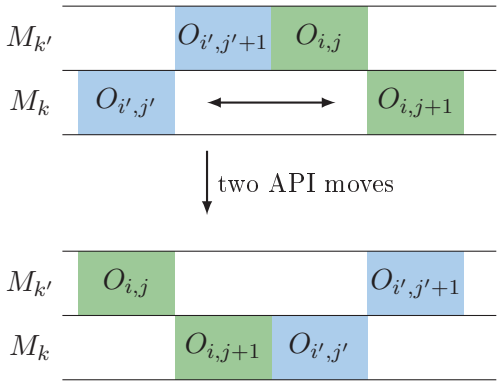
proposed and tested on the blocking job shop problem in comparison to a generic API transition scheme in this thesis. Computational experiments are used to evaluate whether positive effects of a reduced search space on runtime and solution quality shown in the literature are compensated or intensified by the simultaneous incorporation of both, blocking constraints and the minimization of total tardiness.

Furthermore, a job insertion-based neighborhood embedded in a tabu search is proposed by Bürgy in [34] for blocking job shop problems with regular optimization criteria. The transition scheme is guided by the consideration of certain precedence relations, which determine the objective function value of the current schedule. In order to improve the schedule, one of the involved jobs is chosen, excluded and reinserted, whereby the corresponding precedence relation is reverted. Motivated by the promising findings, the idea of objective-guided job reinsertion is applied in a tardiness-based and randomized neighborhood that shifts all operations of a job in the operation-based representation of a schedule. This transition scheme leads to potentially improving neighboring solutions with an arbitrarily large distance from the initial solution. The advantageousness of this property in tackling the BJSPT is additionally examined in the computational study.

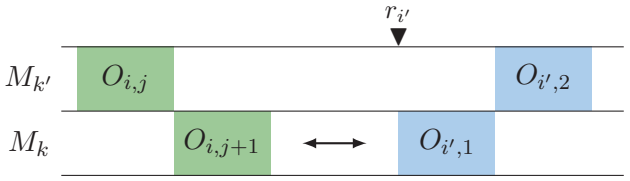
4.7.2 Adjacent Pairwise Interchange-Based Moves and their Transfer

In order to properly describe the API-based transition scheme in the following, the finishing time $f_{i,j}$ of an operation $O_{i,j}$ is introduced with $f_{i,j} = s_{i,j} + p_{i,j} + b_{i,j}$, where $b_{i,j}$ denotes the length of the time period during which operation $O_{i,j}$ blocks the required machine M_k. Note that $f_{i,j} = s_{i,j+1}$ holds for all operations $O_{i,j} \in \mathcal{O}^i, J_i \in \mathcal{J}$ with $j \neq n_i$ and for all operations $O_{i,n_i} \in \mathcal{O}^i, J_i \in \mathcal{J}$, $f_{i,n_i} = s_{i,n_i} + p_{i,n_i}$ is true, since an operation O_{i,n_i+1} does not exist and $b_{i,n_i} = 0$.

Definition 4.1. *An **API move** denotes the interchange of two adjacent operations $O_{i,j}$ and $O_{i',j'}$ of different jobs requiring the same machine $M_k \in \mathcal{M}$ in the machine-based representation of the schedule. Here, adjacency is defined in a strict sense. A pair of operations $O_{i,j}$ and $O_{i',j'}$ is called* **ad-**



(a) Intermediate idle time of a machine caused by a sequence of starting time relations



(b) Intermediate idle time of a machine caused by the release date of a job

Figure 4.11: API moves and the intermediate idle time of a machine

jacent *if* $f_{i,j} = s_{i',j'}$ *holds for an ordering* $O_{i,j} \rightarrow O_{i',j'}$ *on machine* M_k *in schedule* s.

The exclusive consideration of the API moves of pairs of operations without intermediate idle time on the required machine does not limit the set of constructible beneficial schedules. As illustrated in Figure 4.11, an intermediate idle time of a machine is caused either by a sequence of restrictive starting time relations of operations, see part (a), or by the release date of a job, see part (b). Considering the former case, in the machine-based representation of the schedule, there exist two pairs of generally adjacent operations, namely the operations $O_{i',j'}$ and $O_{i,j+1}$ on machine M_k and the operations $O_{i',j'+1}$ and $O_{i,j}$ on machine $M_{k'}$. According to Definition 4.1, an API move can only be applied to the operations $O_{i',j'+1}$ and $O_{i,j}$ on machine $M_{k'}$. Since there exists the sequence of starting time relations $s_{i',j'} < s_{i',j'+1} < s_{i,j} < s_{i,j+1}$, a direct interchange of the operations $O_{i',j'}$ and $O_{i,j+1}$ featuring an intermediate idle time on machine M_k creates a

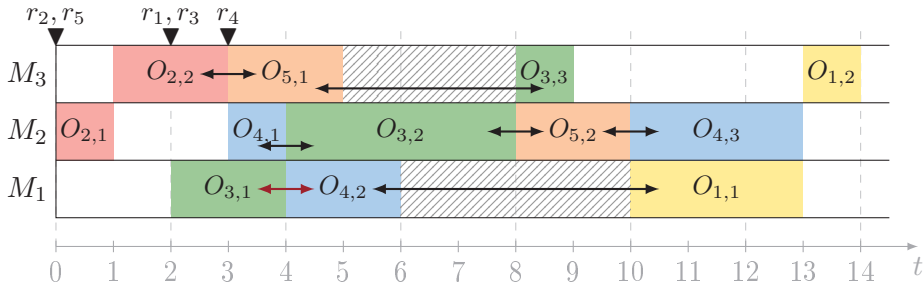


Figure 4.12: Illustration of the set of API moves applicable to a given schedule of the instance GJSP2

cyclic ordering that violates processing sequence constraints. Nonetheless, schedules involving the precedence relation $O_{i,j+1} \rightarrow O_{i',j'}$ on machine M_k can be constructed by applying two consecutive API moves as indicated in Figure 4.11. Part (b) of the figure shows the release date $r_{i'}$ of a job $J_{i'}$ causing an intermediate idle time on machine M_k. In this case, there is no API move performable in the schedule. Thus, there exist schedules involving orderings such as $O_{i',1} \rightarrow O_{i,j+1}$ on machine M_k that are not constructible by applying API moves to the given representation. However, this reduces the search space in an advantageous way. Starting from the existing intermediate idle time, an ordering of operations of the green job J_i posterior to the operations of the blue job $J_{i'}$ can never improve the tardiness-based objective. Implementing Definition 4.1, the seven API moves which are applicable to the given schedule for the instance GJSP2 are indicated with bidirectional arrows in Figure 4.12.

By applying an API move to the machine-based representation of a schedule s, a new schedule s', which is different from s, is derived. Since a change exclusively made in the machine-based representation of the schedule s might lead to an infeasible schedule s', the operation-based representation $s^{op'}$ involving the given API move is required. The priority-based transformation scheme proposed in Section 4.4.1 can be used to generate the permutation $s^{op'}$ of the new schedule, which only differs from the permutation s^{op} of the initial schedule s in the API move. Consider the

schedule s given for GJSP2 in Figure 4.12 with its operation-based and machine-based representations

$$s^{op} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}],$$

$$s^{ma} = [[O_{3,1}, O_{4,2}, O_{1,1}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{5,2}, O_{4,3}], [O_{2,2}, O_{5,1}, O_{3,3}, O_{1,2}]].$$

As an example, the API move of the operations $O_{3,1}$ and $O_{4,2}$, which is indicated in red in Figure 4.12, is applied. Thus, a potentially infeasible machine-based representation of the new schedule s' results in

$$s^{ma'} = [[\underline{O_{4,2}}, \underline{O_{3,1}}, O_{1,1}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{5,2}, O_{4,3}], [O_{2,2}, O_{5,1}, O_{3,3}, O_{1,2}]].$$

Since the transformation scheme aims in keeping the ordering of the remaining operations, the API move is transferred to the permutation $s^{op'}$ as a shift of one of the interchanged operations $O_{3,1}$ or $O_{4,2}$. For a general operation-based representation of a schedule, it is not guaranteed that the pair of operations features adjacent positions in the permutation. For the considered example, $lidx(O_{3,1}) = 2$ holds, while $lidx(O_{4,2}) = 6$ is true. Consequently, there exist two possibilities to implement the API move, namely shifting operation $O_{3,1}$ to the right or alternatively shifting operation $O_{4,2}$ to the left, cf. [13]. Executing the transformation scheme with s^{op} and $s^{ma'}$ as given in Algorithm 1, a *left shift transformation* is realized and yields

$$s_l^{op'} = [O_{2,1}, \underline{O_{4,1}}, \underline{O_{4,2}}, \underline{O_{3,1}}, O_{2,2}, O_{5,1}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}].$$

Since the transformation scheme accounts for the processing sequences of all jobs, the precedence relation $O_{4,1} \rightarrow O_{4,2}$ is automatically kept. Algorithm 1 can easily be adapted to operate reversely from the last to the first operation of a permutation. This leads to the implementation of a *right shift transformation* and the algorithm returns the operation-based representation

$$s_r^{op'} = [O_{2,1}, O_{2,2}, O_{4,1}, O_{5,1}, \underline{O_{4,2}}, \underline{O_{3,1}}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}].$$

Both encodings might be infeasible with regard to blocking constraints and can be transferred to feasible sequences by the BRT proposed in Section 4.5.

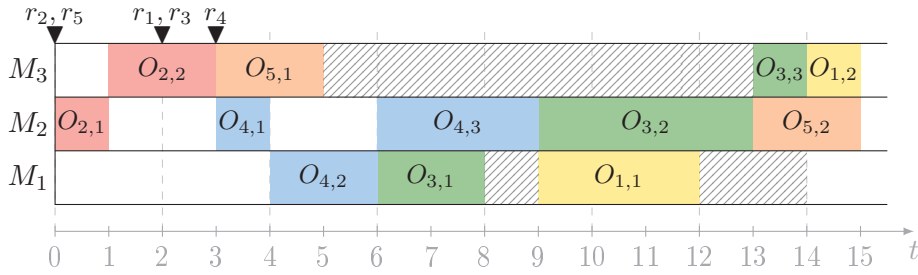


Figure 4.13: API neighbor s' resulting from the API move $O_{3,1} \longleftrightarrow O_{4,2}$ in schedule s for the instance GJSP2

As a result,

$$s_l^{op'} = [O_{2,1}, O_{2,2}, O_{4,1}, \underline{O_{4,2}}, O_{4,3}, \underline{O_{3,1}}, O_{5,1}, O_{3,2}, O_{5,2}, O_{3,3}, O_{1,1}, O_{1,2}],$$

$$s_r^{op'} = [O_{2,1}, O_{2,2}, O_{4,1}, O_{5,1}, \underline{O_{4,2}}, O_{4,3}, \underline{O_{3,1}}, O_{3,2}, O_{5,2}, O_{3,3}, O_{1,1}, O_{1,2}].$$

By chance, these two permutations constitute redundant representations of the same schedule s', which is illustrated in Figure 4.13. Since the equivalence of the resulting permutations does not always appear, two different neighboring schedules might be derived from an initial schedule s by applying one API move followed by right shift and left shift transformation, respectively. Therefore, the neighbors of a schedule are defined accordingly.

Definition 4.2. *An **API-R neighbor** s' of a schedule s is determined by performing an API move in s together with a right shift transformation to obtain the permutation $s^{op'}$ and regain feasibility if necessary. Analogously, an **API-L neighbor** s' of a schedule s is determined by performing an API move in s together with a left shift transformation to obtain the permutation $s^{op'}$ and regain feasibility if necessary. If the API-R and the API-L neighbor appear to be equivalent, the schedule s' is denoted as the **API neighbor** of schedule s.*

As indicated in the previous section, the idea of reducing the search space by exclusively considering promising transitions with regard to the objective function is successfully applied in the literature. In line with this, the concept of API-based transitions is specialized as follows:

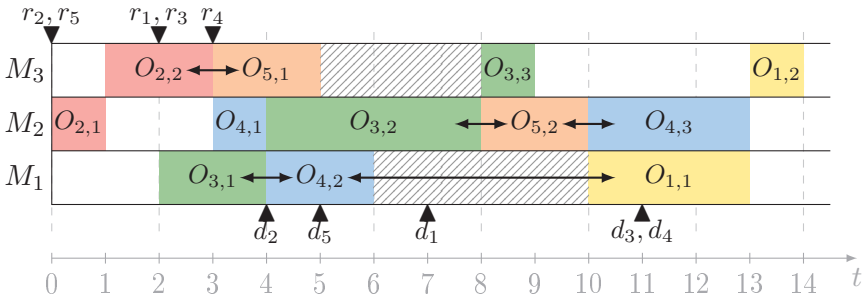


Figure 4.14: Illustration of the set of TAPI moves applicable to a given schedule of the instance GJSP2

Definition 4.3. *A **TAPI move** denotes an interchange of two adjacent operations $O_{i,j}$ and $O_{i',j'}$ of different jobs requiring the same machine $M_k \in \mathcal{M}$ with $O_{i,j} \rightarrow O_{i',j'}$ in the machine-based representation of the schedule, where $f_{i,j} = s_{i',j'}$ holds and the job $J_{i'}$ belongs to the set of tardy jobs $\mathcal{T}$.*

The set of applicable TAPI moves constitutes a subset of the set of applicable API moves. Definition 4.3 involves the additional condition of a tardiness value $T_{i'} > 0$ for the job $J_{i'}$ to which the leftward interchanged operation $O_{i',j'}$ belongs. Figure 4.14 shows the five performable TAPI moves for the schedule s of the instance GJSP2 considered above. Since the job J_3 is completed at $t = 9$ prior to its due date $d_3 = 11$, the API moves interchanging the operations $O_{4,1}$ and $O_{3,2}$ on machine M_2 as well as $O_{5,1}$ and $O_{3,3}$ on machine M_3 do not belong to the set of TAPI moves. Consequently, API- and tardiness-based neighbors of a schedule are defined as follows:

Definition 4.4. *A **TAPI-R neighbor** s' of a schedule s is determined by performing a TAPI move in s together with a right shift transformation to obtain the permutation $s^{op'}$ and regain feasibility if necessary. Analogously, a **TAPI-L neighbor** s' of a schedule s is determined by performing a TAPI move in s together with a left shift transformation to obtain the permutation $s^{op'}$ and regain feasibility if necessary. If the TAPI-R and the TAPI-L neighbor appear to be equivalent, the schedule s' is denoted as the **TAPI neighbor** of schedule s.*

4.7.3 The Advanced Repair Technique

As it is described in the previous section, the operation-based representation $s^{op'}$ derived by applying an API or a TAPI move to a given schedule s might encode an infeasible schedule s'. Since the BRT proposed in Section 4.5 mainly relies on leftward shifts of required operations, it appears that the repairing scheme reverts the performed move and the resulting neighboring schedule s' actually constitutes the initially given schedule s. Consider as an example the schedule of the instance GJSP2 shown in Figure 4.12 and the API move interchanging the operations $O_{4,2}$ and $O_{1,1}$ on machine M_1. By applying the right shift and left shift transformation schemes, the following infeasible permutations are constructed from the machine-based representation $s^{ma'}$.

$$s^{ma'} = [[O_{3,1}, \underline{O_{1,1}}, \underline{O_{4,2}}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{5,2}, O_{4,3}], [O_{2,2}, O_{5,1}, O_{3,3}, O_{1,2}]]$$

$$s_l^{op'} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, \underline{O_{1,1}}, \underline{O_{4,2}}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,2}]$$

$$s_r^{op'} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, O_{3,2}, O_{3,3}, O_{5,2}, \underline{O_{1,1}}, \underline{O_{4,2}}, \underline{O_{4,3}}, O_{1,2}]$$

By using the BRT to generate feasible schedules from $s_l^{op'}$ and $s_r^{op'}$, respectively, the swap groups $W = \{O_{3,2}, O_{4,2}\}$ and $W = \{O_{5,2}, O_{3,3}\}$ are subsequently detected and both swaps are implemented prior to the beginning of the processing of operation $O_{1,1}$. The redundant, here even equal, permutations encoding the feasible schedule $s' = s$ result in

$$\begin{aligned} s_l^{op'} &= s_r^{op'} \\ &= [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, (O_{4,2}, O_{3,2}), (O_{5,2}, O_{3,3}), O_{4,3}, O_{1,1}, O_{1,2}] \end{aligned}$$

and the given ordering $O_{1,1} \rightarrow O_{4,2}$ is reversed. Evidently, swaps constitute very robust structures, which are predetermined by positions of operations prior to the actual swap and the implemented API. Here, the appearances of the operations $O_{4,1}$, $O_{3,1}$ and $O_{5,1}$ with list indices less than $lidx(O_{1,1})$ require both swaps to take place before the processing of operation $O_{1,1}$ is allowed to begin. This implies that further changes in the sequences of the operations on the machines are necessary to realize the ordering of the API move in the neighboring solution s'. Similar observations are made in the literature and as a consequence, the authors

apply destruction operators to a large proportion of the schedule to ensure effective changes and neighboring solutions different from the initial schedules, cf. [98] and [103]. In this thesis, a structured and permutation-based method is proposed to construct a feasible permutation $s^{op'}$ from an initial schedule s given by its encodings s^{op} and s^{ma}, a given API or TAPI move and a machine-based representation $s^{ma'}$ derived from s^{ma} by implementing the desired interchange. This method uses the BRT as its basis and is therefore denoted as the *Advanced Repair Technique (ART)*. Figure 4.15 schematically illustrates the involved strategies and their interaction.

The ART incorporates the mechanisms of the BRT with its subroutines ScheduleOp, AddToQueue and ScheduleCycle as well as a significant extension, which is indicated by the green dashed box on the right-hand side of Figure 4.15. Let $O_{i',j'}$ be a blocking operation determined by the procedure and let its involvement in a swap group be ruled out, the ART checks whether a precedence relation $O_{a,b} \rightarrow O_{i',j'}$ is fixed for the considered operation. Such a sequencing may exist due to a given API-based move or due to a previously required adaptation of the permutation *perm*. The operation $O_{a,b}$ is denoted as the associated operation of $O_{i',j'}$.

If there exists no fixation or the associated operation is already involved in the feasible partial schedule s^{op}, the scheme continues according to the BRT steps. On the contrary, if there exists an associated operation, which is not positioned at a lower list index in the permutation s^{op}, the assignment of the currently treated operation $O_{i',j'}$ to the currently regarded list index $lidx'(*)$ in s^{op} will reverse the given precedence relation. Thus, the ART enters the green extension area, which implements the basic strategy of interchanging the associated operation $O_{a,b}$ further to the left in the operation sequence on its required machine, so that $O_{a,b} \rightarrow O_{i',j'}$ might be realizable in the subsequent run of the ART. More precisely, the subroutine AdaptPerm modifies the given permutation *perm* according to one out of three possible forms of the repositioning strategy and the entire repairing process is started again.

In order to observe the four situations, which might appear when a fixed blocking operation $O_{i',j'}$ is required to be shifted prior to its associated operation $O_{a,b}$, let $O_{i,j}$ be the currently regarded operation with $lidx(O_{i,j}) = 1$ in the permutation *perm* and let $O_{i',j'-1} \rightarrow O_{i,j}$ be the ordering to be

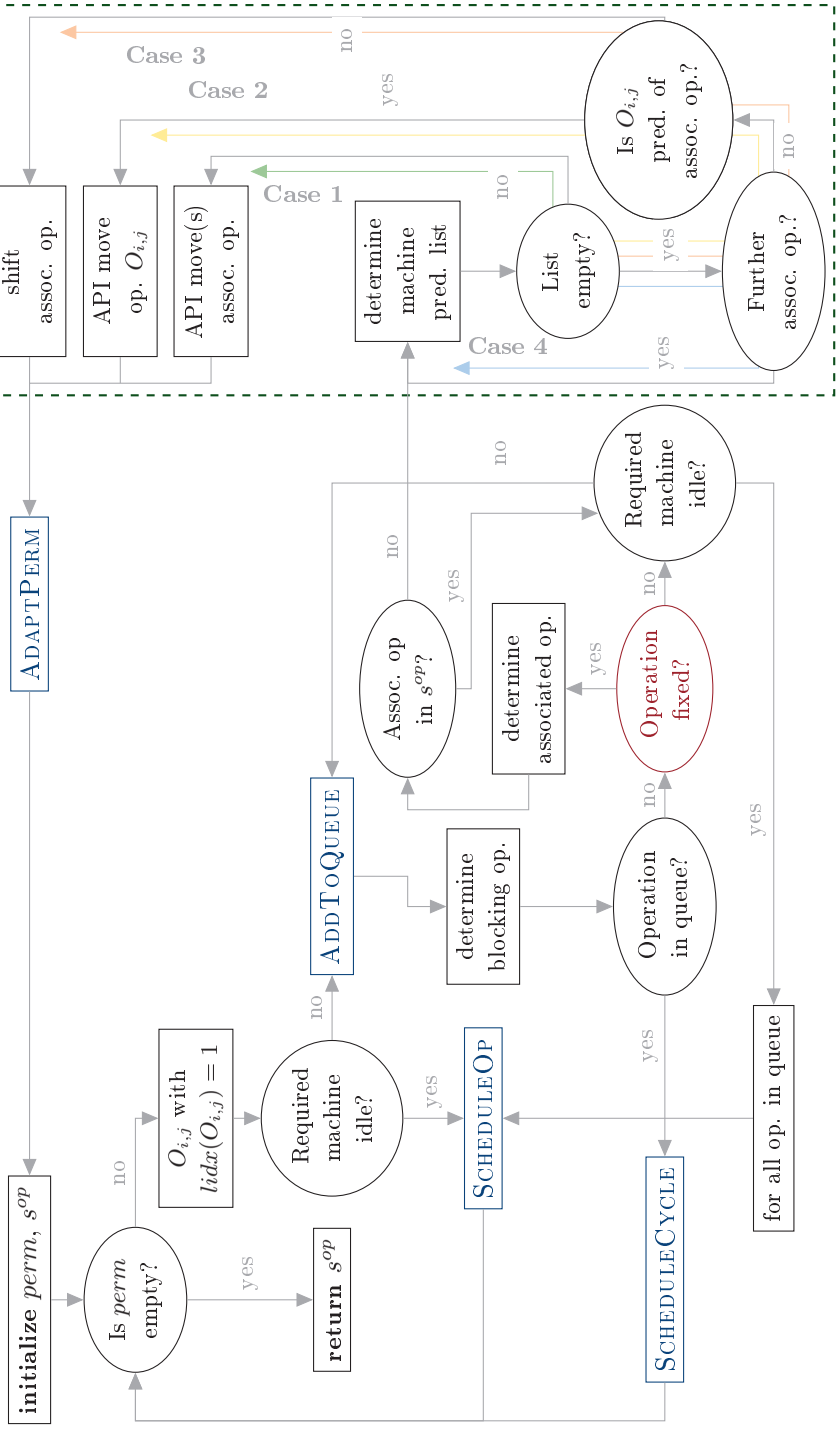


Figure 4.15: Schematic outline of the Advanced Repair Technique (ART)

implemented on machine M_k, whereby operation $O_{i',j'-1}$ is already part of the feasible partial schedule s^{op}. Furthermore, let the set $\mathcal{F}$ incorporate the fixed precedence relations as ordered pairs of operations indicating the irreversible orderings of two operations of different jobs requiring the same machine, such as $(O_{a,b}, O_{i',j'})$. The initially given list *perm* generally involves the following sequence

$$perm = [\ldots, O_{i',j'-1}, \ldots, O_{i,j}, \ldots, O_{a,b}, \ldots, O_{i',j'}, \ldots],$$

with $lidx(O_{i',j'-1}) < lidx(O_{i,j}) < lidx(O_{a,b}) < lidx(O_{i',j'})$, where the operations $O_{i',j'-1}$ and $O_{i,j}$ are directly following each other on machine M_k. Moreover, $O_{a,b}$ and $O_{i',j'}$ are directly processed after another on a different machine $M_{k'}$. According to the strategy mentioned above, an ordered machine predecessor list consisting of all operations which

- precede the associated operation $O_{a,b}$ on machine $M_{k'}$,
- do not belong to the same job J_a and
- feature a position prior to the currently considered index $lidx'(*)$ in s^{op}

is set up. Thereby, $\alpha(O_{a,b})$ denotes the last operation in the predecessor list, which represents the last scheduled operation preceding $O_{a,b}$ on $M_{k'}$. The ART proceeds according to one of the four following cases, which are indicated in green, yellow, orange and blue in Figure 4.15, respectively.

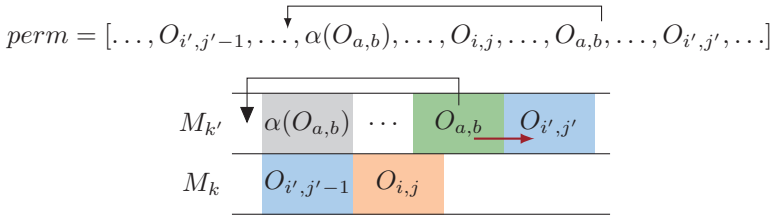
Case 1. If the machine predecessor list is not empty and a machine predecessor $\alpha(O_{a,b})$ exists, AdaptPerm shifts the operation $O_{a,b}$ leftward so that $midx(O_{a,b}) = midx(\alpha(O_{a,b})) - 1$ holds on machine $M_{k'}$. Note that, if $\alpha(O_{a,b})$ constitutes the direct machine predecessor of operation $O_{a,b}$, AdaptPerm performs exactly one API on the machine. Since $lidx(\alpha(O_{a,b})) < lidx(O_{i,j})$ holds for the operations in the current list *perm* and a left shift transformation is applied, the list index of operation $O_{a,b}$ in the modified permutation will be lower than its current index $lidx(O_{a,b})$. The described modification of the permutation *perm* is illustrated in part (a) of Figure 4.16. The red arrow indicates the irreversible precedence relation and the implemented leftward shift of operation $O_{a,b}$ is marked in the diagram as well as in the permutation with a black arrow. Fi-

nally, AdaptPerm extends the set of irreversible orderings $\mathcal{F}$ by the pair $(O_{a,b}, \alpha(O_{a,b}))$.

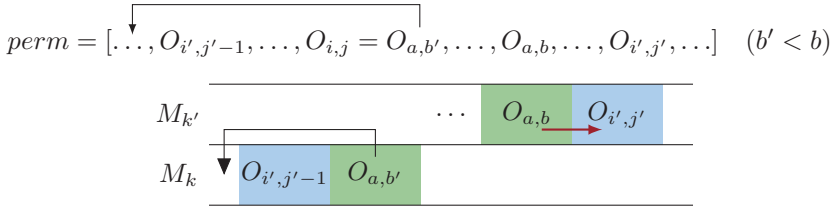
Case 2. If the machine predecessor list is empty and $(O_{a,b}, O_{i',j'})$ describes the only pair involving operation $O_{i',j'}$ as a fixed successor, it might appear that $O_{i,j} = O_{a,b'}$ with $b' < b$ holds and the currently considered operation $O_{i,j}$ constitutes a predecessor of the associated operation $O_{a,b}$ with regard to the processing sequence of job J_a. In this case, AdaptPerm performs an API move and a left shift transformation interchanging operation $O_{a,b'}$ to the left on its machine. As shown in part (b) of Figure 4.16, this leads to a synchronization of the operation sequences on the machines.

Case 3. Assume that the machine predecessor list is empty, $(O_{a,b}, O_{i',j'})$ constitutes the only pair in the set $\mathcal{F}$ involving operation $O_{i',j'}$ as a successor and the associated operation $O_{a,b}$ and the currently considered operation $O_{i,j}$ belong to different jobs. Then, the subroutine AdaptPerm shifts the associated operation $O_{a,b}$ to the left in the permutation $perm$, so that $lidx(O_{a,b}) < lidx(O_{i,j})$ is satisfied and the operation $O_{a,b}$ is forced to enter the feasible partial schedule s^{op} prior to operation $O_{i,j}$ in the subsequent run of the ART. Part (c) of Figure 4.16 illustrates this mechanism.

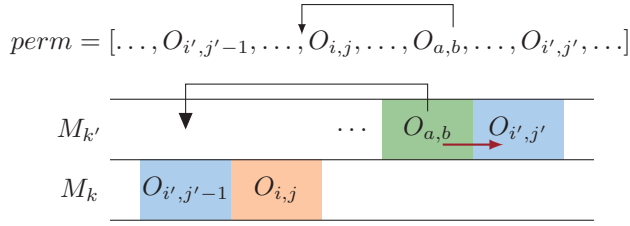
Case 4. This case describes the most complex situation, since it is caused by a recirculating job J_a and may only appear after several repeated runs of the ART on a given permutation. Therefore, the example illustrated in part (d) of Figure 4.16 constitutes an extension of the situation given in Case 2 (part(b)). Assume that due to a given precedence relation $(O_{a,b}, O_{i',j'})$ on machine $M_{k'}$, the ART has operated two times according to Case 2 shifting the operations $O_{a,b'}$ and $O_{a,b''}$ with $b' < b'' < b$ to the left on machine M_k and creating the fixed orderings $(O_{a,b'}, O_{i',j'-1})$ and $(O_{a,b''}, O_{i',j'-1})$. With regard to the current run of the ART, where the operation $O_{i,j}$ requires the blocking operation $O_{i',j'-1}$, there exists more than one operation associated with this fixed operation. Since the set $\mathcal{F}$ is checked according to the entrance ordering of the pairs, the operation $O_{a,b'}$, which does not have a machine predecessor, is first detected as the associated operation of $O_{i',j'-1}$. Consequently, if the machine predecessor list is empty and the blocking operation appears more than once as a successor in a fixed precedence relation, another associated operation is determined, here operation $O_{a,b''}$. Dependent on whether there exists a



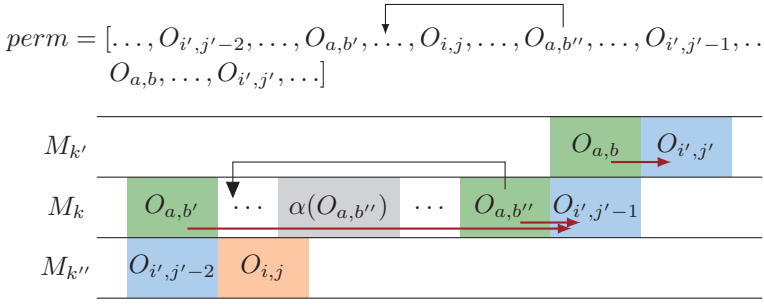
(a) **Case 1:** Shifting the associated operation $O_{a,b}$ in the operation sequence on its machine



(b) **Case 2:** Shifting the currently considered operation $O_{i,j} = O_{a,b'}$ in the operation sequence on its machine



(c) **Case 3:** Shifting the associated operation $O_{a,b}$ in the permutation *perm*



(d) **Case 4:** Detecting and shifting further associated operations like operation $O_{a,b''}$ of the fixed blocking operation $O_{i',j'-1}$

Figure 4.16: Adapting the permutation *perm* due to a fixed blocking operation in the ART

machine predecessor $\alpha(O_{a,b''})$, the operation $O_{a,b''}$ is shifted by the ART according to Case 1 or Case 3.

After determining an operation to be interchanged or shifted together with its newly required position in the permutation, the subroutine AdaptPerm recalls the latest version of the list *perm* and implements the modification. Independent of whether Case 1, Case 2 or Case 3 is to be realized, AdaptPerm directly shifts the operation in the operation-based representation by using built-in list operators. Thus, based on the assumptions on the technical implementation given in the foregoing, the shift of an operation is executed in constant time. Since the ART requires to operate on a permutation *perm*, which is feasible with regard to the processing sequences and the technological routes of all jobs, these restrictions need to be assured during the modification. Thus, for every shift of an operation $O_{a,b}$ implemented by AdaptPerm, it is checked whether the operation is inserted prior to one of its predecessors. If this situation appears, the predecessor $O_{a,b'}$ $(b' < b)$ is additionally shifted either prior to a machine predecessor $\alpha(O_{a,b'})$ that features a position with $lidx(\alpha(O_{a,b'})) < lidx(O_{a,b})$ or to the shifted operation $O_{a,b}$. In the most extreme case, the initially shifted associated operation $O_{a,b}$ constitutes the last operation of the job J_a and the position it is shifted to features a lower list index than $O_{a,1}$. Consequently, all operations of the job J_a require a shift in the permutation and the maximum number of necessary computation steps of AdaptPerm is of order $O(\max\{n_i \mid J_i \in \mathcal{J}\})$, correspondingly.

Algorithm 3 describes the ART technically. The method takes a list *perm* of all operations $O_{i,j} \in \mathcal{O}$, which is feasible with regard to the processing sequences and the technological routes of all jobs $J_i \in \mathcal{J}$, and an irreversible ordering $O_{a,b} \rightarrow O_{c,d}$ of two operations of different jobs on a specific machine M_k as its input. The ART returns the operation-based representation s^{op} of a feasible schedule for the BJSPT involving the given partial sequence $O_{a,b} \rightarrow O_{c,d}$. It can be observed that the underlying basic structure of the BRT is still visible in the lines 5 to 16 and 39 to 54 of Algorithm 3, but there exists one significant difference between the two methods. While the BRT generates a feasible permutation after one initialization phase and a finite number of repetitions of the **while**-loop, the ART might require repeated restarts of the **while** command in line 6 with

Algorithm 3 Advanced Repair Technique (ART)

Input: permutation (list) of operations $perm$, one irreversible API ordering $O_{a,b} \rightarrow O_{c,d}$
Output: operation-based representation s^{op} of a feasible schedule involving the ordering $O_{a,b} \rightarrow O_{c,d}$

initialize s^{op}, Q and W as empty lists, ST as an empty starting times dictionary
initialize MaS as a dictionary including the status of all machines $M_k \in \mathcal{M}$ (= idle)
initialize $\mathcal{F} = \{(O_{a,b}, O_{c,d})\}$, PL as an empty machine predecessor list

$O_{irr} \leftarrow O_{a,b}$
$lidx'(*) \leftarrow 1$
while $perm$ is not empty **do**
  if $O_{irr} \in s^{op}$ **then**
    clear $\mathcal{F}$
  end if
  $O_{i,j} \leftarrow$ operation with $lidx(O_{i,j}) = 1$ in $perm$
  if $Machine(O_{i,j})$ is idle **then**
    ScheduleOp($O_{i,j}$, $perm$, s^{op}, ST, $lidx'(*)$, MaS)
  else
    AddToQueue(Q, $O_{i,j}$)
    $O_{i',j'} \leftarrow$ blocking operation with $lidx'(O_{i',j'}) \leq lidx'(O_{i,j})$
    while $O_{i',j'} \notin Q$ **do**
      if $O_{i',j'} \in \mathcal{F}$ as a successor **then**
        if $\exists$ unscheduled associated operation $O_{a,b'}$ of $O_{i',j'}$ **then**
          PL $\leftarrow$ machine predecessor list of $O_{a,b'}$
          while PL is empty AND $\exists$ further unscheduled assoc. op. of $O_{i',j'}$ **do**
            $O_{a,b'} \leftarrow$ next unscheduled assoc. op. of $O_{i',j'}$ [Case 4]
            PL $\leftarrow$ machine predecessor list of $O_{a,b'}$
          end while
          if PL is empty **then**
            if $J_i == J_a$ **then** [Case 2]
              AdaptPerm($perm$, $O_{i,j} = O_{a,b'}$, $lidx(\alpha(O_{i,j}))$, $\mathcal{F}$)
              reinitialize (repeat lines 1 and 2)
              $lidx'(*) \leftarrow 1$
            else [Case 3]
              AdaptPerm($perm$, $O_{a,b'}$, $lidx(O_{i,j})$, $\mathcal{F}$)
              reinitialize (repeat lines 1 and 2)
              $lidx'(*) \leftarrow 1$
            end if
          else [Case 1]
            AdaptPerm($perm$, $O_{a,b'}$, $lidx(\alpha(O_{a,b'}))$, $\mathcal{F}$)
            reinitialize (repeat lines 1 and 2)
            $lidx'(*) \leftarrow 1$
          end if
        end if
        break

```
41:             else
42:                 if Machine(O_{i',j'}) is not idle then
43:                     AddToQueue(Q, O_{i',j'})
44:                     O_{i',j'} ← blocking operation O_{i'',j''} with lidx'(O_{i'',j''}) ≤ lidx'(O_{i',j'})
45:                 else
46:                     ScheduleOp(O_{i',j'}, perm, s^op, ST, lidx'(∗), MaS)
47:                     for O_{i,j} ∈ Q do ScheduleOp(O_{i,j}, perm, s^op, ST, lidx'(∗), MaS)
48:                     end for
49:                     break
50:                 end if
51:             end if
52:         end while
53:         if Q is not empty then
54:             ScheduleCycle(Q, perm, s^op, ST, lidx'(∗), MaS)
55:         end if
56:     end if
57: end while
58: return s^op (return ST)
```

modified permutations *perm*, see lines 27, 30 and 34. This results from the fact that a given ordering, even if it only involves two operations, constitutes a partial schedule requiring to be completed. Basically, it is obvious that a feasible schedule s' for the BJSPT involving a distinct ordering of two operations of different jobs does always exist. However, its construction as a neighboring solution of an initially given schedule s cannot be performed trivially. When the ART is applied, the schedule is set up stepwise like in the BRT, so that more and more precedence relations given at the beginning of the permutation *perm* are realized in the feasible partial schedule s^{op}. If the predefined ordering $O_{a,b} \rightarrow O_{c,d}$ is implemented at a posterior position in the list *perm*, one of the changes in the ordering of the unscheduled operations, which is required during the extension of the feasible permutation s^{op}, may be forbidden. Thus, the **if** condition in line 17 will be true. In this case, the feasible sequencing given by the beginning of the permutation *perm* contradicts the irreversible ordering $O_{a,b} \rightarrow O_{c,d}$ and the completion of the schedule is impossible. Note that in contrast to the BRT, the ART detects and realizes required changes in the feasible partial schedule to adapt the beginning of the permutation to the given irreversible sequence.

Observations

(I) Once the preceding operation of the predefined ordering $O_{a,b}$ is incorporated in the feasible partial schedule s^{op} and a reversion of the given sequence $O_{a,b} \rightarrow O_{c,d}$ is not possible anymore, the **if** condition in line 7 of Algorithm 3 is true, the set of irreversible orderings $\mathcal{F}$ is cleared and the ART runs as the BRT without entering the **if** statement in line 17.

(II) All operations $O_{a,b'} \in \mathcal{O}^a$ that are shifted or interchanged by the subroutine AdaptPerm belong to the same job $J_a \in \mathcal{J}$, which is defined by the predecessor of the initially given ordering $O_{a,b} \rightarrow O_{c,d}$. More precisely, if the ART enters the modification scheme in the lines 17 to 36 of Algorithm 3, either the firstly given preceding operation $O_{a,b}$ or one of its predecessors $O_{a,b'}$ with $b' < b$ will be assigned to a new position in the list *perm* independent of the implementation of Case 1, Case 2 or Case 3.

Proposition 4.2. *The ART terminates and returns a permutation s^{op} encoding a feasible schedule for the BJSPT involving a predefined ordering $O_{a,b} \rightarrow O_{c,d}$ of two operations of different jobs requiring a specific machine M_k.*

Proof. The ART mirrors the BRT as long as there is no blocking operation $O_{i',j'}$ involved as a successor in an irreversible ordering, see Observation I. Recall that, according to Proposition 4.1 in Section 4.5, it is given that the BRT terminates and returns a feasible encoding s^{op} of a schedule for the BJSPT from an arbitrary permutation *perm*, which is feasible with regard to the processing sequences and the technological routes of all jobs $J_i \in \mathcal{J}$. Furthermore, a change made in the list *perm* by the subroutine AdaptPerm following Case 1, Case 2 or Case 3 given above is denoted as a *modification* of the permutation.

Consequently, it has to be shown that

(1) a modification does not violate processing sequence or technological route restrictions,

(2) a modification can never be reversed,

(3) the number of possible modifications is finite and

(4) there exists a sequence of modifications leading to a feasible schedule for the BJSPT including the predefined ordering $O_{a,b} \rightarrow O_{c,d}$.

The initially given list *perm* is feasible with regard to the processing sequences and the technological routes of all jobs $J_i \in \mathcal{J}$. When the subroutine AdaptPerm is executed, an operation $O_{a,b'}$ is shifted to a position with a lower list index in the permutation. This may only violate the processing sequence or the technological route of the corresponding job J_a, if the operation is shifted prior to one or more of its predecessors $O_{a,b''}$ with $b'' < b'$. The occurrence of this situation is checked during the operation of AdaptPerm and the affected predecessors are additionally shifted, so that $lidx(O_{a,b''}) < lidx(O_{a,b'})$ is fulfilled again by the list *perm*. Thus, (1) is true.

The set of irreversible orderings $\mathcal{F}$ is extended by a pair of operations in every execution of AdaptPerm, forcing the currently shifted operation $O_{a,b'}$ to be scheduled prior to its newly defined list successor in all remaining iterations of the ART. Thus, the incorporated BRT mechanisms can never reverse a modification. Every consecutively required execution of AdaptPerm results from a blocking operation $O_{i',j'}$ involved in an ordering $(O_{a,b'}, O_{i',j'}) \in \mathcal{F}$. Hence, the currently regarded operation $O_{i,j}$ features a list index between $lidx(O_{i',j'-1})$ and $lidx(O_{a,b'})$, since $lidx(O_{a,b'}) = lidx(O_{i',j'}) - 1$ holds. This means that a consecutively required modification does always appear at a position prior to the previous modification causing the considered pair $(O_{a,b'}, O_{i',j'})$ and as a consequence, the operation $O_{a,b'}$ or one of its predecessors $O_{a,b''}$ with $b'' < b'$ is shifted further to a list index smaller than $lidx(O_{i,j}) < lidx(O_{a,b'}) < lidx(O_{i',j'})$ in the list *perm*, see Observation II. Thus, an implemented modification such as $O_{a,b'} \rightarrow O_{i',j'}$ can never be reversed by an ART mechanism. (2) is true.

Since the list *perm* contains a finite number of elements, the set of shifted operations is restricted to at most all operations of a given job $J_a \in \mathcal{J}$, see Observation II, and an implemented modification can never be reversed, the number of possible modification is finite. (3) holds.

Following from Observation II and the proven statements (1) and (2), the strategy of the ART can be summarized as shifting the operations $O_{a,b'}$ of a job J_a determined by the initially given ordering $O_{a,b} \rightarrow O_{c,d}$, $(b' < b)$,

stepwise to smaller list indices in the permutation $perm$ until the predefined partial sequence is realized in a feasible schedule for the BJSPT, which is constructed by the incorporated BRT mechanisms only. This strategy succeeds, since the initially given operation $O_{a,b}$ and, if necessary, its predecessors $O_{a,b'}$ are shifted to the left in the operation sequences on the machines and correspondingly in the permutation $perm$, while the operation $O_{c,d}$ can never be moved to a position prior to $O_{a,b}$. Considering the arguments given to show (3), the final consequence will be a permutation

$$perm = [O_{a,1}, O_{a,2}, \ldots, O_{a,b}, \ldots, \beta(O_{a,1}), \ldots, \beta(O_{a,2}), \ldots, \beta(O_{a,b}), \ldots],$$

where $\beta(O_{i,j})$ denotes the machine successor of an operation $O_{i,j}$ and either $\beta(O_{a,b}) = O_{c,d}$ or $lidx(O_{c,d}) > lidx(\beta(O_{a,b}))$ holds. Thus, in the extreme case, the job J_a involving operation $O_{a,b}$ is scheduled prior to all other jobs involved in the problem and $O_{a,b} \rightarrow O_{c,d}$ is guaranteed. (4) is shown. □

Another important aspect is the runtime in which a feasible operation-based permutation s^{op} can be constructed by the ART. The initialization steps in the lines 1 to 3 in Algorithm 3 are performed in $O(m)$ time. As a basis for the following observations, the number of elements in the permutation s^{op} and the maximum number of operations requiring the same machine $\max\left\{|\Omega^k| \mid M_k \in \mathcal{M}\right\}$ are estimated by $O(n_{op})$.

At first, the mechanisms operating inside the **while**-loop starting in line 6 of Algorithm 3 are considered. The **if** condition checking for the implementation of the given ordering by $O_{irr} \in s^{op}$ in the lines 7 to 9 is operated in $O(n_{op})$. The subsequent steps are equivalent to the BRT in Algorithm 2 up to the conditional statement regarding the fixation of a blocking operation in line 17. Checking whether $O_{i',j'} \in \mathcal{F}$ as a successor requires a simple dictionary request with $O(const)$, whereby the determination of an unscheduled associated operation $O_{a,b'}$ and the setup of a machine predecessor list PL take $O(n_{op})$ each. The following **while**-loop in the lines 20 to 23 may be iterated at most $\max\{n_i \mid J_i \in \mathcal{J}\}$ times, involving the setup of a machine predecessor list in each run. Thus, since $n_i \leq n_{op}$ holds for all $J_i \in \mathcal{J}$, the **while**-loop takes $O\left((n_{op})^2\right)$ steps. Considering the lines 24 to 35 of the ART, independent of the satisfied conditions, the procedure

runs the function AdaptPerm with $O(\max\{n_i \mid J_i \in \mathcal{J}\}) = O(n_{op})$ and reinitializes the input data with $O(m)$. Consequently, when **if** is true in line 17, there are $O\left(2 \cdot n_{op} + (n_{op})^2 + n_{op} + m\right) = O\left((n_{op})^2\right)$ steps to be executed before **while** is left in line 37. On the contrary, when **else** is true in line 38, the steps to be implemented are equivalent to the lines 12 to 20 in Algorithm 2 of the BRT and operated in $O(m)$. The surrounding **while**-loop in the lines 16 to 49 may be operated either by running **if** singly in $O\left((n_{op})^2\right)$ or by running **else** singly in $O(m)$ or by running the **else** lines 39 to 41 in $O(m)$ followed by **if** with $O\left((n_{op})^2\right)$. Since $m < (n_{op})^2$ holds, the **while**-loop in the lines 16 to 49 is operated in $O\left((n_{op})^2\right)$. The following lines 50 to 52 do also constitute a BRT mechanism realizable in $O(m)$ steps.

Considering the overall procedure, the **while**-loop, namely the lines 6 to 54, is iterated with repeated reinitialization until the irreversible ordering $O_{a,b} \rightarrow O_{c,d}$ can be implemented in the schedule s^{op} and the last iteration runs without entering the **if** condition in line 17. One iteration of the **while**-loop with reinitialization is operated in $O\left(n_{op} + (n_{op})^2 + m\right) = O\left((n_{op})^2\right)$. The maximum number of possible modifications of the list *perm* will be reached, if all operations of a specific job are shifted stepwise from the last to the first positions in the permutation. Thus, the maximum number of reinitializations is restricted by the product of the number of operations n_a of the job J_a, to which the initially given predecessor $O_{a,b}$ belongs, and the number of operations in the permutation n_{op}. As a consequence, the **while**-loop in the lines 6 to 54 involves at most $O\left((n_{op})^2 \cdot (n_{op})^2\right) = O\left((n_{op})^4\right)$ steps. Since $m < (n_{op})^4$, the runtime of the initialization can be disregarded. Overall, the ART determines the operation-based representation of a feasible schedule involving a given irreversible ordering in polynomial time

$$O\left((n_{op})^4\right).$$

In the following, the ART is applied to construct feasible neighboring solutions s' from given schedules s and chosen API or TAPI moves. Thus, the feasibility of the API-based neighbors and the TAPI-based neighbors described in the Definitions 4.2 and 4.4, respectively, is assured by the technique proposed above. Consider the example of an API move inter-

changing the operation $O_{4,2}$ and $O_{1,1}$ in a schedule s of the instance GJSP2 shown at the beginning of this section. The following permutations do not yield neighboring solutions s' that are different from the initially given schedule s when the BRT is applied to construct feasible schedules for the BJSPT.

$$\begin{aligned} s_l^{op'} &= perm_l \\ &= [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, \underline{O_{1,1}}, O_{4,2}, O_{3,2}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,2}] \\ s_r^{op'} &= perm_r \\ &= [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, \underline{O_{3,2}}, O_{3,3}, O_{5,2}, O_{1,1}, O_{4,2}, O_{4,3}, O_{1,2}] \end{aligned}$$

On the contrary, the ART operates on the operation-based representations $s_l^{op'}$ and $s_r^{op'}$ while taking the irreversible API $O_{1,1} \rightarrow O_{4,2}$ into account. The algorithm detects the fixed blocking operation $O_{4,2}$ and the a required modification during the consideration of the underlined operations $O_{1,1}$ in $perm_l$ and $O_{3,2}$ in $perm_r$, respectively. In both executions of the ART, the machine predecessor list is determined with $PL = [O_{3,1}]$ and the associated operation $O_{1,1}$ is shifted according to Case 1 to the second position in the permutation, so that $lidx(O_{3,1}) - 1 = lidx(O_{1,1})$ is fulfilled. Since operation $O_{1,1}$ constitutes the first operation of job J_1, there is no additional shift of preceding operations required and the ART is reinitialized with the extended set of irreversible orderings $\mathcal{F} = \{(O_{1,1}, O_{4,2}), (O_{1,1}, O_{3,1})\}$. After the first two operations of the modified permutations, namely the operations $O_{2,1}$ and $O_{1,1}$, have been considered and added to the feasible partial schedules $s_l^{op'}$ and $s_r^{op'}$, respectively, the set $\mathcal{F}$ is cleared and the procedure constructs complete feasible schedules based on the BRT mechanisms. Due to the simplicity of the given example, the resulting permutations are equivalent and do equally encode the neighboring schedule s' illustrated in Figure 4.17.

$$\begin{aligned} s_l^{op'} &= s_r^{op'} \\ &= [O_{2,1}, O_{1,1}, O_{1,2}, O_{3,1}, O_{2,2}, O_{4,1}, O_{5,1}, (O_{4,2}, O_{3,2}), (O_{3,3}, O_{5,2}), O_{4,3}] \end{aligned}$$

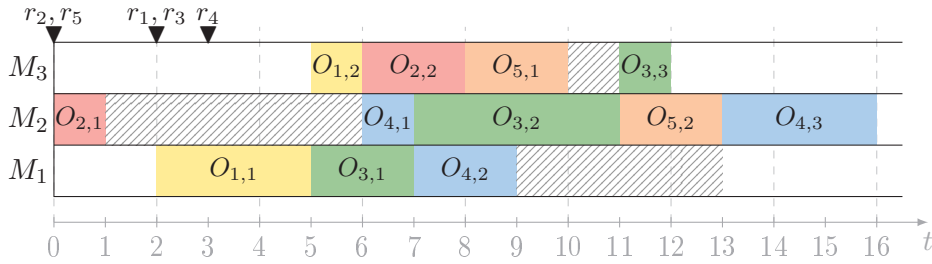


Figure 4.17: API neighbor s' resulting from the API move $O_{4,2} \longleftrightarrow O_{1,1}$ in schedule s for the instance GJSP2

Note that the distance (see Section 4.3) of the initially given schedule s and its API neighbor s', which is constructed based on one single API move, results in

$$\begin{aligned}\delta(s,s') &= (h_{1,1,3,1} + h_{1,1,4,2} + h_{3,1,4,2}) \\ &\quad + (h_{2,1,3,2} + h_{2,1,4,1} + h_{2,1,4,3} + h_{2,1,5,2} + h_{3,2,4,1} + h_{3,2,4,3} + \\ &\qquad h_{3,2,5,2} + h_{4,1,5,2} + h_{4,3,5,2}) \\ &\quad + (h_{1,2,2,2} + h_{1,2,3,3} + h_{1,2,5,1} + h_{2,2,3,3} + h_{2,2,5,1} + h_{3,3,5,1}) \\ &= (1+1+0) + (0+0+0+0+0+0+0+0+0) + \\ &\quad (1+1+1+0+0+0) \\ &= 5.\end{aligned}$$

Since the incorporation of blocking constraints in the job shop problem leads to additionally required changes in the permutations of neighboring solutions in many cases, it can be expected that the distances of API-based neighbors are significantly higher for the BJSPT than for other less constrained job shop problems. This conclusion is drawn in line with the literature, cf. [19], and this property of the BJSPT is empirically observed in Section 4.7.7.

4.7.4 Adjacent Pairwise Interchange-Based Neighborhoods

Considering the API-based neighbors introduced by the Definitions 4.2 and 4.4 in Section 4.7.2 and applying the ART proposed in Section 4.7.3, the following two neighborhoods for the BJSPT are defined.

Definition 4.5. *The **API neighborhood** $\mathcal{N}_A(s)$ of a schedule s is defined as the set of schedules s', where s' is a feasible API-R or a feasible API-L neighbor of s.*

Definition 4.6. *The **TAPI neighborhood** $\mathcal{N}_T(s)$ of a schedule s is defined as the set of schedules s', where s' is a feasible TAPI-L or a feasible TAPI-R neighbor of s.*

Observations

(III) The API neighborhood of a schedule s is empty, if the release dates of the jobs are given so that the jobs are processed after another without any overlapping, see Figure 4.11.

(IV) The TAPI neighborhood of a schedule s is empty, if either the API neighborhood of the schedule is empty or the total tardiness of the schedule is equal to zero.

(V) $\delta(s, s') \in \left[1, \sum_{M_k \in \mathcal{M}} \binom{|\Omega^k|}{2}\right]$ holds for the distance of an initial schedule s and any API-based neighbor s' .

4.7.5 A Randomized Shift-Based Neighborhood

The API-based neighborhoods proposed in the previous section are derived from a basic and nicely controllable transition scheme, which is intended to intensify the search in the direction of local optima. Since the BJSPT constitutes a highly constrained problem and the search space is supposed to be rugged, a randomized component shall be incorporated in the application of the metaheuristics. To diversify the search, a *Tardy Job (TJ) Neighborhood* is introduced, which relies on random shifts of all operations of a randomly chosen tardy job in the operation-based representation of a schedule.

Definition 4.7. *A **random leftward shift** denotes the shift of an operation $O_{i,j} \in \mathcal{O}$ featuring the current list index $lidx(O_{i,j})$ in the operation-based representation of a schedule s to a randomly determined new list index $lidx'(O_{i,j})$ with*

$$lidx'(O_{i,j}) \in [1, 2, \ldots, lidx(O_{i,j}) - 1] \qquad \text{for } j = 1, J_i \in \mathcal{J},$$
$$lidx'(O_{i,j}) \in [lidx'(O_{i,j-1}) + 1, \ldots, lidx(O_{i,j})] \qquad \text{for } j = 2, \ldots, n_i, J_i \in \mathcal{J}.$$

To construct a neighboring solution s' of a given schedule s, the set of tardy jobs, which involves all jobs $J_i \in \mathcal{J}$ with $T_i > 0$, is determined. A neighbor-defining job $J_{i'}$ is randomly chosen from this set and the random leftward shift is applied to all operations $O_{i',j'} \in \mathcal{O}^{i'}$ according to increasing $j' = 1, \ldots, n_{i'}$. This construction of the operation-based representation $s^{op'}$ of the neighboring schedule s' generally takes $O(n + \max\{n_i \mid J_i \in \mathcal{J}\})$ steps with regard to the assumptions on the technical implementation given in the foregoing. Consider as an example the schedule s of the instance GJSP2 shown in Figure 4.14 with the given due dates $d_1, \ldots, d_5$. The tardiness values of the jobs are determined by $T_1 = 7,\ T_2 = T_3 = 0,\ T_4 = 2,\ T_5 = 5$ and the set of tardy jobs consists of J_1, J_4 and J_5, accordingly. Let the job J_5 be chosen to determine a neighboring schedule s'. The application of the random leftward shift to the operations $O_{5,1}$ and $O_{5,2}$ may be executed as follows:

$$s^{op} = [O_{2,1}, O_{3,1}, O_{2,2}, O_{4,1}, \underline{O_{5,1}}, O_{4,2}, O_{3,2}, O_{3,3}, \underline{O_{5,2}}, O_{4,3}, O_{1,1}, O_{1,2}]$$

$$s^{op'} = [O_{2,1}, O_{3,1}, \underline{O_{5,1}}, O_{2,2}, O_{4,1}, \underline{O_{5,2}}, O_{4,2}, O_{3,2}, O_{3,3}, O_{4,3}, O_{1,1}, O_{1,2}]$$

The resulting permutation $s^{op'}$ is feasible with regard to the processing sequences and the technological routes of all jobs by definition, but may be infeasible with regard to blocking constraints. Therefore, the BRT is operated to construct a feasible schedule s' for the BJSPT. The procedure returns

$$s^{op'} = [O_{2,1}, O_{3,1}, O_{5,1}, (O_{2,2}, O_{5,2}), O_{4,1}, (O_{4,2}, O_{3,2}), O_{3,3}, O_{4,3}, O_{1,1}, O_{1,2}]$$

encoding the neighboring schedule illustrated in Figure 4.18.

Definition 4.8. *A **TJ neighbor** s' of a schedule s is determined by applying random leftward shifts to all operations of a tardy job $J_{i'}$ starting from $O_{i',1}$ to $O_{i',n_{i'}}$ and regain feasibility, if necessary.*

Note that the application of the ART is not possible here, since there are $n_{i'} > 1$ irreversible orderings defined in the construction of the neighboring solution, which do not necessarily correspond to a feasible and expendable partial schedule. Therefore, the BRT constitutes the only applicable repairing scheme and consequently, a feasible TJ neighbor s' may result to be equivalent to the initially given schedule s.

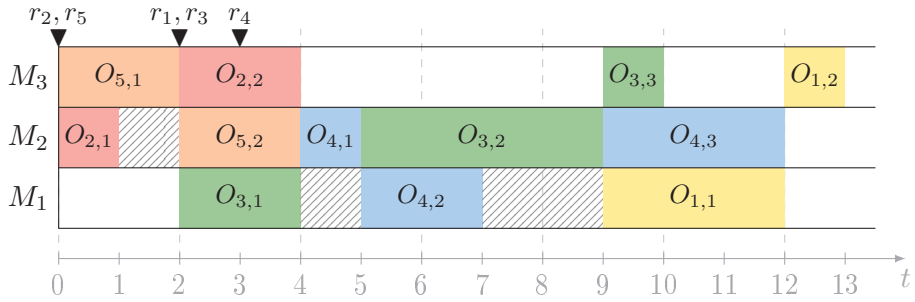


Figure 4.18: TJ neighbor s' resulting from random leftward shifts of all operations of job J_5 in schedule s of the instance GJSP2

Definition 4.9. *The* ***TJ neighborhood*** $\mathcal{N}_J(s)$ *of a schedule* s *is defined as the set of schedules* s'*, where* s' *is a feasible TJ neighbor of* s*.*

Observations

(VI) The TJ neighborhood of a schedule s is empty, if the total tardiness of the solution is equal to zero.

(VII) The distance $\delta(s, s')$ of the initial schedule s and a TJ neighbor s' may be an arbitrary integer from the interval $\left[0, \sum_{M_k \in \mathcal{M}} \binom{|\Omega^k|}{2}\right]$.

4.7.6 On the Connectivity of the Neighborhoods

The connectivity of the neighborhood is a fundamental characteristic, which implies that every existing solution can be transformed into every other existing solution by the (repeated) application of the given neighbor-defining operator, cf. [32] and [64]. This property guarantees that an operating metaheuristic is capable to obtain the optimal solution of a given problem. However, a theoretical finding on the connectivity of a neighborhood can only give an implication on the expected performance of the transition structure on test instances. Thus, the following proposition constitutes a first step in evaluating the applicability and advantageousness of the given neighborhoods for the BJSPT.

Proposition 4.3. *The proposed neighborhoods for the BJSPT, namely the API neighborhood, the TAPI neighborhood and the TJ neighborhood, are* ***not connected****.*

Proof. Refer to the Observations III, IV and VI given in the Sections 4.7.4 and 4.7.5, respectively. Based on the general properties of the problem, such as release dates, technological routes and processing times, there may exist isolated solutions in all of the three neighborhoods, for which no neighboring solution is defined even if it exists. Thus, none of the neighborhoods is connected. □

Considering the finding on the API and the TAPI neighborhoods, it can be observed that the main aspect breaking the connectivity is the general release dates $r_i \geq 0$ for $J_i \in \mathcal{J}$. However, the existence of isolated schedules, for which the API-based neighborhoods are empty, does only appear, when the release dates are given so that the jobs can be processed one after another without overlapping in time. This constitutes an extreme case of a scheduling problem, for which the application of a heuristic search procedure is actually not required. Thus, it can be assumed that, if practically relevant BJSPT instances are considered, where jobs are competing for processing on the machines, isolated solutions do not exist in the API-based neighborhoods and their application may still be successful.

For the TJ neighborhood, the emptiness of the neighborhood of a schedule is caused by the fact that the tardiness value $T_i = 0$ for all $J_i \in \mathcal{J}$. Thus, every isolated schedule constitutes an optimal solution for the BJSPT. Since the application of a metaheuristic does not mainly aim at determining all optimal solutions of a given problem and the randomized transition scheme is incorporated to diversify the search process, the TJ neighborhood may still form a beneficial component.

4.7.7 Characteristics of the API-Based Neighboring Solutions and the Corresponding Neighborhoods

In this section, several properties of the neighboring schedules defined in the previous sections and the proposed neighborhoods are examined. Especially, some characteristics arising from the required construction and repairing schemes are considered. The underlying data is collected from all neighboring solutions that are generated during the application of the

SA metaheuristic presented in Section 4.9. The empirical study involves the following aspects:

- the proportion of API and TAPI neighbors (equivalently resulting from right shift and left shift transformation) among all API-based neighboring solutions,
- the performance of non-equivalent API-based neighbors resulting from right shift and left shift transformation with regard to the total tardiness,
- the mean distance of neighboring schedules in the API-based neighborhoods,
- the distribution of the distance measure among all API-based neighbors and
- potential differences in the distance distributions of API-based neighbors generated by applying left shift and right shift transformation.

On the proportion of equivalent solutions resulting from right shift and left shift transformation following the same API or TAPI move. The Figures 4.19 and 4.20 are based on all neighboring schedules s', which are constructed by one specific API or TAPI move applied to a given schedule s. The first illustrated aspects are the proportion of equivalent permutations after transformation (EQ) shown in gray and the proportion of API or TAPI moves leading to different schedules s' indicated by the sum of the corresponding blue stacks (LS, RS). Considering the train scheduling-inspired instances, more than 75% of the API and TAPI moves construct exactly one neighboring solution, for which the choice of the transformation scheme is irrelevant, see Figure 4.19. On the contrary, this statement is only true for the Lawrence instances la01 to la15 in Figure 4.20, while the proportion of equivalent permutations generally decreases with increasing problem size. Note that for the instances featuring similar quantities of jobs and machines, namely the instances ts01 to ts15 and la16 to la30, the Lawrence instances show a significantly smaller proportion of equivalent neighboring solutions. This may be reasoned by the amount of structure and randomness involved in the underlying problems. While the train scheduling inspired-instances are based on a predefined railway network and jobs are processed according to a finite number of technological routes

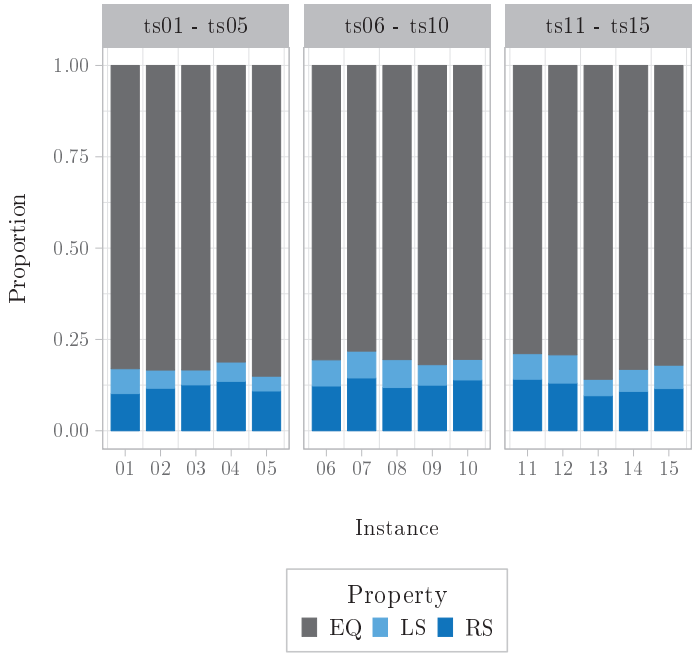


Figure 4.19: Proportion of equivalent solutions and performance of right shift and left shift transformation among all API-based neighbors for the train scheduling-inspired instances of the BJSPT

including recirculation, the technological routes are determined randomly and without recirculation for the jobs in the Lawrence instances. Thus, it can be well argued that internal structures restrict the repairing mechanisms and lead to a higher amount of equivalent feasible permutations for given API or TAPI moves.

The performance of non-equivalent API-L/TAPI-L against API-R/TAPI-R neighbors with regard to the total tardiness of the resulting feasible schedules. This constitutes another aspect, which is represented in the Figures 4.19 and 4.20. The proportion of transitions, for which the left shift transformation (LS) leads to a neighboring solution with smaller to-

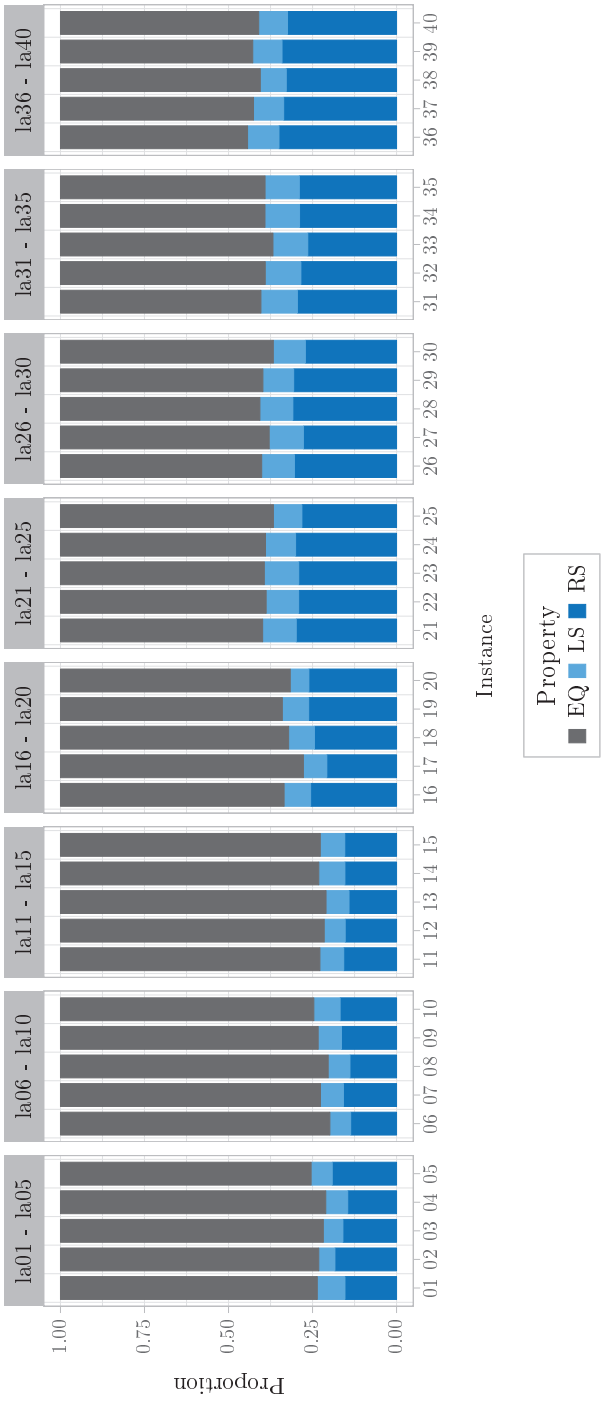


Figure 4.20: Proportion of equivalent solutions and performance of right shift and left shift transformation among all API-based neighbors for the Lawrence instances of the BJSPT

tal tardiness compared to the schedule resulting from right shift transformation (RS), is indicated in light blue. Similarly, the dark blue stacks depict the proportion of moves, for which the right shift transformation creates the better schedule with regard to the objective function. Over all benchmark instances, it can be observed that the right shift transformation leads to a significantly higher proportion of superior neighboring solutions as against the left shift transformation. Considering Figure 4.20 more precisely, this behavior strengthens with increasing problem size. As a consequence, it might be reasonable to exclusively apply the right shift transformation for the purpose of shorter computation time. The proportion of API and TAPI moves, for which an inferior neighboring schedule is constructed when following this idea, is less than 13% for all benchmark instances.

On the mean distance of API-based neighboring solutions. The Figure 4.21 and 4.22 depict the mean distance of all considered API-based neighbors for the train scheduling-inspired and the Lawrence instances, respectively. It can be observed that, even if an API or a TAPI move initially implements $\delta(s, s') = 1$ for a given schedule s and the neighboring permutation s', the mean distance between feasible solutions features $\delta(s, s') > 5$ for all benchmark instances. In line with observations in the literature, this implies that the required repairing procedure for the BJSPT as a highly constrained optimization problem causes a significant increase in the distances between feasible neighboring schedules, cf. [91]. In Figure 4.22, the Lawrence instances are visually grouped and indicated by green shadings for $m = 5$, blue colors for $m = 10$ and purple for $m = 15$. Evaluating these categories of instances, the increase in the mean distance for a growing number of jobs is considerably more distinct for the Lawrence instances involving ten machines (blue) compared to the problems featuring $m = 5$ (green). Thus, both properties, the number of machines m and the number of jobs n, empirically show a multiplicative effect on the mean distance of feasible neighboring schedules.

The distribution of the distance measures in API-based neighborhoods. By means of boxplots, the distribution of the distance of API-based neighbors is visualized in Figure 4.23 for the train scheduling-inspired instances and in Figure 4.24 for the Lawrence instances. As commonly defined, the box

represents the interquartile range of the distances, in which 50% of the measures can be found, and the horizontal line inside the box indicates the median. The whiskers depict the minimum and maximum measure that is not more than one and a half interquartile ranges far from the box. Over all benchmark instances, it can be observed that the occurring range of distances significantly increases with growing problem size, while the median is not highly affected. Considering the groups of Lawrence instances with different quantities of machines (green, blue and purple) in Figure 4.24, the dependence of the distance distribution on the number of jobs n and the number of machine m and the multiplying effect can clearly be seen. In comparing the boxplots for the instances ts01 to ts15 and la16 to la30 of similar problem size and equal coloring, it is remarkable that the distribution of the distance of the Lawrence instances seems to be more variable for instances featuring the same number of jobs and machines. This implies that the internal structure of the train scheduling-inspired instances leads to a similar neighborhood pattern for problems of the same size. This aspect may be beneficial when creating heuristic solution methods with a desired robust performance.

On the distance distribution of non-equivalent API-based neighbors generated by left shift and right shift transformation. Figure 4.25 comparatively depicts the distributions of the distances of non-equivalent API-L and API-R neighbors for the train scheduling-inspired instances. Similarly, boxplots on the distances of non-equivalent TAPI-L and TAPI-R neighbors for the Lawrence instances are shown in the Figures 4.26 and 4.27. Since the following aspect is equivalently observable for both neighborhoods on both classes of problems, the corresponding complementary graphs will not be given explicitly here. It can be noted that, for all benchmark instances and both neighborhoods, the distances of neighbors generated by right shift transformation and repair are significantly smaller compared to the feasible neighbors constructed by left shift transformation. This may be caused by a higher number of required additional changes in the operation sequences on the machines when applying the ART to an infeasible permutation involving a left shifted operation. Since the API-based neighborhoods are intended to intensify the search around local optima in less distant steps and in favor of a structured and controllable search process,

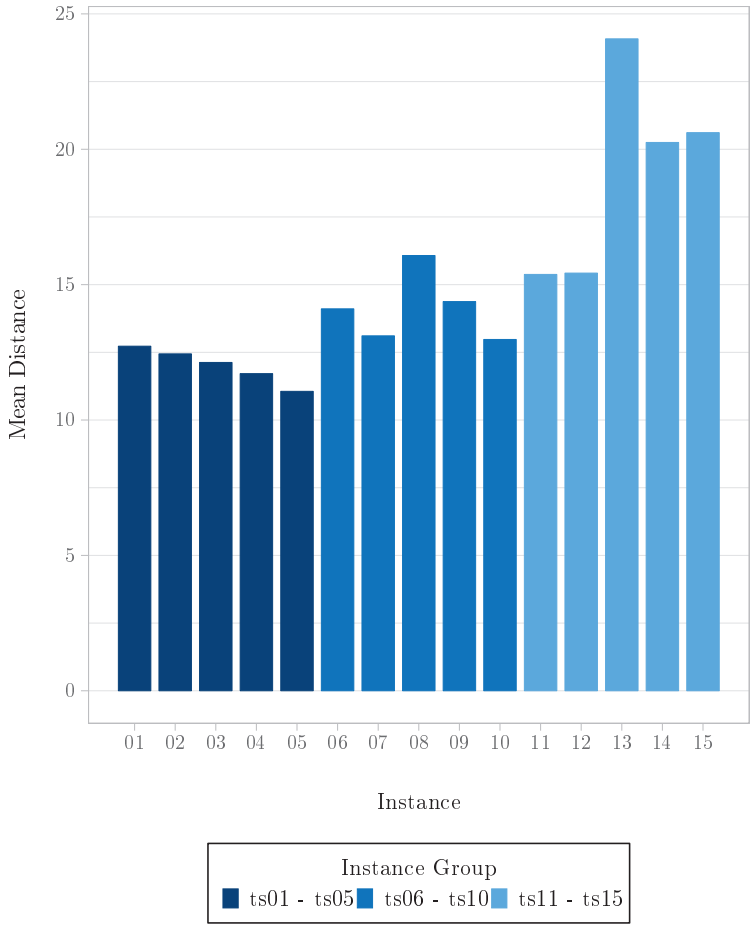


Figure 4.21: Mean distance of all API-based neighbors for the train scheduling-inspired instances of the BJSPT

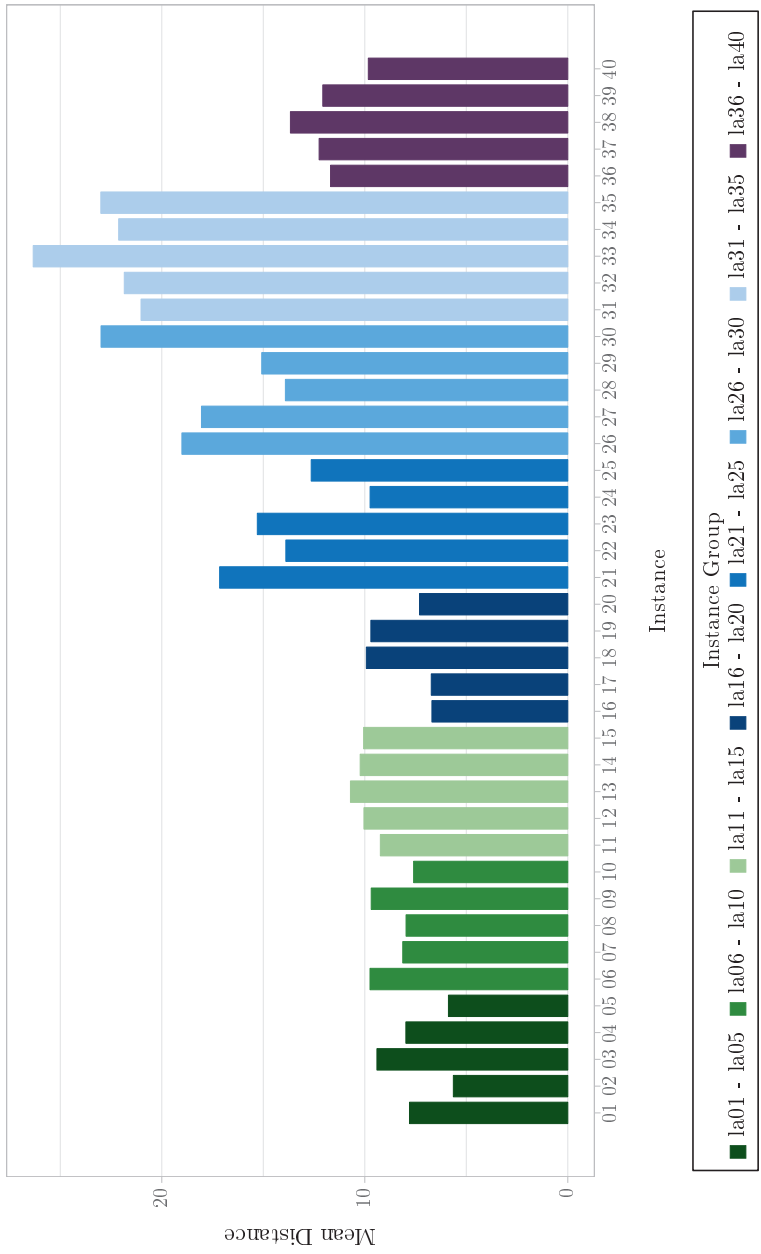


Figure 4.22: Mean distance of all API-based neighbors for the Lawrence instances of the BJSPT

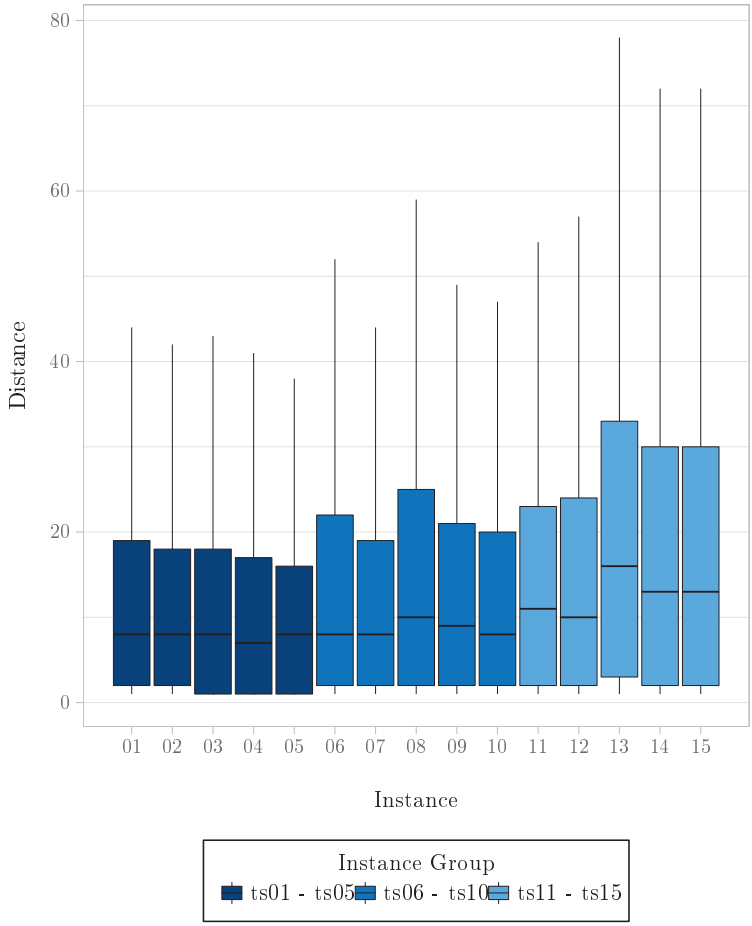


Figure 4.23: Boxplots representing the distribution of the distance measure in the API-based neighborhoods for the train scheduling-inspired instances of the BJSPT

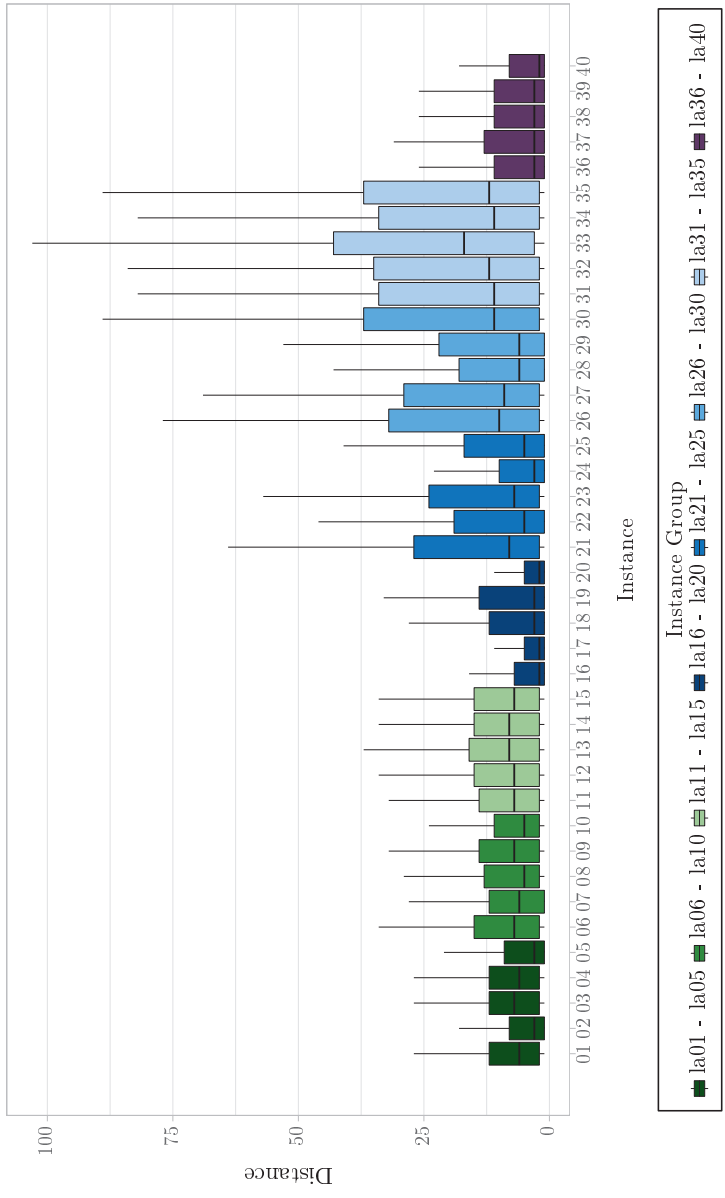
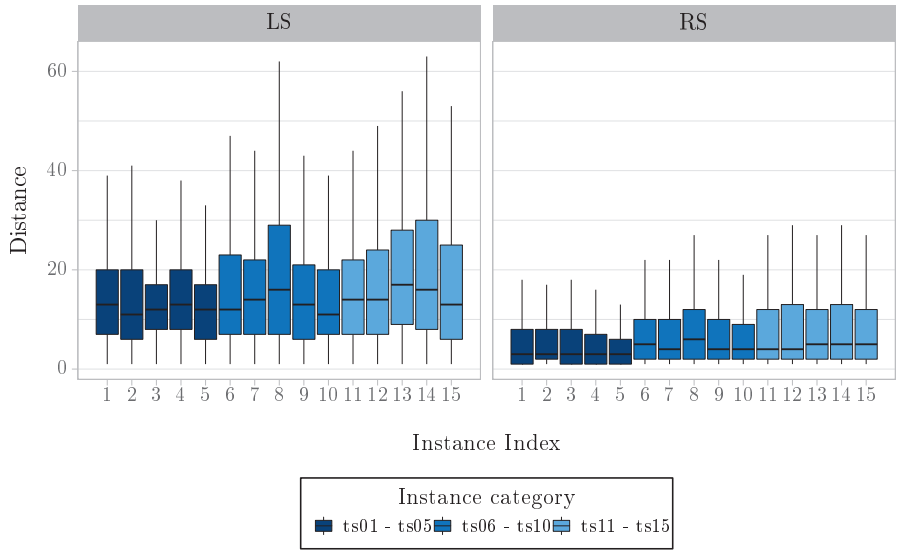
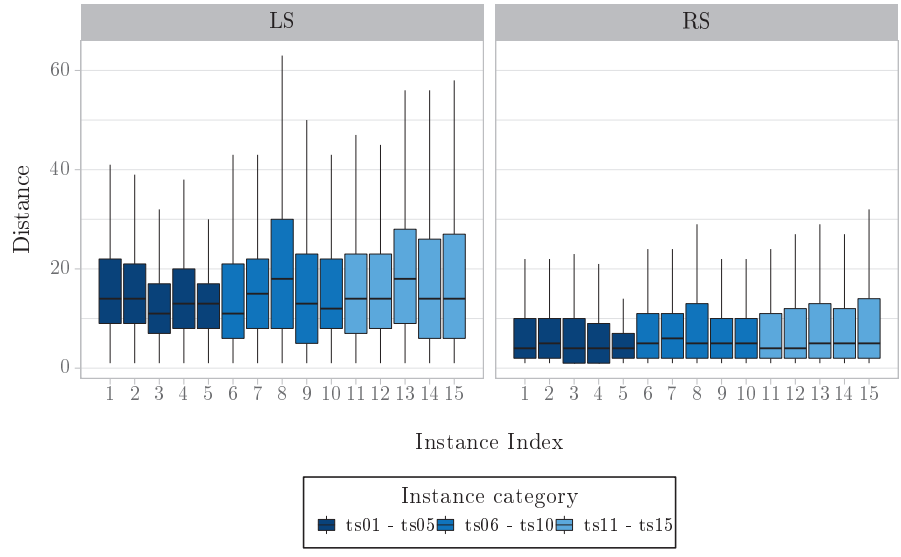


Figure 4.24: Boxplots representing the distribution of the distance measure in the API-based neighborhoods for the Lawrence instances of the BJSPT



(a) Distribution among API-based neighbors



(b) Distribution among TAPI-based neighbors

Figure 4.25: Boxplots representing the distribution of the distance measure among neighbors based on API-moves for the train scheduling-inspired instances of the BJSPT

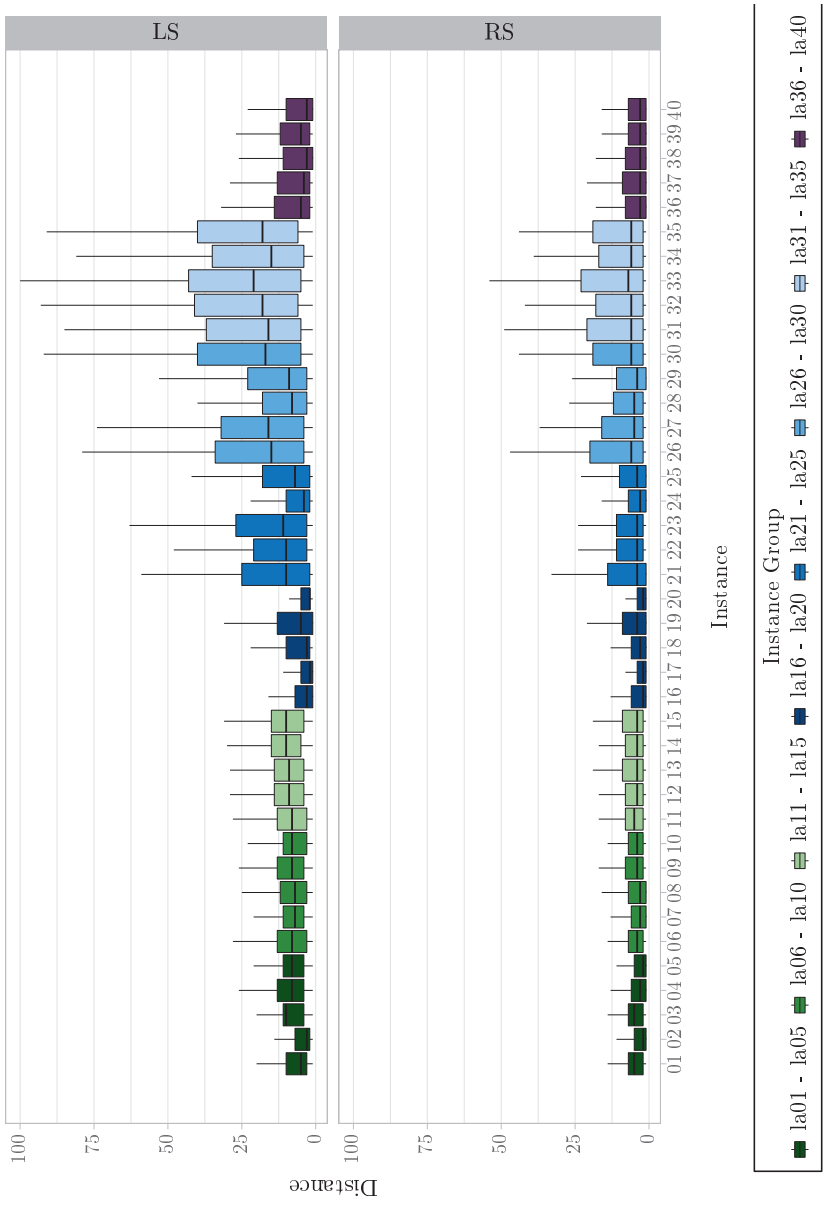


Figure 4.26: Boxplots representing the distribution of the distance measure among API-based neighbors for the Lawrence instances of the BJSPT

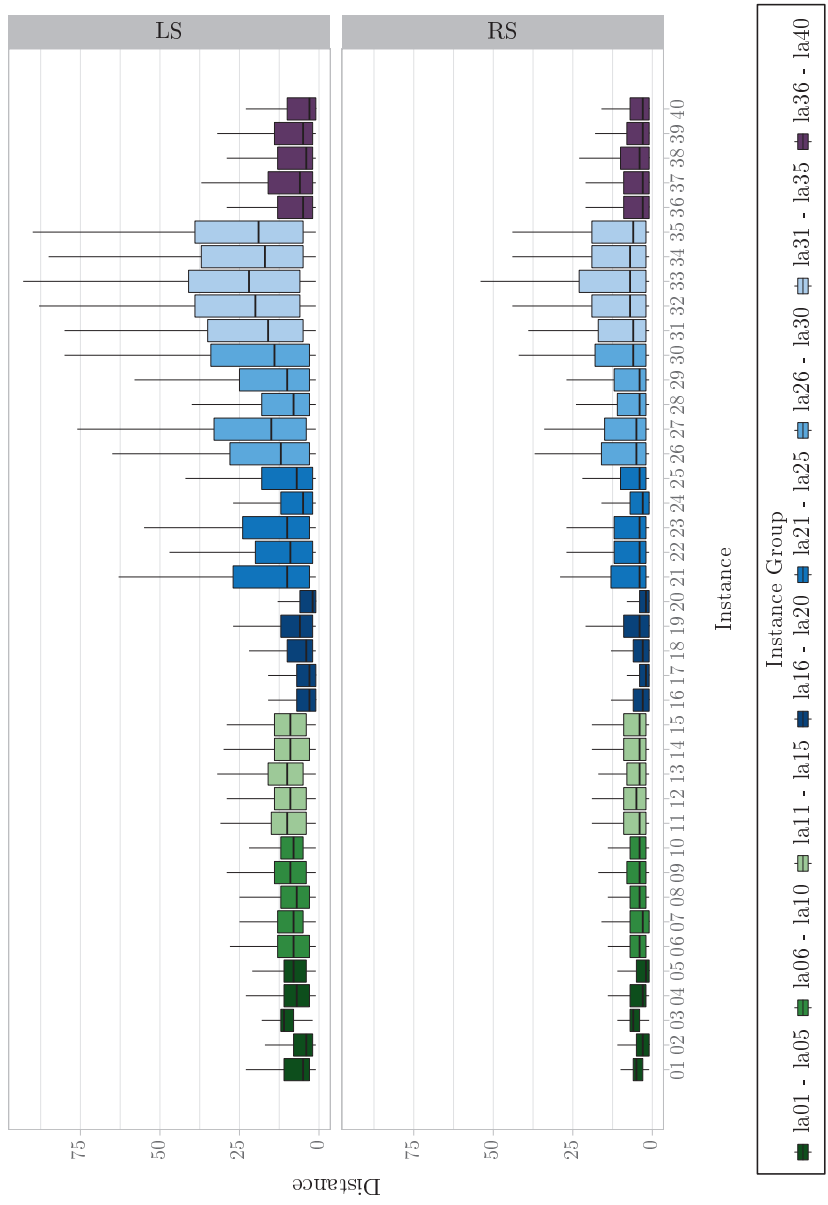


Figure 4.27: Boxplots representing the distribution of the distance measure in the TAPI-based neighbors for the Lawrence instances of the BJSPT

the application of right shift transformation is to be preferred to implement an API or a TAPI move in the operation-based representation of a neighboring schedule.

4.8 Local Neighborhood Search

One of the most generic metaheuristic methods, namely the local descent or iterative improvement scheme, cf. for instance [32], is described and implemented in this section. It is intended to test the API and the TAPI neighborhood for the BJSPT and to generate initial experimental results on their performance. In the subsequent section 4.8.1, the iterative improvement scheme and its implementation are technically explained. The computational results obtained by the LS for all benchmark instances of the BJSPT are presented and discussed in Section 4.8.2.

4.8.1 The Iterative Improvement Scheme

Algorithm 4 describes the basic structure of the applied neighborhood search method, cf. [32], where $\mathcal{N}(s)$ denotes a general neighborhood and $T(s)$ specifies the total tardiness of a given schedule s. The initial solution is determined as the best out of ten randomly generated schedules in line 2. The condition of the **while**-loop starting in line 4 incorporates the three termination criteria that are implemented next to the direct **return** of the locally optimal solution when no better or equivalent neighbor $\tilde{s}$ is found in $\mathcal{N}(s)$, see line 18. The iterative improvement scheme will be stopped, if

- a runtime limit of two hours is reached,
- the total number of generated neighbors (NumNb) exceeds 65000 or
- more than ten iterations without improvement (EqCount) are operated.

As given in the lines 5 to 10 of Algorithm 4, the procedure implements a *best fit strategy*, where the entire neighborhood of the current schedule s is set up and evaluated. Consequently depending on the total tardiness, the best neighbor $\tilde{s}$ is chosen to be the new incumbent solution s or the iterative improvement scheme terminates.

Algorithm 4 Iterative Improvement Scheme

Input: instance input data
Output: locally optimal solution s^*

```
1: s* ← ∅, NumNb ← 0, EqCount ← 0
2: s ← best of 10 generated solutions with priority rule RANDOM
3: determine T(s)
4: while timelimit < 7200s AND NumNb ≤ 65000 AND EqCount ≤ 10 do
5:     set up N(s)
6:     for s' ∈ N(s) do
7:         determine T(s') and store
8:         NumNb ← NumNb +1
9:     end for
10:    s̃ ← solution with T(s̃) = min{T(s') | s' ∈ N(s)}
11:    if T(s̃) < T(s) then
12:        s ← s̃
13:    else if T(s̃) = T(s) then
14:        s ← s̃
15:        EqCount ← EqCount +1
16:    else
17:        s* ← s
18:        return s*
19:    end if
20: end while
21: s* ← s
22: return s*
```

4.8.2 Computational Results

The benchmark instances introduced in Section 2.5 are solved by applying the iterative improvement scheme implemented according to Algorithm 4 in Python 3 and running on a notebook, which operates an Intel Dual Core i5 processor (2.20 GHz) with 8 GB RAM. For each run of the LS, the general neighborhood $\mathcal{N}(s)$ is specified by either the API neighborhood $\mathcal{N}_A(s)$ or the TAPI neighborhood $\mathcal{N}_T(s)$ described in Section 4.7.4. In detail, an API or TAPI move is performed on the incumbent schedule s and both neighboring solutions resulting from the right shift and the left shift transformation are set up. In case that the corresponding feasible

permutations are not equivalent, the schedule featuring the least total tardiness determines the neighbor $s' \in \mathcal{N}(s)$. This preselection can be done without loss of generality, since an inferior solution will never be chosen as the best neighbor $\tilde{s} \in \mathcal{N}(s)$.

Tables 4.1 and 4.2 as well as 4.3 and 4.4 show the results for five independent runs for each neighborhood and each class of instances, respectively. The first two columns indicate the considered instance and the best objective function value obtained by solving the MIP formulation. As in Section 3.5, proven optimal solutions are denoted by an asterisk. Besides, the group of four columns presents the mean total tardiness $\overline{\sum T_i}$ and the minimum total tardiness $\min(\sum T_i)$ reached by the LS applying the corresponding neighborhood structure, the mean number of iterations $\overline{\#(\text{It.})}$ and the mean runtime in seconds.

An iteration of the iterative improvement scheme is defined by a single execution of the **while**-loop (lines 5 to 19) of Algorithm 4. In the computational results, it can be observed that the mean number of iterations seems to be solely dependent on the size of the instance and independent of the chosen neighborhood. Similarly, the runtime of the LS features a stronger dependence on the number of iterations than on the underlying transition scheme. Thus, the advantageousness of one or the other API-based neighborhood for the BJSPT can be evaluated exclusively based on the mean and minimum objective function values reached. Comparing on the one hand the averaged total tardiness values for the train scheduling-inspired instances shown in Tables 4.1 and 4.2, the best fit strategy on the API neighborhood leads to better results for most of the problems. Considering on the other hand the Lawrence instances in Tables 4.3 and 4.4, none of the neighborhoods can be stated as favorable against the other by means of the averaged objective function values. Consequently, a beneficial reduction of the search space by means of total tardiness cannot be observed in the application of LS.

Nonetheless, especially the results obtained for the smallest instances ts01 to ts05 reason the application of API-based neighborhoods for the BJSPT, since there are medium quality schedules obtained in short computation time. Observing the remarkable differences between the mean and the minimum total tardiness values reached by LS gives empirical evidence for

Table 4.1: Computational results of the Iterative Improvement Scheme with API neighborhood applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	API neighborhood			
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\#(\text{It.})}$	Time
ts01	138*	167.8	146	14.2	3.72
ts02	90*	124.0	103	16.0	2.60
ts03	72*	102.2	83	17.6	3.28
ts04	41*	65.8	41*	13.8	3.19
ts05	71*	89.4	72	14.0	1.71
ts06	88*	242.8	219	18.0	6.93
ts07	172*	335.6	290	21.4	9.51
ts08	163*	292.6	256	18.2	7.46
ts09	153*	266.6	226	21.6	8.81
ts10	97*	294.8	238	17.0	6.89
ts11	366	683.0	581	21.2	17.88
ts12	419	793.4	678	21.6	18.65
ts13	452	752.4	702	21.6	21.56
ts14	459	715.6	636	18.6	18.30
ts15	418	611.8	525	22.4	21.99

Table 4.2: Computational results of the Iterative Improvement Scheme with TAPI neighborhood applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	TAPI neighborhood			
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\#(\text{It.})}$	Time
ts01	138*	170.4	138*	14.4	2.01
ts02	90*	131.6	113	10.4	1.45
ts03	72*	108.0	76	12.2	1.70
ts04	41*	78.0	42	11.6	1.92
ts05	71*	129.2	85	10.4	1.70
ts06	88*	220.8	178	19.2	7.06
ts07	172*	405.8	345	16.0	8.42
ts08	163*	338.4	284	17.2	8.02
ts09	153*	326.2	233	11.8	4.71
ts10	97*	302.2	237	19.0	8.26
ts11	366	699.4	559	18.2	15.18
ts12	419	640.2	587	21.4	18.48
ts13	452	807.2	625	20.2	19.39
ts14	459	813.0	595	17.4	15.73
ts15	418	629.0	583	21.0	20.18

Table 4.3: Computational results of the Iterative Improvement Scheme with API neighborhood applied to the Lawrence instances of the BJSPT

Inst.	MIP	API neighborhood			
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\#(\text{It.})}$	Time
la01	762*	2221.6	1401	10.8	0.58
la02	266*	951.8	488	13.6	0.72
la03	357*	1967.4	1263	11.0	0.60
la04	1165*	2066.0	1605	8.6	0.62
la05	557*	1145.6	1000	11.0	0.63
la06	2516	5791.4	5187	10.4	1.89
la07	1677*	4554.2	3671	16.4	2.86
la08	1829*	5473.2	3253	11.4	1.89
la09	2851	6118.6	4722	11.4	2.12
la10	1841*	5288.4	3350	9.8	1.89
la11	6534	11745.8	10630	16.6	5.56
la12	5286	12035.0	9810	18.8	6.93
la13	7737	13613.8	12030	14.2	5.20
la14	6038	11208.0	9289	16.4	5.81
la15	7082	11955.2	9738	18.6	6.18
la16	330*	1527.2	1043	32.2	8.11
la17	118*	1505.6	475	32.6	7.84
la18	159*	2788.2	835	21.4	6.56
la19	243*	3349.4	1704	15.4	4.89
la20	42*	2755.4	389	23.8	6.68
la21	1956	7489.4	7015	25.2	23.16
la22	1455	6262.4	4364	35.2	27.43
la23	3436	9190.4	6956	17.6	15.98
la24	560*	7504.8	5781	25.6	19.56
la25	1002	6118.8	3795	35.6	27.02

continued on the next page

Inst.	MIP	API neighborhood			
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\#(\text{It.})}$	Time
la26	7961	16417.2	13580	25.4	50.75
la27	8915	15886.8	13745	32.6	61.50
la28	2226	11907.4	8298	44.6	75.89
la29	2018	15525.6	10055	41.0	71.25
la30	6655	15151.8	11644	46.2	84.08
la31	20957	45865.6	40261	39.8	248.28
la32	23150	43811.6	35292	43.4	268.85
la33	none	34336.2	29355	45.0	275.03
la34	none	38602.0	36792	49.6	277.48
la35	none	37849.8	32164	51.0	317.49
la36	675	9321.2	6839	46.4	110.19
la37	1070	11648.0	7479	36.6	100.85
la38	489*	10054.6	6078	35.4	89.74
la39	754	10006.0	6769	36.6	104.21
la40	407*	8257.0	6822	45.4	116.57

Table 4.4: Computational results of the Iterative Improvement Scheme with TAPI neighborhood applied to the Lawrence instances of the BJSPT

Inst.	MIP	TAPI neighborhood			
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\#(\text{It.})}$	Time
la01	762*	1912.4	1472	7.6	0.48
la02	266*	1558.8	810	9.8	0.53
la03	357*	890.0	405	15.0	0.87
la04	1165*	2436.4	1672	6.0	0.39
la05	557*	1465.0	1068	9.2	0.50
la06	2516	5268.8	4222	12.8	2.13
la07	1677*	5157.8	2910	12.8	2.15
la08	1829*	4733.2	3767	17.8	2.60
la09	2851	6323.2	4347	13.0	2.04
la10	1841*	5035.0	4467	15.8	2.45
la11	6534	12341.0	10549	17.0	5.52
la12	5286	10729.0	8325	15.4	5.22
la13	7737	11206.8	9284	15.4	5.59
la14	6038	11246.0	9150	14.2	5.01
la15	7082	11876.8	10584	12.8	4.83
la16	330*	1734.2	744	26.0	5.52
la17	118*	965.8	627	28.6	5.49
la18	159*	1445.6	882	25.0	5.97
la19	243*	2889.0	1558	16.0	4.62
la20	42*	1044.4	498	28.2	5.63
la21	1956	8201.2	6100	25.4	22.11
la22	1455	6863.6	5382	28.4	22.04
la23	3436	9222.8	7490	17.4	16.76
la24	560*	6786.4	4441	31.0	23.24
la25	1002	7631.8	3162	28.8	20.75

continued on the next page

Inst.	MIP	TAPI neighborhood			
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\#(\text{It.})}$	Time
la26	7961	17470.2	15116	22.4	44.73
la27	8915	15630.4	13698	32.0	57.06
la28	2226	15673.0	9880	37.6	61.65
la29	2018	12886.0	10960	46.0	72.36
la30	6655	14670.2	13678	38.8	63.79
la31	20957	44430.6	33391	40.6	231.94
la32	23150	42909.2	36749	37.6	234.63
la33	none	36551.8	29299	47.8	274.25
la34	none	40365.2	34186	42.8	260.78
la35	none	45085.2	36731	37.2	224.79
la36	675	9779.4	5962	45.2	115.74
la37	1070	9756.8	6741	44.4	113.15
la38	489*	8438.2	6656	36.6	96.35
la39	754	8849.0	6938	35.4	90.56
la40	407*	10099.0	7037	36.4	93.01

the ruggedness of the search space of the BJSPT and emphasizes beneficial effects of several independent runs of any heuristic procedure. Generally, the results are not satisfactory with regard to solution quality, so that the necessity of an advanced heuristic method is implied.

4.9 Simulated Annealing

To improve the results obtained by the iterative improvement scheme for the benchmark instances of the BJSPT, an SA metaheuristic is described and applied in this section, cf. [79]. This method similarly belongs to the class of local search algorithms, cf. for instance [32] and [102], since it is equivalently based on the implementation of beneficial moves in the neighborhood of a given schedule. Indeed, SA constitutes a generalization of the generic iterative improvement scheme through the incorporation of additional control parameters, cf. [122]. The acceptance of non-improving transitions and the simultaneous incorporation of different neighborhoods expand the search capabilities and play a central role in avoiding the early entrapment of the method in local optima, cf. [13]. Nonetheless, the SA procedure is simple enough to act as a framework in evaluating the performance of neighborhood structures, cf. [76]. The technical outline of the SA and the implemented parameter settings are given in the subsequent Section 4.9.1. The results obtained by SA with different combinations of the proposed neighborhoods are presented and discussed in Section 4.9.2.

4.9.1 The Simulated Annealing Algorithm

SA constitutes a well-known generic metaheuristic, for which various descriptions and experimental analyses are provided in the literature, see for instance [17], [32], [102], [121] and [122]. The algorithm involves a temperature t as the main control feature, which ranges between an initial value t_{init} and a terminal value t_{term}. The temperature evolves during the search process according to a predefined cooling scheme. Based on the temperature t, the probability P of the acceptance of the currently regarded

neighbor s' of a schedule s as the new incumbent solution is determined by

$$P \;=\; \min\left\{1,\; e^{-\frac{T(s')-T(s)}{t}}\right\}.$$

The structure of the SA implemented for the BJSPT in this thesis is technically described in Algorithm 5. Three temperature-related control parameters $[t_{init},\; t_{term},\; c]$ are required as input. Preliminary computational experiments have shown that the total tardiness behaves quite divers among neighboring solutions and that consequently, the application of a distribution-based cooling scheme as shown in [122] leads to a non-beneficial enormous slowdown of the temperature decrease. Thus, a simple geometric cooling scheme $t' = c \cdot t$ with $c \in (0, 1)$ is applied to determine a subsequent temperature level t' from the current temperature t. Since the choice of adequate initial and terminal temperatures strongly depends on the magnitude of the objective function value of the considered instance, different settings are preliminarily tested for the benchmark problems. The following values of the control parameters turn out to be most efficient and the corresponding usage for certain groups of instances is given with the results in Tables 4.5 to 4.12.

Setting 1: [20, 0.5, 0.9925],

Setting 2: [20, 10, 0.999],

Setting 3: [200, 50, 0.995],

Setting 4: [1000, 100, 0.99].

Starting the SA, the initial solution s is generated by the use of the priority rules described in Section 4.6, see the lines 2 to 5 of Algorithm 5. Using this schedule, the best found solution s^* is firstly updated in line 7. As long as the stopping criterion $t \leq t_{term}$ is not fulfilled and the total runtime does not exceed two hours, a sequence of iterations is performed for each temperature level. This can be interpreted as a Markov process reasoning the asymptotic convergence of SA for an infinite number of such iterations, cf. [122]. The **while**-loop in the lines 9 to 20 is repeated $n_{op} - m$ times, which corresponds to the maximum number of choosable API moves provided that all of them exist. In each iteration, a neighborhood is chosen according to a given probability distribution, see

Algorithm 5 Simulated Annealing Algorithm

Input: instance input data, control parameters $[t_{init}, t_{term}, c]$, neighborhood choice probability distribution

Output: locally optimal solution s^*

1: $s^* \leftarrow \emptyset$, $t \leftarrow t_{init}$
2: **for** PrioRule $\in$ {SPT, SL, S/OPN, MOD, ATC, RANDOM} **do**
3: generate and store 10 solutions acc. to PrioRule
4: **end for**
5: $s \leftarrow$ best generated schedule [initial solution]
6: determine $T(s)$
7: **while** $t > t_{term}$ AND timelimit $< 7200s$ **do**
8: chain $\leftarrow 0$
9: **while** chain $\leq n_{op} - m$ **do** [Markov chain per temperature level]
10: choose neighborhood $\mathcal{N}(s)$ acc. to probability distribution
11: $s' \leftarrow$ arbitrary element of $\mathcal{N}(s)$
12: determine $T(s')$
13: determine P
14: $u \leftarrow$ random real number in $(0, 1)$
15: **if** $u < P$ **then**
16: $s \leftarrow s'$, $T(s) \leftarrow T(s')$
17: **end if**
18: **if** $T(s) < T(s^*)$ **then**
19: $s^* \leftarrow s$, $T(s^*) \leftarrow T(s)$
20: **end if**
21: chain $\leftarrow$ chain $+1$
22: **end while**
23: $t \leftarrow c \cdot t$ [geometric cooling scheme]
24: **end while**
25: **return** s^*

line 10. For all runs of the SA performed in this computational study, an API-based neighborhood is applied with a probability of 0.9, while the randomized TJ neighborhood is chosen with a probability of 0.1. This slight incorporation of randomness turned out to be most effective during some preliminary tests on different probability distributions. To compare the performance of the API and the TAPI neighborhood for the BJSPT, these neighborhoods are always exclusively applied either as an API-TJ combination or as a TAPI-TJ pairing. Following the neighborhood choice, an arbitrary feasible neighbor $s' \in \mathcal{N}(s)$ is defined for consideration in line 11. Based on the total tardiness measure $T(s')$ and the corresponding acceptance probability P, the neighboring schedule s' is either set to be the new incumbent solution s or the current incumbent schedule is kept, see lines 12 to 18. At this point, inferior schedules might probably be accepted while improving solutions are always chosen. If the total tardiness of the new incumbent solution is less than the objective function value of the best found schedule s^*, this schedule is updated in the lines 19 to 21. Consequently, the performed iteration increases the number of considered neighbors (chain) at the given temperature level. In case that the maximum length of the Markov chain is reached, the temperature is decreased according to the geometric cooling scheme in line 21 and a new sequence of iterations is started. Once a terminal condition is satisfied, the best found schedule s^* is returned.

4.9.2 Computational Results

The SA is implemented in Python 3 according to the outline in Algorithm 5. Computational tests are run for all benchmark instances introduced in Section 2.5 on a notebook operating an Intel Dual Core i5 processor (2.20 GHz) with 8 GB RAM. For each instance, one or two appropriate settings are determined and both API-based neighborhoods are separately implemented as described in the previous section. Tables 4.5 to 4.12 display the computational results for five independent runs operated for each instance, setting and neighborhood. The first three columns of each table consist of the instance index, the best objective function value provided by MIP in Section 3.5 and the total number of iterations performed by the SA. Further, the average total tardiness $\overline{\sum T_i}$, the minimal total tardiness

$\min(\sum T_i)$ and the mean runtime of the five runs are shown for each instance for the corresponding neighborhood. As before, objective function values with proven optimality are denoted by an asterisk. To evaluate the performance of the API-based neighborhoods against each other, the smaller mean total tardiness measure is highlighted by boldface printing.

Tables 4.5 and 4.6 as well as 4.7 and 4.8 present the computational results for the train scheduling-inspired instances obtained by the SA in the settings 1 and 2, respectively. Compared to Setting 1, the cooling scheme in Setting 2 focuses more on lower temperatures resulting in more critical acceptance probabilities and leads to a higher total number of iterations. Considering the average total tardiness values achieved with both selections of control parameters, the SA in Setting 2 performs slightly better than the SA in Setting 1 in 12 out of 15 instances for both neighborhoods. Since the results do not show a significant difference, especially when regarding the minimum total tardiness values, the advantage of Setting 2 simply seems to be based on the higher amount of iterations and runtime. However, it can be observed that the choice of the setting and the choice of the neighborhood show an interdependence. By means of the average total tardiness, the API neighborhood outperforms the TAPI neighborhood in 10 out of 15 instances operated in Setting 1 (Tables 4.5 and 4.6), while the TAPI neighborhood obtains better average objective function values for 10 out of 15 instances with Setting 2 (Tables 4.7 and 4.8). This gives empirical evidence for the proposition that the TAPI neighborhood interacts beneficially with lower temperature levels while the API neighborhood does with less critical ones. Generally, the results obtained for the train scheduling-inspired instances support the application of API-based neighborhood structures embedded in an SA to generate high quality schedules for the BJSPT. The proven optimal solution is simultaneously determined for seven out of nine instances and the best feasible solution found by MIP is improved for four out of six problems. Finally, note that the given average runtime depicts the computation time for the entire SA procedure, whereby the best schedule is generated at an earlier stage in the search process for most of the instances. Thus, a time limit of a few minutes might be sufficient to obtain schedules of good quality.

The computational results for the Lawrence instances solved by SA in Setting 3 are shown in Tables 4.9 and 4.10. It can be observed that the comparative measures for these randomness-based instances are less conclusive than the results obtained for the structure-related problems. When evaluating the mean total tardiness values, the SA operating with a TAPI neighborhood generates superior schedules for 22 out of 40 instances, while the implementation using the API neighborhood acts advantageous for 16 problems. Especially for the instances of larger size, the difference between the mean total tardiness measures becomes quite small compared to the magnitude of the objective function values, so that the advantageousness of one or the other neighborhood structure may mainly be related to the random components of the search method. Thus, both neighborhoods are well employable to tackle instances of the BJSPT and these results show the successful application of a heuristic method of easy structure to combinatorially hard scheduling problems. Considering the instances la11 to la15 of size $(20, 5)$, la26 to la30 of size $(20, 10)$ and la31 to la35 of size $(30, 10)$ especially, SA improves the best found feasible solutions of the MIP solver for 13 out of these 15 problems in considerably less computation time. Note that the difference between the mean total tardiness measures and the minimal obtained objective function values can equivalently be observed here. Thus, repeated independent runs of the heuristic procedure seem to be advantageous.

As an addition, Tables 4.11 and 4.12 depict the computational results of SA applied with Setting 4 to the instances la31 to la35. Since the magnitude of the total tardiness measures of these problems is significantly higher compared to the other Lawrence instances, the adaptation of the control parameters effects the computation time even more beneficially while schedules of similar quality are obtained. Nonetheless, for some instances of smaller size, SA is not able to meet the MIP results in solution quality and runtime. Thus, this gives rise to the idea of enhancing a heuristic search method with MIP technology and combining the favorable features of both solution approaches.

Table 4.5: Computational results of the SA with API neighborhood in Setting 1 applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	#(It.)	API neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
ts01	138*	23568	**140.0**	138*	294.54
ts02	90*	22586	**95.0**	91	271.68
ts03	72*	23568	**78.8**	72*	268.12
ts04	41*	24550	41.4	41*	334.42
ts05	71*	22586	**71.2**	71*	253.84
ts06	88*	40753	125.0	108	642.30
ts07	172*	41735	**196.0**	184	694.96
ts08	163*	40753	185.6	163*	599.42
ts09	153	41735	**174.0**	160	632.11
ts10	97*	39771	116.6	107	541.47
ts11	366	56956	**409.4**	387	1068.83
ts12	419	57938	**429.2**	412	980.05
ts13	452	58920	492.2	472	1039.72
ts14	459	57938	**500.6**	473	1062.64
ts15	418	58920	433.2	413	1088.04

Table 4.6: Computational results of the SA with TAPI neighborhood in Setting 1 applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	#(It.)	TAPI neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
ts01	138*	23568	142.6	138*	402.59
ts02	90*	22586	96.6	90*	262.61
ts03	72*	23568	84.8	76	349.55
ts04	41*	24550	**41.2**	41*	334.02
ts05	71*	22586	71.6	71*	261.58
ts06	88*	40753	**119.4**	109	610.09
ts07	172*	41735	201.0	192	638.76
ts08	163*	40753	185.6	181	591.97
ts09	153	41735	175.2	161	652.84
ts10	97*	39771	**112.6**	108	557.61
ts11	366	56956	411.8	392	996.59
ts12	419	57938	442.4	419	964.35
ts13	452	58920	**478.2**	445	1047.28
ts14	459	57938	508.8	492	1081.69
ts15	418	58920	**428.2**	387	1014.65

Table 4.7: Computational results of the SA with API neighborhood in Setting 2 applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	#(It.)	API neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
ts01	138*	33264	140.2	138*	384.99
ts02	90*	31878	**94.6**	91	394.39
ts03	72*	33264	**74.2**	72*	396.98
ts04	41*	34650	41.8	41*	437.78
ts05	71*	31878	71.4	71*	404.85
ts06	88*	57519	121.6	107	883.49
ts07	172*	58905	195.4	189	880.14
ts08	163*	57519	**184.2**	179	820.17
ts09	153	58905	178.8	168	921.90
ts10	97*	56133	114.8	97*	767.11
ts11	366	80388	406.4	390	1430.03
ts12	419	81774	428.2	412	1360.35
ts13	452	83160	462.6	448	1497.99
ts14	459	81774	**462.8**	418	1398.62
ts15	418	83160	**419.4**	401	1390.91

Table 4.8: Computational results of the SA with TAPI neighborhood in Setting 2 applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	#(It.)	TAPI neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
ts01	138*	33264	**140.0**	138*	367.36
ts02	90*	31878	95.2	91	420.21
ts03	72*	33264	74.4	72*	377.38
ts04	41*	34650	**41.0***	41*	385.66
ts05	71*	31878	**71.0***	71*	362.21
ts06	88*	57519	**119.8**	111	859.53
ts07	172*	58905	**192.8**	185	897.89
ts08	163*	57519	185.0	181	888.25
ts09	153	58905	**177.4**	174	858.67
ts10	97*	56133	**112.0**	105	774.23
ts11	366	80388	**401.6**	387	1292.55
ts12	419	81774	**424.6**	405	1340.96
ts13	452	83160	**460.6**	447	1543.25
ts14	459	81774	495.0	466	1400.34
ts15	418	83160	435.0	414	1393.00

Table 4.9: Computational results of the SA with API neighborhood in Setting 3 applied to the Lawrence instances of the BJSPT

Inst.	MIP	#(It.)	API neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la01	762*	11080	787.4	773	147.22
la02	266*	11080	283.4	266*	354.58
la03	357*	11080	357.0*	357*	118.16
la04	1165*	11080	**1217.2**	1165*	205.85
la05	557*	11080	557.0*	557*	138.57
la06	2516	18005	**2790.0**	2616	239.06
la07	1677*	18005	1942.2	1869	259.52
la08	1829*	18005	2335.0	1905	204.66
la09	2851	18005	3275.2	3161	202.80
la10	1841*	18005	2178.2	2069	203.90
la11	6534	24930	6186.2	5704	349.30
la12	5286	24930	5070.0	4859	400.51
la13	7737	24930	7850.6	7614	329.37
la14	6038	24930	**6616.8**	5714	379.36
la15	7082	24930	**7088.6**	5626	331.12
la16	330*	22160	395.8	335	428.88
la17	118*	22160	144.2	120	341.19
la18	159*	22160	**229.4**	159*	331.01
la19	243*	22160	306.6	243*	302.85
la20	42*	22160	55.6	42*	330.91
la21	1956	36010	**2847.2**	2101	675.21
la22	1455	36010	**2052.8**	1773	670.79
la23	3436	36010	**3692.6**	3506	650.58
la24	560*	36010	966.8	761	669.17
la25	1002	36010	**1557.4**	1289	667.03

continued on the next page

Inst.	MIP	#(It.)	API neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la26	7961	49860	9275.8	8475	1383.19
la27	8915	49860	**7588.0**	6596	1304.75
la28	2226	49860	3430.8	2876	1253.97
la29	2018	49860	**2948.0**	2432	1273.96
la30	6655	49860	7621.6	6775	1328.15
la31	20957	77560	18921.8	17984	4174.65
la32	23150	77560	21991.4	20401	4779.26
la33	none	77560	**22494.2**	19750	4432.09
la34	none	77560	**20282.8**	18633	4770.32
la35	none	77560	21895.0	18778	4789.15
la36	675	54015	1856.0	1711	2403.28
la37	1070	54015	**1774.2**	1621	2154.29
la38	489*	54015	760.4	645	2298.98
la39	754	54015	**1573.0**	1391	2110.06
la40	407*	54015	**1008.6**	613	2163.05

Table 4.10: Computational results of the SA with TAPI neighborhood in Setting 3 applied to the Lawrence instances of the BJSPT

Inst.	MIP	#(It.)	TAPI neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la01	762*	11080	**783.8**	773	140.23
la02	266*	11080	**277.6**	266*	164.69
la03	357*	11080	357.0*	357*	141.91
la04	1165*	11080	1284.2	1165*	118.56
la05	557*	11080	557.0*	557*	106.27
la06	2516	18005	2912.4	2847	246.24
la07	1677*	18005	**1904.2**	1677*	228.02
la08	1829*	18005	**2129.6**	1829*	190.77
la09	2851	18005	**3226.6**	3131	223.26
la10	1841*	18005	**2119.4**	2046	266.12
la11	6534	24930	**5846.4**	5253	338.76
la12	5286	24930	**4997.8**	4809	365.42
la13	7737	24930	**7611.8**	7342	350.23
la14	6038	24930	6872.4	6459	362.70
la15	7082	24930	7153.6	6330	330.66
la16	330*	22160	**360.8**	335	349.97
la17	118*	22160	**118.8**	118*	359.02
la18	159*	22160	264.0	235	643.07
la19	243*	22160	**301.0**	243*	321.86
la20	42*	22160	**42.0***	42*	298.16
la21	1956	36010	2961.8	2680	757.18
la22	1455	36010	2123.0	1988	667.06
la23	3436	36010	3746.8	3424	692.07
la24	560*	36010	**724.0**	644	781.59
la25	1002	36010	1583.0	1390	701.11

continued on the next page

Inst.	MIP	#(It.)	TAPI neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la26	7961	49860	**8600.8**	7858	1458.37
la27	8915	49860	7641.8	6457	1382.26
la28	2226	49860	**3367.6**	2849	1259.37
la29	2018	49860	3099.0	2626	1296.92
la30	6655	49860	**7372.8**	6395	1321.03
la31	20957	77560	**18409.6**	17751	4217.92
la32	23150	77560	**21632.2**	20546	4688.54
la33	none	77560	22913.2	20553	4756.87
la34	none	77560	21911.8	19577	4669.14
la35	none	77560	**21384.4**	20537	4724.02
la36	675	54015	**1839.0**	1599	2318.08
la37	1070	54015	1835.8	1594	2226.63
la38	489*	54015	**745.4**	676	2118.74
la39	754	54015	1850.2	1551	1889.38
la40	407*	54015	1187.6	912	1928.19

Table 4.11: Computational results of the SA with API neighborhood in Setting 4 applied to selected Lawrence instances of the BJSPT

Inst.	MIP	#(It.)	API neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la31	20957	64400	20198.4	17090	3486.60
la32	23150	64400	22024.8	21092	3532.15
la33	none	64400	23007.2	20135	3454.80
la34	none	64400	**20079.0**	18810	3558.23
la35	none	64400	**21203.0**	19709	3475.94

Table 4.12: Computational results of the SA with TAPI neighborhood in Setting 4 applied to selected Lawrence instances of the BJSPT

Inst.	MIP	#(It.)	TAPI neighborhood		
			$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la31	20957	64400	**18512.6**	15692	3376.26
la32	23150	64400	**20698.4**	19303	3315.97
la33	none	64400	**21874.8**	18949	3319.12
la34	none	64400	20682.8	19734	3249.74
la35	none	64400	22578.4	21314	3278.52

5 Matheuristic Solution Approach for the Blocking Job Shop Problem with Total Tardiness Minimization

The results shown in the previous chapters reveal advantages and disadvantages of the application of pure MIP and permutation-based heuristic solution methods to the BJSPT. While the MIP techniques presented in Chapter 3 are able to obtain optimal solutions in short computation time for small instances, they fail in providing even good feasible solutions for problems of large size. On the contrary, the heuristic methods provided and tested in Chapter 4 will obtain medium quality schedules to arbitrarily large instances, but are not capable of finding the optimal or near-optimal solutions when problem size increases. This chapter presents a hybrid

matheuristic approach to solving the BJSPT, which is developed in a *joint project with Dr. Reinhard Bürgy* from University of Fribourg, Switzerland. First, existing matheuristic methods to solve machine scheduling problems are reviewed in the subsequent Section 5.1. In line with the idea by Balas and Vazacopoulos [13] of combining different resolution approaches to multiply beneficial effects, general and scheduling-tailored construction schemes and neighborhoods, which rely on the mathematical formulation of the BJSPT, are described and evaluated in Section 5.2. Finally, a *Variable Neighborhood Search (VNS)* is performed on the benchmark instances involving promising MIP-based techniques. The setting and the computational results are reported in Section 5.3.

5.1 Existing Matheuristic Approaches to Machine Scheduling Problems

A matheuristic is a hybrid approach, which combines MIP and heuristic techniques most commonly in a master and slave pattern, cf. [39]. Thus, the important decisions to be made in designing a matheuristic method are on which construction schemes, search procedures and neighborhood structures to apply in which hierarchy. It is desired that the implemented techniques feature orthogonal effects to smartly guide the procedure through diverse but promising areas of the search space, cf. [47]. The following examples indicate different realizations of this idea in the field of machine scheduling.

Single and parallel batch machine scheduling problems are tackled with a matheuristic approach by Mönch and Roob in [94]. Such problems naturally consist of two planning levels, which refer to the assignment of batches to and the sequencing of batches on machines in the first stage and the formation of the batches in the second stage. Considering the minimization of the number of tardy jobs, the authors apply a genetic algorithm to obtain good assignments and sequences and consecutively optimize the batch formation based on a mathematical formulation. An unrelated parallel machine scheduling problem with additional resource allocation is treated by Fanjul-Peyro, Perea and Ruiz in [51]. This problem similarly incor-

porates a two-stage decision process, in which the assignment of jobs to machines and the allocation of the limited resource need to be determined. Three matheuristic strategies are tested using two different mathematical formulations with makespan minimization. Compared to the subsequent machine assignment and resource allocation on the one hand and a heuristic reduction of the set of possible assignments before applying an exact solution method on the other hand, a greedy procedure, which sequentially solves the entire problem for subsets of jobs and performs promising fixings, yields the most satisfactory results.

Considering job assignment and sequencing in heterogeneous parallel factories, Behnamian [15] proposes a hybrid heuristic approach, which is guided by the relaxed solution of the mathematical formulation. The master pattern of the solution method consecutively operates an electromagnetism-like local search algorithm and a variable neighborhood search involving seven neighborhood structures. Here, the choice of the applied neighborhood is made based on the result of the linear programming relaxation of the problem. An order scheduling problem on parallel machines, which simultaneously features characteristics of a permutation flow shop and a job shop setting, is treated by Framinan, Perez and Gonzalez [56]. For the purpose of minimizing the total tardiness of all orders, a matheuristic procedure, which iteratively reoptimizes the schedule given a job position oscillation with a fixed maximum position change of an arbitrarily chosen job in the permutation, is applied.

With regard to shop scheduling problems, the flow shop environment constitutes the only class of problems on which matheuristic approaches are tested. Kononova and Kochetov [75] solve the practical application of $F2 \mid\mid C_{max}$ with limited buffers and passive prefetching by a variable neighborhood search technique involving an MIP-based neighborhood structure. The problem $F2 \mid\mid \sum C_i$ is approached by a two-stage matheuristic method by Della Croce et al. [47]. The master pattern consists of a recovering beam search based on job reinsertion, whereby parts of the permutation, called job windows, are repeatedly reoptimized using a mathematical formulation and MIP methods in a subroutine. Furthermore, Ta et al. [116] compare several matheuristic approaches for a permutation flow shop problem $Fm \mid perm \mid \sum T_i$. The applied techniques rely in different proportion

on MIP-based job position oscillation and job window reoptimization and prove to be competitive to a pure genetic algorithm.

To the best of the authors' knowledge, there exists no reported implementation of a matheuristic method for a job shop scheduling problem. Thus, pioneering work is required in evaluating the applicability of different general and scheduling-tailored MIP techniques to design a succeeding hybrid approach. While job position oscillation and sequential scheduling and fixing of subsets of jobs seem to be promising and adaptable ideas from the literature, the exact resolution of separable subproblems and job window reoptimization are not expected to be usable due to the large degree of restrictedness and appearing infeasible solutions of the BJSPT.

5.2 Preliminary Evaluation of MIP-Based Construction Schemes and Neighborhoods

As the overall pattern of the proposed matheuristic for the BJSPT, the VNS is chosen, since it involves a large variety of possibilities to integrate and combine different procedures, see [67]. Generally, the search technique requires an initial solution, for which neighboring solutions are sequentially constructed based on a set of possible transition schemes and evaluated. In this study, all components operated by the VNS are intended to rely on the mathematical formulation MF1 presented with satisfactory results in Chapter 3. Thus, at least one MIP-based construction scheme of feasible solutions needs to be implemented to obtain an initial schedule and a diverse set of promising MIP-based neighborhood structures is supposed to be used. In the following Section 5.2.1, various general-purpose and scheduling-tailored construction schemes of feasible schedules from relaxed linear programming results are described and initial evaluations are reported. Similarly, different MIP-based neighborhoods partially based on scheduling characteristics are discussed with regard to the expected benefits of their incorporation in the VNS in Section 5.2.2.

Preliminary Remark. The *Linear Programming (LP) Relaxation* of the considered mixed-integer problem formulation constitutes a common basis of most of the mathematical model-related construction schemes and

neighborhood structures, see Section 3.4 and Pinedo [102] for basic explanations. Due to the weak LP relaxation of the mathematical models of the BJSPT outlined in Section 3.4, it can be expected that procedures operating with relaxed solutions have difficulties in yielding satisfactory results. In the following, $\mathbf{x}$ is used to indicate the solution vector of the precedence variables obtained by LP or MIP techniques. Accordingly, $\bar{\mathbf{x}}$ denotes the initial or the incumbent feasible solution, which corresponds to a feasible schedule for the BJSPT, and $\mathbf{x}^{LP}$ describes the solution of the LP relaxation of a considered mathematical formulation, which encodes an infeasible schedule in most of the cases. Regarding the transition schemes, the vector $\mathbf{x}^{N}$ indicates a feasible neighboring solution.

5.2.1 Constructive Generation of Schedules

In this thesis, construction schemes particularly refer to stepwise procedures that generate a feasible schedule $\bar{\mathbf{x}}$ for the BJSPT from an infeasible, mainly relaxed solution $\mathbf{x}^{LP}$ or from an empty schedule. In order to represent the variety of existing approaches and detect promising components for the VNS, general-purpose techniques applied to obtain feasible solutions in Branch & Bound procedures are compared to scheduling-tailored methods, which use the involved structures such as jobs and machines. The considered generic MIP-based techniques include

- the Feasibility Pump (FP),
- Fix and Dive (FD) and
- Fix and Solve (FS),

while the scheduling-related methods cover

- Sequential Job Insertion (SJI),
- Sequential Machine Scheduling (SMS) and
- Fix one Machine and Solve (FMS).

These procedures are briefly explained in the following and their individual performance on the benchmark instances of the BJSPT is summarized. All techniques are implemented in Java and applied in five independent runs to each problem using MF1 with precedence variables $y_{i,j,i',j'}$ for

$O_{i,j}, O_{i',j'} \in \Omega^k, M_k \in \mathcal{M}$ featuring $i < i'$. The experiments are performed with a time limit of 30 minutes each on a notebook operating an Intel Dual Core i5 processor (2.20 GHz) with 8 GB RAM. As the MIP Solver, Gurobi Optimizer 7.5.2 is used due to coding reasons. Initial tests have shown that similar results are to be expected when running IBM ILOG CPLEX 12.8.0.

Fischetti, Glover and Lodi [52] proposed the *Feasibility Pump (FP)* to approach the NP-hard problem of finding a feasible solution to an integer or mixed-integer mathematical model with linear constraints and objective function. The key idea is the combination of two solution vectors, which feature infeasibility with regard to different aspects. A solution $\mathbf{x}^{LP}$ is obtained satisfying the linear constraints but discarding integrality. Based on this, a solution $\tilde{\mathbf{x}}$ is derived from $\mathbf{x}^{LP}$ by generic rounding satisfying integrality but most often violating the linear constraints. In every pumping cycle, a new fractional solution $\mathbf{x}^{LP}$ with minimal distance (L_1-norm) to the current integer solution $\tilde{\mathbf{x}}$ is computed by solving a linear program and the incumbent $\tilde{\mathbf{x}}$ is updated by rounding the new $\mathbf{x}^{LP}$. These steps are repeated until a pair $\tilde{\mathbf{x}}$ and $\mathbf{x}^{LP}$ with a minimal distance of zero is found and the current integer solution $\tilde{\mathbf{x}}$ constitutes a feasible schedule for the BJSPT.

Applied to the benchmark instances, FP obtains feasible solutions after executing two pumping cycles for each problem. The constructed feasible solution vectors are of unsatisfactory quality, since they constitute permutation schedules, which involve all the jobs consecutively inserted in reverse index ordering. Evidently, the result of the LP relaxation found by the MIP solver is so close to the permutation schedule, that this is the only feasible solution ever immediately determined. Short computation times can be named as an advantage of this method.

Fix and Dive (FD) constitutes one of the standard techniques implemented in state-of-the-art commercial solvers to find feasible solutions to mixed-integer programs within a Branch & Bound solution method, see Bixby et al. [21]. Considering the relaxed solution $\mathbf{x}^{LP}$ of the blocking job shop problem formulation, the involved precedence variables are sorted according to increasing difference from binarity. A fixed portion α of fractional variables with smallest differences is set to their nearest binary value.

Given these fixed integers, the linear relaxation is solved again, and sorting and value determination are repeatedly applied. The procedure iterates until a feasible schedule is found or the infeasibility of the partially fixed solution is detected.

The experiments are performed on the benchmark instances comparing different portions $\alpha \in \{0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5\}$. While the number of iterations decreases as expected with increasing values of α, the solutions are not affected by the measure of the portion. Equivalently to FP above, FD constructs permutations schedules for all considered instances of the BJSPT in all five runs.

Fix and Solve (FS) is a simple strategy applied in a convergent heuristic for mixed-integer programs proposed by Wilbaut and Hanafi [130]. A feasible solution $\bar{\mathbf{x}}$ is constructed based on the solution of the LP relaxation by fixing all variables in $\mathbf{x}^{LP}$, which have turned out to be binary spontaneously, and running an exact MIP solution approach on the reduced mixed-integer model.

The method is applied to the BJSPT instances and the evaluation shows two main aspects. On the one hand, FS performs well for problems of small and medium size, especially for instances featuring an inner structure like the train scheduling-inspired ones, since the procedure obtains feasible schedules with less than 50% gap compared to the best objective function values found by exact MIP solving, see Section 3.5, in short computation time. On the other hand, FS is not favorable for medium and large problems involving randomized structures, since it is not clear which proportion of variables will be fixed in the first step and the resulting mixed-integer program might still be to complex to be solved by the exact method. Accordingly, FS does not obtain any feasible solution within 30 minutes of computation time for half of the Lawrence instances.

The first tested scheduling-tailored construction scheme is the *Sequential Job Insertion (SJI)*, which is based on a combinatorial job insertion method for the no-wait job shop problem introduced by Bürgy and Gröflin [35], the generic insertion technique proposed by Werner and Winkler [129] and general idea of the NEH heuristic for the permutation flow shop problem, cf. [95]. Starting from an entire LP relaxation of the model for the BJSPT,

which is interpreted as an empty schedule, the integrality of subsets of precedence variables is stepwise reintroduced and relaxed mixed-integer programs are repeatedly solved. According to a randomly generated insertion ordering, the sequence-defining variables related to x chosen jobs are set to be binary and an optimal partial schedule is determined and fixed in every iteration. Thus, the schedule is expanded by single or groups of jobs until a feasible solution $\bar{\mathbf{x}}$ for the BJSPT is found.

Since the exact solution of a highly constrained job shop problem might require long computation time even if there are only two or three jobs involved, cf. [31], the method is implemented with $x \in \{1, 2, 3\}$. This is referred to as SJIx in the following. All three variants of the SJI generate medium quality schedules for all benchmark instances in less than five minutes, while the computation time slightly increases with growing x. Especially for instances of larger size, it cannot clearly be observed that a higher number x of simultaneously inserted jobs improves the solution quality. Hence, the choice of an advantageous insertion sequence seems to have a significant impact, so that several independent runs of SJI featuring a small value of x are favorable against one time consuming run of SJI with a larger x measure.

As an adaption of the idea of the shifting bottleneck procedure proposed by Adams, Balas and Zawack [2], *Sequential Machine Scheduling (SMS)* is applied to construct feasible schedules. Similar to SJI, the LP relaxation of the mathematical formulation of the BJSPT is taken as the starting point. According to a randomly generated machine sequence, the precedence variables related to the currently considered machine are set to be binary in every iteration and the partial schedule obtained by solving the relaxed mixed-integer program is fixed. Consequently, the schedule is stepwise constructed until a feasible solution $\bar{\mathbf{x}}$ for the BJSPT is found or a contradiction in the fixed operation sequences on the machines is detected.

Since the problem under study is highly constrained, SMS terminates with contradicting operation sequences and without a feasible schedule for a vast majority of the benchmark instances. Especially for the train scheduling-inspired problems featuring internal structures, the method is only able to detect a feasible solution for one out of the 15 instances. Furthermore, the initial single machine scheduling problem $1 \mid r_i \mid \sum T_i$ constitutes a

challenging optimization problem itself, so that an exact MIP approach requires a considerable amount of computation time to even fix the operation sequence on the first machine for large instances.

Fix one Machine and Solve (FMS) is introduced here as a scheduling-tailored counterpart of FS with the intention to overcome the issue of contradicting operation sequences. Equivalently to SMS, the procedure starts by solving a relaxed mixed-integer program for $1 \mid r_i \mid \sum T_i$, where only the precedence variables related to a particular, randomly chosen machine are required to by binary. Considering the resulting operation sequence as fixed, the entire reduced BJSPT is solved by an exact MIP approach to obtain a feasible schedule $\bar{\mathbf{x}}$.

The results obtained by FMS are of high quality especially for the train scheduling-inspired instances featuring structured patterns. For half of the considered benchmark problems, FMS generates the best feasible schedule among all construction schemes, but it features a significant disadvantage. Regarding the majority of the runs, FMS is terminated due to the time limit of 30 minutes. Thus, the considerable amount of computation time seems to reason the solution quality rather than the method itself. Furthermore, there exist four of the Lawrence instances, where neither a solution for the initial single machine problem could be obtained nor a feasible schedule for the BJSPT.

To summarize, the generic methods FD and FP captivate due to constructing feasible schedules for all benchmark instances in impressively short computation times, but have a disadvantage in solely generating a job index-based permutation schedule with low solution quality. Thus, these schemes can easily be substituted by a combinatorial technique, which returns arbitrary sequences of the jobs and correspondingly different permutation schedules. FS may be a beneficial construction procedure for small instances featuring internal structures, while the required computation times of 10 to 30 minutes might not be reasonable for the determination of an initial feasible solution of a superior heuristic method. SJIx turns out to be the most promising construction scheme in the preliminary experiments. All three variants yield feasible schedules for all benchmark instances of convincing solution quality in short computation time. Dependent on the overall solution method, the trade-off between objective

function value and runtime can be controlled by an appropriate choice of the parameter x and multiple independent runs of the SJI. On the contrary, SMS constitutes the worst performing construction schemes for the BJSPT due to significant feasibility issues. FMS, similar to FS, seems to be favorable for small and medium size instances, if 30 minutes of computation time are acceptable within the aggregated solution method.

5.2.2 Neighborhood Structures

In the following, six different MIP-based transition schemes are explained and their advantageousness in finding optimal or near-optimal schedules for the BJSPT is briefly discussed. On the one hand, two neighborhoods are generically set up to find good feasible solutions to a mixed-integer program and purely rely on the mathematical formulation of the problems, namely

- Local Branching (LBR) and
- the Relaxation Induced Neighborhood Search (RINS).

On the other hand, four neighborhood structures utilize given characteristics such as jobs and machines. They include

- Job Insertion (JI),
- Job Insertion with Flexibility (JIF),
- Local Position Changes (LPC) and
- Machine Fixing (MF).

All these neighborhoods are individually tested in a simple iterative improvement scheme as described in Section 4.8.1. Dependent on the particular determination of neighboring solutions, either the best neighbor is directly generated by optimally solving a relaxed or reduced mixed-integer program or all possible neighboring solutions are explicitly constructed by an MIP technique and the best neighbor is chosen by comparison. For every benchmark instance, six initial feasible solutions are provided, which involve three different permutation schedules of low quality and another three schedules with medium solution quality constructed by SJI1, SJI2 and SJI3, respectively. Five independent runs are performed for each prob-

lem and initial solution taking a time limit of ten minutes. The neighborhood structures are implemented in Java and the experiments are conducted using a notebook featuring an Intel Dual Core i5 processor (2.20 GHz) and 8 GB RAM.

The first generic neighborhood structure *Local Branching (LBR)* is proposed by Fischetti and Lodi in [53]. It involves a feasible incumbent solution $\bar{\mathbf{x}}$ and a distance parameter $\gamma \in \mathbb{Z}_{>0}$, which can be interpreted as the neighborhood radius, cf. [43]. For every feasible schedule $\bar{\mathbf{x}}$, a γ-neighborhood is defined by the set of feasible neighboring solutions $\mathbf{x}^N$ that satisfy the local branching constraint $\delta(\bar{\mathbf{x}}, \mathbf{x}^N) \leq \gamma$, see Section 4.3 for explanations on the distance measure. By solving the MIP model of the BJSPT and taking the given inequality into account, the best neighboring schedule in the γ-neighborhood is determined in every iteration. Note that this corresponds to finding the best neighbor $\mathbf{x}^N$, which does not require more than γ APIs to be obtained from $\bar{\mathbf{x}}$.

For the purpose of investigating the trade-off between neighborhood size and computation time, the neighborhood structure is tested with $\gamma \in \{5, 10, 15\}$, referred to as LBR5, LBR10 and LBR15, respectively. The runtime of the procedure increases with growing values of the distance parameter γ and is additionally affected by the size of the considered instance and the quality of the incumbent feasible solution. For small instances, it can clearly be observed that the objective function values of the locally optimal schedules improve with increasing distance γ. For medium and large instances, contrarily, the solution quality is almost equally spread among the variants of LBR. Thus, randomized components seem to gain an impact as well as the fact that LBR5 may outperform LBR15 due to shorter computation time of single iterations and a correspondingly higher number of conducted iterations within the time limit. Generally, LBRγ yields satisfactory results and improves almost all initially given schedules for the benchmark instances.

The *Relaxation Induced Neighborhood Search (RINS)* constitutes the second general neighborhood scheme, which combines an incumbent feasible solution $\bar{\mathbf{x}}$ and the solution vector $\mathbf{x}^{LP}$ of the LP relaxation of the mathematical formulation. Danna, Rothberg and Le Pape present promising results of this technique for a job shop problem without blocking constraints

and recirculation in [42] and introduce RINS generally in [43]. During every iteration of the search, the feasible schedule $\bar{\mathbf{x}}$ and the relaxed solution $\mathbf{x}^{LP}$ are compared and variables, which take equivalent values, are fixed. Subsequently, a reduced mixed-integer program of the BJSPT is exactly solved to determine a new feasible solution $\mathbf{x}^{N}$. If the resulting schedule $\mathbf{x}^{N}$ is superior to the incumbent feasible solution, $\bar{\mathbf{x}}$ is replaced by the neighboring schedule and the comparison to $\mathbf{x}^{LP}$ is repeatedly performed.

The preliminary experiments show that RINS features higher potential with initial solutions of lower quality, here the permutation schedules. Comparing LBR and RINS in operating on feasible schedules generated by SJIx, the resulting solution quality is similar while RINS is terminated more often by the time limit and does not obtain an improved schedule for some of the large Lawrence instances. Considering the results of the iterative improvement scheme initialized with permutations schedules, RINS outperforms LBR in solution quality, but requires the full computation time of ten minutes for all medium and large size instances.

As the first proposed scheduling-tailored neighborhood, *Job Insertion (JI)* modifies and applies the idea of the SJIx construction scheme. Given an initial feasible schedule $\bar{\mathbf{x}}$, a neighboring solution $\mathbf{x}^{N}$ is determined by extracting and optimally reinserting x jobs. Thus, all variables related to the arbitrarily chosen jobs are set to be unknown, while the values of the remaining variables and the ordering of the remaining jobs and operations are correspondingly preserved. A reduced mixed-integer program is solved in the construction of every neighboring schedule so that the optimal positions for reinsertion are determined. The neighborhood is implemented in a standard best fit strategy. Thus, the complete JIx-neighborhood of the incumbent feasible schedule $\bar{\mathbf{x}}$ is determined by reinserting all possible subsets of x jobs from the job set and the best neighbor replaces the incumbent schedule, if the objective function value is improved.

In accordance to the preliminary observations on the hardness of the reduced mixed-integer programs, the JI neighborhood is applied with $x \in \{1, 2, 3\}$, referred to as JI1, JI2 and JI3, respectively. The neighborhood structure performs very well on all benchmark instances with both classes of initial solutions. For the medium quality starting schedules, the obtained objective function values improve with increasing number of jobs

x to be reinserted. While JI1 shows a similar performance as the generic methods LBRγ and RINS, JI2 and JI3 clearly outperform these techniques. With regard to the initial permutation schedules, this behavior can only be observed for small instances. Especially for the Lawrence instances of medium and large size, JI1 generates the best locally optimal schedules among the insertion-based neighborhoods. This might again be reasoned by the fact that the computation time for the construction of a single neighbor is short with $x = 1$, and a higher number of iterations can be conducted until the time limit is reached. Generally, the computation time of JIx increases with growing measure of x, with a higher number of jobs and machines involved in the instance and with lower quality of the initial solution. JI3 shows a similar runtime behavior as LBRγ, while JI1 and JI2 require significantly less computation time compared to the generic neighborhood structures.

Job Insertion with Flexibility (JIF) extends the JIx neighborhood by one aspect regarding the fixed partial sequences of the unaffected jobs. On each involved machine, a corridor of three positions prior and posterior to the extraction position of a job is defined. Therein, the operations of the remaining schedule are allowed to be moved to smoothly merge the preserved parts and improve the objective function value of the neighboring schedule $\mathbf{x}^N$. The JIFx-neighborhood is equivalently implemented following a best fit strategy.

Unsurprisingly, the required computational effort of the construction of a neighboring solution $\mathbf{x}^N$ increases with the admission of flexibility. Thus, JIFx is tested on all benchmark instances with $x = 1$ and $x = 2$. While JIF1 returns a locally optimal schedule for all settings in less than ten minutes, JIF2 is terminated due to the time limit for some of the large Lawrence instances. The procedures provide optimal and near-optimal solutions to several problems, where it can be observed that JIF2 succeeds on instances of small and medium size and JIF1 is favorable for large instances. Furthermore, a beneficial effect of the flexibility is not obviously indicated by the results, since JIFx is outperformed by one of the JIx neighborhoods for almost half of the benchmark instances. Especially when operating the iterative improvement scheme with an initial permu-

tation schedule on large Lawrence instances, JI1 and JIF1 obtain exactly the same average results.

The *Local Position Changes (LPC)* neighborhood applies the idea of a bounded Hamming distance of two neighboring schedules given by LBRγ in a scheduling-tailored and operation-based way. Starting from a feasible incumbent solution $\bar{\mathbf{x}}$, the best neighboring solution $\mathbf{x}^N$ is determined among all schedules, for which the difference in the order-position r of every operation $O_{i,j} \in \mathcal{O}$ on its required machine $M_k \in \mathcal{M}$ is not greater than the distance parameter $\gamma \in \mathbb{Z}_{>0}$. The maximum Hamming distance per operation is implemented by adding constraints to the mixed-integer program of the BJSPT, which is solved by an MIP technique in every iteration.

The iterative improvement scheme operates LPC with the distance parameters $\gamma = 3$ and $\gamma = 5$, referred to as LPC3 and LPC5, respectively. Small instances are solved to near-optimal solutions in less than one minute with both transition schemes and initial schedules of different quality. Comparing LPC3 and LPC5 for medium and large problems, it can be observed that the computation time increases with a higher value of the distance parameter γ, while both procedures are terminated by the time limit in many cases. Similar to the behavior of previously described structures, the expected advantage of a higher number of feasible solutions in the neighborhood with $\gamma = 5$ is reportable for small and medium size instances but goes into reverse for large problems. While LPC3 and LPC5 both outperform the generic neighborhoods especially when starting with permutation schedules, they cannot improve the results obtained by insertion-based neighborhoods with regard to solution quality. As an exception, LPC5 generates the best schedules for the group of Lawrence instances of size $(30, 10)$ among all neighborhoods when a permutation schedule is initially given.

Machine Fixing (MF) constitutes the neighborhood equivalent of the FMS construction scheme. Regarding an incumbent feasible schedule $\bar{\mathbf{x}}$, the operation sequence of an arbitrarily chosen machine is taken as given and the remaining schedule is reoptimized. Accordingly, the variables related to the preserved operation sequence are fixed, while the other precedence variables are set to be unknown and the reduced mixed-integer formula-

tion of the BJSPT is solved. By fixing the operation sequences on all machines $M_k \in \mathcal{M}$ individually and determining the corresponding neighboring schedules $\mathbf{x}^N$, the entire MF-neighborhood of the incumbent solution $\bar{\mathbf{x}}$ is set up and evaluated in every iteration. Following a best fit strategy, the neighbor featuring the minimal total tardiness replaces the incumbent feasible schedule, if an improvement in the objective function value is achieved.

MF shows considerably good results for small instances, since optimal schedules are constructed in less than 30 seconds. This refers to the smallest amount of computation time among the procedures obtaining a similar solution quality. With regard to instances of medium and large size, the computation time required by MF increases significantly. The neighborhood structure is competitive to other scheduling-tailored schemes for the train scheduling-inspired instances, but does not generate schedules of satisfactory quality for randomized instances of increasing size.

Overall, the generic neighborhood structures LBRγ and RINS are clearly outperformed by the scheduling-tailored schemes in computation time and solution quality for both classes of initial schedules. Comparing the job insertion-based neighborhoods JIx and JIFx, the integration of flexibility does not obviously constitute an advantage in the generation of good neighboring schedules. The required computation time increases with higher numbers of x and additional flexibility, so that JI1 and JI2 seem to be most favorable especially for large size instances. The LPCγ-neighborhoods show convincing results of satisfactory quality with short runtimes for problems of small and medium size. For large instances, the quality of the obtained schedules is still competitive to other scheduling-tailored schemes but the computation time increases significantly. A similar behavior can be observed for the MF transition scheme. Especially for problems featuring internal structures, the iterative improvement procedure generates optimal and near-optimal schedules in considerably short computation times by applying this neighborhood, while the results become unsatisfactory for large randomized instances.

5.3 MIP-Based Variable Neighborhood Search

In the following, explanations on the applied matheuristic approach, which combines advantages of MIP solution techniques and heuristic methods, to tackle the BJSPT are given. Section 5.3.1 describes the choice of a construction scheme and different neighborhoods together with their embedding in the VNS framework. Subsequently, computational results are reported and discussed in Section 5.3.2.

5.3.1 Components and Setting

VNS constitutes a generic search method, which offers a variety of possibilities to form hybrid techniques of heuristic and non-heuristic components. Hansen, Mladenovic and Perez [67] provide detailed explanations on well-known VNS structures and a comprehensive overview of existing hybrid approaches and applications. In this thesis, a reduced variant of VNS is proposed for the BJSPT, where the number of considered neighbors and the runtime are restricted for each transition scheme. A technical description of the implemented procedure is given in Algorithm 6.

Based on the preliminary experiments reported in Section 5.2.1, the Sequential Job Insertion (SJI) is chosen as the construction scheme. Since a promising combination of short computation time and satisfactory solution quality can be observed, it is integrated in the subroutine in line 2 of Algorithm 6. With regard to the neighborhood structures, beneficial trade-offs between runtime and potential improvement are shown by the Job Insertion-based neighborhoods JIx and JIFx as well as the Local Position Changes LPCγ, see Section 5.2.2. Consequently, there are only scheduling-tailored schemes involved in the VNS, since they exceed the general-purpose methods with a reliably good performance over all instances. The number of neighborhoods $\varphi^{max} = 5$ is defined and the following sequence of neighborhoods is implemented in the VNS.

$$\mathcal{N}_1(\bar{\mathbf{x}}) = \text{JI1}, \quad \mathcal{N}_2(\bar{\mathbf{x}}) = \text{JI2}, \quad \mathcal{N}_3(\bar{\mathbf{x}}) = \text{JIF2},$$

$$\mathcal{N}_4(\bar{\mathbf{x}}) = \text{LPC3}, \quad \mathcal{N}_5(\bar{\mathbf{x}}) = \text{LPC5}$$

Algorithm 6 Reduced Variable Neighborhood Search

Input: instance input data, number of neighborhoods φ^{max}
Output: best found solution $\bar{\mathbf{x}}$

```
1:  x̄ ← ∅
2:  x_init ← solution generated by ConstructionScheme
3:  x̄ ← x_init
4:  while overall terminal condition(s) is (are) not fulfilled do
5:      φ ← 1
6:      while φ ≤ φ^max do
7:          while terminal condition(s) for N_φ(x̄) is (are) not fulfilled do
8:              generate and evaluate x^N ∈ N_φ(x̄) according to T(x^N)
9:              if T(x^N) < T(x̄) then
10:                 x̄ ← x^N
11:                 restart line 4
12:             end if
13:         end while
14:         φ ← φ + 1
15:     end while
16:     adapt neighborhood terminal condition(s)
17: end while
18: return x̄
```

The selected schemes JI1, JI2 and LPC3 realize improvements especially for medium and large benchmark instances, while requiring only a limited amount of computation time. Contrarily, moves in JIF2 and LPC5 yield a remarkable solution quality for problems of small size. The sequentially operated neighborhoods are treated as one structure, which is examined in a first improvement scheme. This means that, if the objective function value of the incumbent solution is improved by a neighbor $\mathbf{x}^N$ (line 9) and $\bar{\mathbf{x}}$ is updated in line 10, the search procedure is restarted beginning with neighborhood JI1 (see line 11).

To assure a balanced search process, there are one or more terminal conditions defined for every transition scheme $\mathcal{N}_\varphi(\bar{\mathbf{x}})$. In the neighborhoods JI1, JI2 and JIF2, neighboring solutions can explicitly be constructed and a maximum number of evaluated neighbors per iteration is defined. Additionally, a time limit is set for the examination of every single neighbor-

hood, namely the execution of the **while**-loop in lines 7 to 13 of Algorithm 6. This is of special importance for the moves in LPC3 and LPC5, where the generation and evaluation of a neighboring solution in line 8 corresponds to the determination of the best neighbor by solving a reduced mixed-integer program. Thus, these neighborhoods are not investigated by several discrete calculations but by one single computational process. Since the terminal conditions checked in line 7 restrict the method to a partial observation of each scheme $\mathcal{N}_{\varphi}(\bar{\mathbf{x}})$, there may exist improving neighbors of an incumbent solution $\bar{\mathbf{x}}$, which are not evaluated in the current iteration. Therefore, the VNS does not terminate after an iteration without improvement in the sequence of all five neighborhoods. Contrarily, the terminal conditions are adapted (line 16), so that the search steps are intensified, and the procedure continues with examining neighbors in JI1 again until the overall time limit is reached.

Figure 5.1 graphically represents the execution of the reduced VNS and depicts the specification of the different parameters and terminal conditions. After an initial solution is set as the best solution of three runs of SJI, the incumbent feasible schedule $\bar{\mathbf{x}}$ is updated. Subsequently, the first neighborhood structure JI1 is operated until a neighbor $\mathbf{x}^N$ with $T(\mathbf{x}^N) < T(\bar{\mathbf{x}})$ is found or one of the neighborhood-specific terminal conditions is reached. To avoid an overproportional use of a specific transition scheme, the number of considered neighbors is bounded by ten and an initial time limit of 10 seconds is set for the examination of every neighborhood. If no improvement is reached, the VNS continues by investigating the next neighborhood structure, here JI2. This mechanism is equivalently executed for all neighborhoods in the sequence. In case that an improving neighbor is found, the incumbent solution $\bar{\mathbf{x}}$ is updated and the VNS continues with examining JI1. If there is no solution featuring $T(\mathbf{x}^N) < T(\bar{\mathbf{x}})$ obtained in all five transition schemes, the neighborhood time limit is doubled to intensify the search process. There is an upper bound of 300 s defined, which is automatically taken as the time limit, in case the new value exceeds this measure. Furthermore, the neighborhood time limit is replaced by the remaining overall time, if this period is shorter than the current runtime bound of the transition scheme. The VNS operates iteratively and returns the current best solution $\bar{\mathbf{x}}$ when the overall time limit is reached.

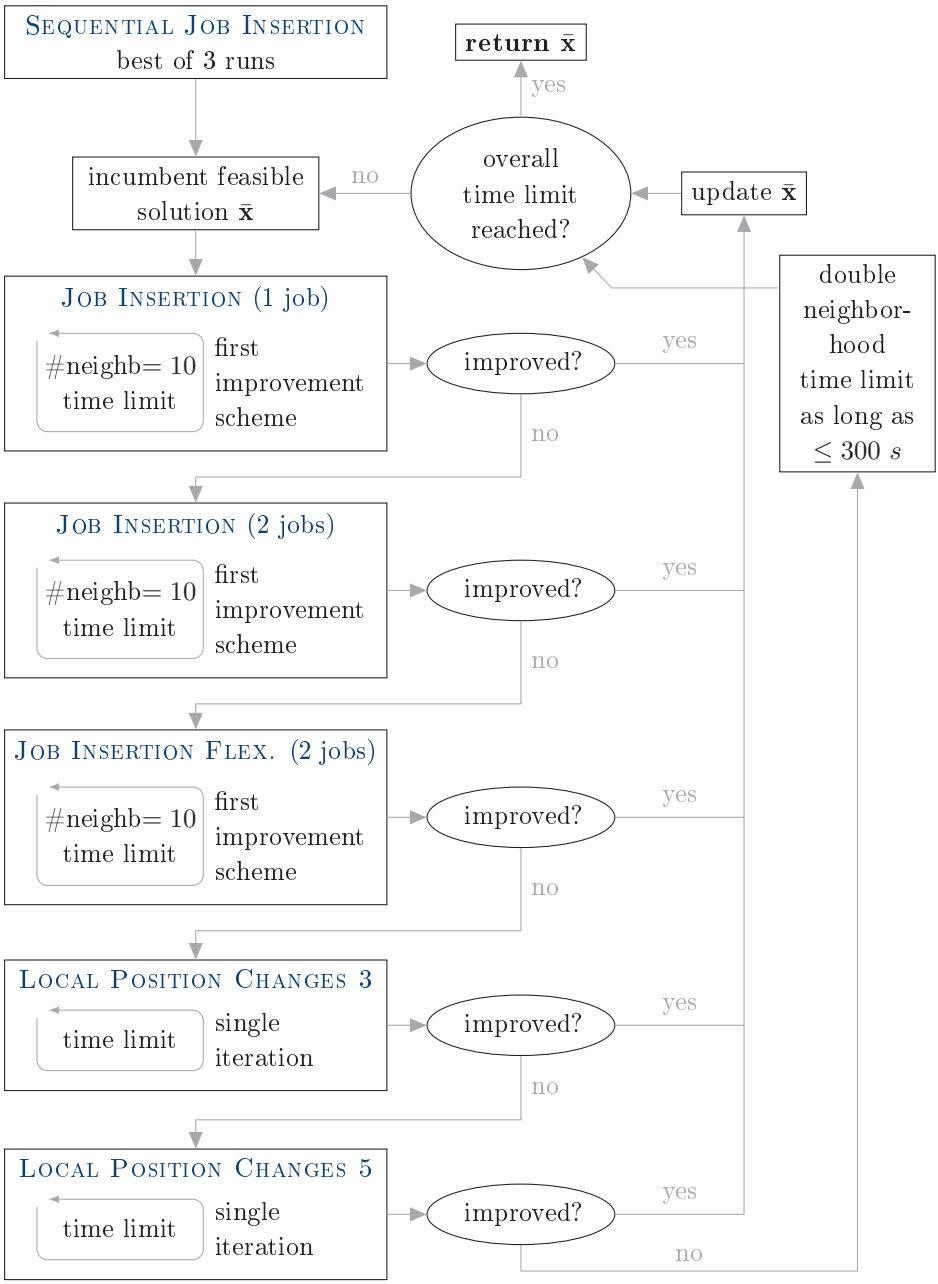


Figure 5.1: Illustration of the Reduced Variable Neighborhood Search applied to the BJSPT

5.3.2 Computational Experiments

Equivalent to the preliminary tests, the computational experiments are conducted with the VNS given in Algorithm 6 implemented in Java and applying Gurobi 7.5.2. Since the initial findings indicate that an overall time limit of 30 minutes is sufficient to obtain schedules of good quality and short runtimes are a desirable feature of every solution approach, the procedure is restricted to this time period as the overall terminal condition. The results for the benchmark instances are obtained in five independent runs for each problem using a notebook operating an Intel Dual Core i5 processor (2.20 GHz) with 8 GB RAM.

Tables 5.1 and 5.2 present an overview of the numerical experiments performed on the train scheduling-inspired instances and the Lawrence instances, respectively. The first two columns contain the instance index and the objective function value of the best schedule found by the MIP solver, see Section 3.5. As before, solutions with proven optimality are marked by an asterisk. The next two columns show the mean total tardiness $\overline{\sum T_i}$ and the minimum total tardiness $\min(\sum T_i)$ obtained by five independent runs of SA in Setting 2 applying the API-neighborhood to facilitate a direct comparison of the exact, the heuristic and the matheuristic solution method. Finally, the last three columns of the tables equivalently depict the mean total tardiness and minimum total tardiness reached by the VNS together with the mean time required to obtain the best schedule in seconds.

Considering the objective function values of both classes of benchmark instances, it can be stated that the VNS successfully combines the advantages of the exact and the heuristic solution method. For all instances, for which an optimal solution is found by the MIP solver, the best possible schedule is equivalently determined by the matheuristic. Note that the hybrid approach additionally benefits from its MIP components in reliably generating these optimal solutions in all five runs. Furthermore, the procedure features a considerable enhancement in computation time by only involving the exact solution of reduced optimization programs. Positive effects of the heuristic search framework can be observed especially for instances of large size. Minimal total tardiness values, which are exclusively

Table 5.1: Computational results of the VNS applied to the train scheduling-inspired instances of the BJSPT

Inst.	MIP	SA-API		VNS		
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
ts01	138*	140.2	138*	138.0*	138*	8.0
ts02	90*	94.6	91	90.0*	90*	4.2
ts03	72*	74.2	72*	72.0*	72*	1.6
ts04	41*	41.8	41*	41.0*	41*	3.8
ts05	71*	71.4	71*	71.0*	71*	2.6
ts06	88*	121.6	107	92.8	88*	52.2
ts07	172*	195.4	189	182.2	172*	130.4
ts08	163*	184.2	179	165.0	163*	365.4
ts09	153	178.8	168	158.8	156	457.6
ts10	97*	114.8	97*	97.0*	97*	37.6
ts11	366	406.4	390	368.6	**359**	674.8
ts12	419	428.2	412	416.0	**398**	556.8
ts13	452	462.6	448	450.8	**416**	611.8
ts14	459	462.8	418	460.4	447	781.0
ts15	418	419.4	401	377.0	**355**	495.2

obtained by the matheuristic, are highlighted by boldface printing in Tables 5.1 and 5.2. Due to the heuristic guidance of the VNS in allocating restricted search capacities and the involvement of neighborhoods of different structure, the overall best schedule is found for four of the largest train scheduling-inspired instances and 19 Lawrence instances of sizes $(20, 5)$, $(15, 10)$, $(20, 10)$, $(30, 10)$ and $(15, 15)$. Thus, a hybridization of exact and heuristic solution methods is particularly favorable against a separate application with regard to the size problems for which near-optimal solutions can be determined. This observation gives a strong emphasis on matheuristic solution techniques as a promising research direction to solve practically relevant job shop scheduling problems.

Table 5.2: Computational results of the VNS applied to the Lawrence instances of the BJSPT

Inst.	MIP	SA-API		VNS		
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la01	762*	787.4	773	762.0*	762*	12.0
la02	266*	283.4	266*	266.0*	266*	0.2
la03	357*	357.0*	357*	357.0*	357*	0.8
la04	1165*	1217.2	1165*	1165.0*	1165*	54.8
la05	557*	557.0*	557*	557.0*	557*	1.4
la06	2516	2790.0	2616	2548.0	2516	445.6
la07	1677*	1942.2	1869	1769.6	1677*	423.6
la08	1829*	2335.0	1905	1829.0*	1829*	71.6
la09	2851	3275.2	3161	2864.6	2854	286.6
la10	1841*	2178.2	2069	1841.0*	1841*	112.2
la11	6534	6186.2	5704	5436.0	**4973**	1288.8
la12	5286	5070.0	4859	4746.2	**4389**	718.4
la13	7737	7850.6	7614	6760.2	**6542**	1454.4
la14	6038	6616.8	5714	5812.2	**5558**	849.6
la15	7082	7088.6	5626	6352.6	5923	1046.6
la16	330*	395.8	335	330.0*	330*	9.6
la17	118*	144.2	120	118.0*	118*	3.6
la18	159*	229.4	159*	159.0*	159*	6.8
la19	243*	306.6	243*	243.0*	243*	4.2
la20	42*	55.6	42*	42.0*	42*	1.8
la21	1956	2847.2	2101	2045.4	**1834**	935.8
la22	1455	2052.8	1773	1413.4	1308	808.2
la23	3436	3692.6	3506	3408.4	**3013**	636.0
la24	560*	966.8	761	560.0*	560*	28.6
la25	1002	1557.4	1289	1021.4	**990**	218.4

continued on the next page

Inst.	MIP	SA-API		VNS		
		$\overline{\sum T_i}$	$\min(\sum T_i)$	$\overline{\sum T_i}$	$\min(\sum T_i)$	Time
la26	7961	9275.8	8475	6646.0	**6284**	787.4
la27	8915	7588.0	6596	6186.8	**5779**	743.8
la28	2226	3430.8	2876	2408.4	**2189**	984.4
la29	2018	2948.0	2432	1926.6	**1707**	1124.0
la30	6655	7621.6	6775	5551.8	**4860**	1092.2
la31	20957	18921.8	17984	16374.4	**15665**	1154.6
la32	23150	21991.4	20401	17500.2	**16924**	1199.2
la33	none	22494.2	19750	17732.6	**15610**	1249.4
la34	none	20282.8	18633	17535.0	**16256**	1323.4
la35	none	21895.0	18778	19152.8	**17932**	1420.6
la36	675	1856.0	1711	981.0	678	809.6
la37	1070	1774.2	1621	952.8	**861**	478.8
la38	489*	760.4	645	489.0*	489*	313.6
la39	754	1573.0	1391	903.4	**751**	792.4
la40	407*	1008.6	613	407.0*	407*	240.6

6 Concluding Remarks

In this thesis, different research questions concerning the job shop scheduling problem with characteristics, constraints and an optimization criterion of practical relevance are addressed. Since the literature shows a lack of comprehensive empirical and theoretical results, with which statements on the exact and heuristic resolvability of complex job shop problems can be reasoned, solution techniques for the blocking job shop problem with total tardiness minimization are presented, tested and discussed.

With the intention to investigate the performance of general-purpose MIP solvers on the problem under study, two well-known mathematical formulations applying ordering variables are compared. In line with the observation of a significantly lower number of variables and constraints in the model, the computational experiments, representatively performed by

IBM ILOG CPLEX 12.8.0, show that the implementation of the BJSPT by precedence variables, which indicate the pairwise orderings of operations requiring the same machine, is most advantageous with respect to solution quality and computation time. Among the diverse benchmark instances involving strong routing patterns as well as purely randomized structures, the state-of-the-art MIP software obtains an optimal solution of small instances of sizes like $(10, 5)$ and $(10, 11)$ within seconds, while it is not able to return satisfactory feasible schedule for problems including 10 to 15 machines and 15 to 30 jobs after two hours of runtime. Thus, the boundaries of exact resolvability of complex job shop scheduling problems by general MIP techniques do not meet the appearing practical necessities in instance size and computation time. Here, enhancements in the relaxation-based lower bounds of the mathematical formulations especially for indicator constraints and tardiness-based objectives can be named as an important research issue.

Considering the relation of particular instance properties and the expected computational effort to obtain an optimal schedule, the experiments indicate that complex characteristic measures are required to precisely cover all involved effects. Nonetheless, the proposed instance key figures, namely the mean machine utilization rate $\bar{u}$ and the mean machine slack $\bar{l}$, seem to constitute good references in comparing problems of the same size (n, m). They represent the internal amount of competition of jobs for machine working time as a highly influential aspect. In line with this, further research is required to design a reliable combined measure, which includes simple instance properties like the number of jobs and the number of machines as well as indicators of the amounts of involved structure and randomness and of the individual workload of the machines.

Three proposed scheduling-tailored neighborhood structures are operated by an iterative improvement scheme and an SA metaheuristic on the benchmark instances. The computational results clearly show that a heuristic method is able to construct high quality schedules for the BJSPT. Especially when allowing several independent runs of the procedure to balance the impact of randomized components, the permutation-based SA involving the interchange-focused transition scheme produces optimal and near-optimal schedules for instances of small and medium size, easily. For large

problems, the method does still outperform the general MIP technique in constructing feasible solutions, but the growing quantity of possible interchanges and the fast increasing amount of computation time needed to actually obtain a feasible neighbor with a given interchange cut down beneficial effects. This issue can be observed in various applications of heuristic methods to the blocking job shop problem, so that the creation of a simple feasibility checking procedure is highlighted as a key issue of future research.

The utilization of single and nested permutations as representations of a schedule features the advantages of simplicity and applicability of basic transition operators but involves redundancy and requires a list scheduling algorithm to detect the potential infeasibility of the solution. Incorporating blocking constraints and recirculation causes the enormous indispensable complexness of the algorithmic repair procedure to generate neighboring solutions based on an incumbent schedule and a given adjacent pairwise interchange. Permutation-based representation techniques are adaptable and applicable in solving complex job shop scheduling problems, but there is further research needed to state whether or not they are favorable against graph-based methods.

The advantageousness of matheuristic methods to solve complex job shop scheduling problems is investigated by applying a VNS scheme, which consists of MIP-based components. Preliminary experiments show that scheduling-tailored mechanisms outperform general-purpose techniques in finding and improving feasible solutions for the BJSPT. The results obtained by the combination of five selected promising methods featuring different computational structures clearly highlight the remarkable capability of hybrid solution approaches in job shop scheduling. The advantage of the exact MIP technique, namely returning optimal solutions for small optimization programs in extremely short runtime, is used and enhanced by the guidance of the heuristic search procedure. Thus, the proposed VNS matheuristic constitutes a successful first step in a new research direction.

Altogether, comprehensive theoretical advances and computational studies shed light onto the complicatedness and the exact and heuristic solvability of job shop scheduling problems with practical relevance. The analysis of the behavior of permutation-based schedule encodings creates an insight

into structural effects of real-world constraints, and two new techniques to heuristically solve the problem under study is presented. Overall, this thesis expands the understanding and the resolvability of complex job shop scheduling problems and gives rise to complementary further research issues.

Bibliography

[1] Aarts, E. H. L., Van Laarhoven, P. J. M., Lenstra, J. K., and Ulder, N. L. J. A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing 6*, 2 (1994), 118–125.

[2] Adams, J., Balas, E., and Zawack, D. The shifting bottleneck procedure for job shop scheduling. *Management Science 34*, 3 (1988), 391–401.

[3] Adams, S. Un-freakonomics. *Forbes (online, latest access: 2018, dec 10th)* (2010).

[4] AitZai, A., and Boudhar, M. Parallel branch-and-bound and parallel PSO algorithms for job shop scheduling problem with blocking. *International Journal of Operational Research 16*, 1 (2013), 14–37.

[5] Anderson, E. J., Glass, C. A., and Potts, C. N. *Machine Scheduling*. Wiley Chichester, UK, 1997, ch. 11, pp. 361–414.

[6] Anderson, E. J., and Nyirenda, J. C. Two new rules to minimize tardiness in a job shop. *International Journal of Production Research 28*, 12 (1990), 2277–2292.

[7] Applegate, D., and Cook, W. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing 3*, 2 (1991), 149–156.

[8] Baker, K. R. Sequencing rules and due-date assignments in a job shop. *Management Science 30*, 9 (1984), 1093–1104.

[9] Baker, K. R., and Kanet, J. J. Job shop scheduling with modified due dates. *Journal of Operations Management 4*, 1 (1983), 11–22.

[10] Balas, E. Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. *Operations Research 17*, 6 (1969), 941–957.

[11] Balas, E. On the facial structure of scheduling polyhedra. In *Mathematical Programming Essays in Honor of George B. Dantzig Part I*. Springer Berlin Heidelberg, 1985, pp. 179–218.

[12] Balas, E., Lancia, G., Serafini, P., and Vazacopoulos, A. Job shop scheduling with deadlines. *Journal of Combinatorial Optimization 1*, 4 (1998), 329–353.

[13] Balas, E., and Vazacopoulos, A. Guided local search with shifting bottleneck for job shop scheduling. *Management Science 44*, 2 (1998), 262–275.

[14] Beasley, J. E. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society 41*, 11 (1990), 1069–1072.

[15] Behnamian, J. Matheuristic for the decentralized factories scheduling problem. *Applied Mathematical Modelling 47* (2017), 668–684.

[16] Belotti, P., Bonami, P., Fischetti, M., Lodi, A., Monaci, M., Nogales-Gómez, A., and Salvagnin, D. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications 65*, 3 (2016), 545–566.

[17] Bertsimas, D., and Tsitsiklis, J. Simulated annealing. *Statistical Science 8*, 1 (1993), 10–15.

[18] Bierwirth, C. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum 17*, 2-3 (1995), 87–92.

[19] Bierwirth, C., Mattfeld, D. C., and Kopfer, H. On permutation representations for scheduling problems. In *Parallel Problem Solving from Nature IV*. Springer Berlin Heidelberg, 1996, pp. 310–318.

[20] Bierwirth, C., Mattfeld, D. C., and Watson, J.-P. Landscape regularity and random walks for the job-shop scheduling problem. In *European Conference on Evolutionary Computation in Combina-*

torial Optimization (2004), Springer, pp. 21–30.

[21] Bixby, E. R., Fenelon, M., Gu, Z., Rothberg, E., and Wunderling, R. MIP: theory and practice – closing the gap. In *System Modelling and Optimization* (Boston, MA, 2000), M. J. D. Powell and S. Scholtes, Eds., Springer US, pp. 19–49.

[22] Błażewicz, J., Domschke, W., and Pesch, E. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research 93*, 1 (1996), 1–33.

[23] Błażewicz, J., Dror, M., and Weglarz, J. Mathematical programming formulations for machine scheduling: A survey. *European Journal of Operational Research 51*, 3 (1991), 283–300.

[24] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Weglarz, J. *Handbook on Scheduling: From Theory to Applications.* International Handbook on Information Systems. Springer Berlin Heidelberg, 2007.

[25] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., and Weglarz, J. *Scheduling Computer and Manufacturing Processes.* Springer, 2013.

[26] Błażewicz, J., Pesch, E., and Sterna, M. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research 127*, 2 (2000), 317–331.

[27] Bonami, P., Lodi, A., Tramontani, A., and Wiese, S. On mathematical programming with indicator constraints. *Mathematical Programming 151*, 1 (2015), 191–223.

[28] Bowman, E. H. The schedule-sequencing problem. *Operations Research 7*, 5 (1959), 621–624.

[29] Brizuela, C. A., Zhao, Y., and Sannomiya, N. No-wait and blocking job-shops: Challenging problems for GA's. In *International Conference on Systems, Man and Cybernetics, 2001* (2001), vol. 4, IEEE, pp. 2349–2354.

[30] Brucker, P. *Scheduling Algorithms.* Springer Berlin Heidelberg, 2004.

[31] Brucker, P., and Jurisch, B. A new lower bound for the job-shop

scheduling problem. *European Journal of Operational Research 64*, 2 (1993), 156–167.

[32] Brucker, P., and Knust, S. *Complex Scheduling*. Springer, 2011.

[33] Burdett, R. L., and Kozan, E. A disjunctive graph model and framework for constructing new train schedules. *European Journal of Operational Research 200*, 1 (2010), 85–98.

[34] Bürgy, R. A neighborhood for complex job shop scheduling problems with regular objectives. *Journal of Scheduling 20*, 4 (2017), 391 – 422.

[35] Bürgy, R., and Gröflin, H. Optimal job insertion in the no-wait job shop. *Journal of Combinatorial Optimization 26*, 2 (2012), 345–371.

[36] Bürgy, R., and Gröflin, H. The blocking job shop with rail-bound transportation. *Journal of Combinatorial Optimization 31*, 1 (2016), 152–181.

[37] Carballo, L., Lazarev, A. A., Vakhania, N., and Werner, F. Search on the enumeration tree in the multiprocessor job-shop problem. *IFAC Proceedings Volumes 45*, 6 (2012), 81–86.

[38] Carlier, J., and Pinson, E. A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research 26* (1990).

[39] Caserta, M., and Voss, S. Metaheuristics: Intelligent problem solving. In *Matheuristics*. Springer US, 2009, pp. 1–38.

[40] Cheng, R., Gen, M., and Tsujimura, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms - I. representation. *Computers & Industrial Engineering 30*, 4 (1996), 983–997.

[41] Cheng, R., Gen, M., and Tsujimura, Y. A tutorial survey of job-shop scheduling problems using genetic algorithm - II: hybrid genetic search strategies. *Computers & Industrial Engineering 36*, 2 (apr 1999), 343–364.

[42] Danna, E., Rothberg, E., and Le Pape, C. Integrating mixed integer programming and local search: A case study on job-shop scheduling problems. In *Fifth International Workshop on Integration*

of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems 2003 (2003), pp. 65–79.

[43] Danna, E., Rothberg, E., and Le Pape, C. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming 102*, 1 (2005), 71–90.

[44] Dantzig, G. B. A machine-job scheduling model. *Management Science 6*, 2 (1960), 191–196.

[45] D'Ariano, A., D'Urgolo, P., Pacciarelli, D., and Pranzo, M. Optimal sequencing of aircrafts take-off and landing at a busy airport. In *Proceedings of the 13th International IEEE Annual Conference on Intelligent Transportation Systems* (2010), pp. 1569–1574.

[46] D'Ariano, A., Pacciarelli, D., and Pranzo, M. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research 183*, 2 (2007), 643–657.

[47] Della Croce, F., Grosso, A., and Salassa, F. A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research 213*, 1 (2011), 67–78.

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

[49] Diestel, R. *Graph theory (Graduate texts in mathematics)*, vol. 173. Springer, 2016.

[50] Dorndorf, U., and Pesch, E. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research 22*, 1 (1995), 25–40.

[51] Fanjul-Peyro, L., Perea, F., and Ruiz, R. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research 260*, 2 (2017), 482–493.

[52] Fischetti, M., Glover, F., and Lodi, A. The feasibility pump. *Mathematical Programming 104*, 1 (2005), 91–104.

[53] Fischetti, M., and Lodi, A. Local branching. *Mathematical Programming 98*, 1-3 (2003), 23–47.

[54] Fischetti, M., and Monaci, M. Using a general-purpose mixed-integer linear programming solver for the practical solution of real-time train rescheduling. *European Journal of Operational Research 263*, 1 (2017), 258–264.

[55] Fisher, H., and Thompson, G. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling*, J. Muth and G. Thompson, Eds. Prentice Hall, 1963, pp. 225–251.

[56] Framinan, J. M., and Perez-Gonzalez, P. Order scheduling with tardiness objective: Improved approximate solutions. *European Journal of Operational Research 266*, 3 (2018), 840–850.

[57] Garey, M. R., and Johnson, D. S. *Computers and Intractability*. W.H. Freeman and Company, New York, 1997.

[58] Garey, M. R., Johnson, D. S., and Sethi, R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*, 2 (1976), 117–129.

[59] Gere, W. S. J. Heuristics in job shop scheduling. *Management Science 13*, 3 (1966), 167–190.

[60] Giffler, B., and Thompson, G. L. Algorithms for solving production-scheduling problems. *Operations Research 8*, 4 (1960), 487–503.

[61] Gonzalez, T., and Sahni, S. Flowshop and jobshop schedules: Complexity and approximation. *Operations Research 26*, 1 (1978), 36–52.

[62] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, vol. 5. Elsevier, 1979, pp. 287–326.

[63] Gröflin, H., and Klinkert, A. Feasible insertions in job shop scheduling, short cycles and stable sets. *European Journal of Operational Research 177*, 2 (2007), 763–785.

[64] Gröflin, H., and Klinkert, A. A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics 157*, 17 (2009), 3643–3655.

[65] Hall, N. G., and Sriskandarajah, C. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research 44*, 3 (1996), 510–525.

[66] Hamming, R. W. Error detecting and error correcting codes. *Bell Labs Technical Journal 29*, 2 (1950), 147–160.

[67] Hansen, P., Mladenović, N., and Moreno Pérez, J. A. Variable neighbourhood search: methods and applications. *Annals of Operations Research 175*, 1 (2008), 367–407.

[68] Harrod, S. Modeling network transition constraints with hypergraphs. *Transportation Science 45*, 1 (2011), 81–97.

[69] Haupt, R. A survey of priority rule-based scheduling. *OR Spektrum 11*, 1 (1989), 3–16.

[70] Holthaus, O., and Rajendran, C. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics 48*, 1 (1997), 87–105.

[71] Holthaus, O., and Rajendran, C. Efficient jobshop dispatching rules: further developments. *Production Planning & Control 11*, 2 (2000), 171–178.

[72] Jaehn, F., and Pesch, E. *Ablaufplanung: Einführung in Scheduling.* Springer, 2014.

[73] Koch, T. *Rapid Mathematical Programming.* PhD thesis, Technische Universität Berlin, 2004.

[74] Kolonko, M. Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research 113*, 1 (1999), 123–136.

[75] Kononova, P. A., and Kochetov, Y. A. The variable neighborhood search for the two machine flow shop problem with a passive prefetch. *Journal of Applied and Industrial Mathematics 7*, 1 (2013), 54–67.

[76] Kuhpfahl, J., and Bierwirth, C. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research 66* (2016), 44–57.

[77] Lange, J. Approaches to modeling job-shop problems with blocking constraints. In *Proceedings of the 7th Multidisciplinary International Conference on Scheduling: Theory and Applications* (2015), pp. 645–648.

[78] Lange, J., and Werner, F. Approaches to modeling train scheduling problems as job-shop problems with blocking constraints. *Journal of Scheduling 21*, 2 (2018), 191–207.

[79] Lange, J., and Werner, F. A permutation-based neighborhood for the blocking job-shop problem with total tardiness minimization. In *Operations Research Proceedings*. Springer International Publishing, 2018, pp. 581–586.

[80] Lawrence, S. Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University, Pittsburgh, PA* (1984).

[81] Lenstra, J., and Rinnooy Kan, A. Computational complexity of discrete optimization problems. In *Discrete Optimization I, Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium*. Elsevier, 1979, pp. 121–140.

[82] Lenstra, J., Rinnooy Kan, A., and Brucker, P. Complexity of machine scheduling problems. In *Studies in Integer Programming*. Elsevier, 1977, pp. 343–362.

[83] Liu, S. Q., and Kozan, E. Scheduling trains as a blocking parallel-machine job shop scheduling problem. *Computers & Operations Research 36*, 10 (2009), 2840–2852.

[84] Liu, S. Q., and Kozan, E. Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model. *Transportation Science 45*, 2 (2011), 175–198.

[85] Lodi, A., and Tramontani, A. Performance variability in mixed-integer programming. In *Theory Driven by Influential Applications*. INFORMS, 2013, pp. 1–12.

[86] Manne, A. S. On the job-shop scheduling problem. *Operations Research 8*, 2 (1960), 219–223.

[87] Mascis, A., and Pacciarelli, D. Machine scheduling via alternative graphs. Tech. rep., Universita degli Studi Roma Tre, DIA, 2000.

[88] Mascis, A., and Pacciarelli, D. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research 143*, 3 (2002), 498–517.

[89] Mati, Y., Rezg, N., and Xie, X. A taboo search approach for deadlock-free scheduling of automated manufacturing systems. *Journal of Intelligent Manufacturing 12*, 5-6 (2001), 535–552.

[90] Matsuo, H., Suh, C., and Sullivan, R. A controlled search simulated annealing method for the general jobshop scheduling problem. Tech. rep., Working Paper 3-4-88, Department of Management, Graduate School of Business, University of Texas, Austin, 1988.

[91] Mattfeld, D., Bierwirth, C., and Kopfer, H. A search space analysis of the job shop scheduling problem. *Annals of Operations Research 86* (1999), 441–453.

[92] Mattfeld, D. C., and Bierwirth, C. An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research 155*, 3 (2004), 616–630.

[93] Meng, L., and Zhou, X. Simultaneous train rerouting and rescheduling on an n-track network: A model reformulation with network-based cumulative flow variables. *Transportation Research Part B: Methodological 67* (2014), 208–234.

[94] Mönch, L., and Roob, S. A matheuristic framework for batch machine scheduling problems with incompatible job families and regular sum objective. *Applied Soft Computing 68* (2018), 835–846.

[95] Nawaz, M., Enscore, E. E., and Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega 11*, 1 (1983), 91–95.

[96] Nowicki, E., and Smutnicki, C. A fast taboo search algorithm for the job shop problem. *Management Science 42*, 6 (1996), 797–813.

[97] Nowicki, E., and Smutnicki, C. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling 8*, 2 (2005), 145–159.

[98] Oddi, A., Rasconi, R., Cesta, A., and Smith, S. F. Iterative

improvement algorithms for the blocking job shop. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling* (2012).

[99] Oliveira, E., and Smith, B. M. A job-shop scheduling model for the single-track railway scheduling problem. Research Report Series 21, University of Leeds, 2000.

[100] Pacciarelli, D., and Pranzo, M. Production scheduling in a steelmaking-continuous casting plant. *Computers & Chemical Engineering 28*, 12 (2004), 2823–2835.

[101] Pan, C.-H. A study of integer programming formulations for scheduling problems. *International Journal of Systems Science 28*, 1 (1997), 33–41.

[102] Pinedo, M. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[103] Pranzo, M., and Pacciarelli, D. An iterated greedy metaheuristic for the blocking job shop scheduling problem. *Journal of Heuristics 22*, 4 (jan 2016), 587–611.

[104] Raghu, T., and Rajendran, C. An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics 32*, 3 (1993), 301–313.

[105] Rego, C., and Duarte, R. A filter-and-fan approach to the job shop scheduling problem. *European Journal of Operational Research 194*, 3 (2009), 650–662.

[106] Russell, R., Dar-El, E., and Taylor, B. A comparative analysis of the covert job sequencing rule using various shop performance measures. *International Journal of Production Research 25*, 10 (1987), 1523–1540.

[107] Schiavinotto, T., and Stützle, T. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research 34*, 10 (2007), 3143–3153.

[108] Smith-Miles, K., James, R., Giffin, J., and Tu, Y. Understanding the relationship between scheduling problem structure and heuristic performance using knowledge discovery. *Learning and Intelligent*

Optimization 3 (2009).

[109] Smith-Miles, K., and Lopes, L. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research 39*, 5 (2012), 875–889.

[110] Sotskov, Y. N., and Gholami, O. Shifting bottleneck algorithm for train scheduling in a single-track railway. In *Proceedings of the 14th IFAC Symposium INCOM'12 on Information Control Problems in Manufacturing* (2012).

[111] Sotskov, Y. N., Gholami, O., and Werner, F. Solving a job-shop scheduling problem by an adaptive algorithm based on learning. In *Manufacturing Modeling, Management and Control* (2013), pp. 1352–1357.

[112] Steinhöfel, K., Albrecht, A., and Wong, C. Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research 118*, 3 (1999), 524–548.

[113] Steinhöfel, K., Albrecht, A., and Wong, C. Fast parallel heuristics for the job shop scheduling problem. *Computers & Operations Research 29*, 2 (2002), 151–169.

[114] Storer, R. H., Wu, S. D., and Vaccari, R. New search spaces for sequencing problems with application to job shop scheduling. *Management Science 38*, 10 (1992), 1495–1509.

[115] Szpigel, B. Optimal train scheduling on a single track railway. In *Operations Research '72* (1973), M. Ross, Ed., International Federation of Operational Research Societies, North-Holland Publishing Company, pp. 343–352.

[116] Ta, Q. C., Billaut, J.-C., and Bouquard, J.-L. Matheuristic algorithms for minimizing total tardiness in the m-machine flow-shop scheduling problem. *Journal of Intelligent Manufacturing 29*, 3 (2015), 617–628.

[117] Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operations Research 64* (1993), 278 – 285.

[118] Taillard, E. D. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing 6*, 2 (1994), 108–

117.

[119] Trabelsi, W., Sauvey, C., and Sauer, N. Heuristic methods for problems with blocking constraints solving jobshop scheduling. In *Proceedings of the 8th International Conference on Modelling and Simulation* (2010).

[120] Vaessens, R. J. M., Aarts, E. H., and Lenstra, J. K. Job shop scheduling by local search. *INFORMS Journal on Computing 8*, 3 (1996), 302–317.

[121] Van Laarhoven, P. J., and Aarts, E. H. *Simulated annealing*. Kluwer Academic Publishers, 1987.

[122] Van Laarhoven, P. J., Aarts, E. H., and Lenstra, J. K. Job shop scheduling by simulated annealing. *Operations Research 40*, 1 (1992), 113–125.

[123] Vepsalainen, A. P., and Morton, T. E. Priority rules for job shops with weighted tardiness costs. *Management Science 33*, 8 (1987), 1035–1047.

[124] Wagner, H. M. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly 6*, 2 (1959), 131–140.

[125] Wang, T.-Y., and Wu, K.-B. A revised simulated annealing algorithm for obtaining the minimum total tardiness in job shop scheduling problems. *International Journal of Systems Science 31*, 4 (2000), 537–542.

[126] Watson, J.-P., Beck, J. C., Howe, A. E., and Whitley, L. D. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence 143*, 2 (2003), 189–217.

[127] Werner, F. Some relations between neighbourhood graphs for a permutation problem. *Optimization 22*, 2 (1991), 297–306.

[128] Werner, F. *A survey of genetic algorithms for shop scheduling problems*. Nova Science Publishers, 2013, ch. 8, pp. 161–222.

[129] Werner, F., and Winkler, A. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics 58*, 2 (1995), 191–211.

[130] Wilbaut, C., and Hanafi, S. New convergent heuristics for 0–1 mixed integer programming. *European Journal of Operational Research 195*, 1 (2009), 62–74.

[131] Yamada, T., and Nakano, R. A genetic algorithm applicable to large-scale job-shop problems. In *Proceedings of the Second International Conference on Parallel Problem Solving from Nature* (1992), R. Männer and B. Manderick, Eds., vol. 2, Elsevier Science Publishers, pp. 281–290.

[132] Zaefferer, M., Stork, J., and Bartz-Beielstein, T. Distance measures for permutations in combinatorial efficient global optimization. In *Parallel Problem Solving from Nature XIII*. Springer International Publishing, 2014, pp. 373–383.

[133] Zhang, R., and Wu, C. A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research 38*, 5 (2011), 854–867.

[134] Zhou, X., and Zhong, M. Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds. *Transportation Research Part B: Methodological 41*, 3 (2007), 320–341.