Produktion und Logistik

RESEARCH

Jens Kuhpfahl

Job Shop Scheduling with Consideration of Due Dates

Potentials of Local Search Based Solution Techniques

Springer Gabler

Produktion und Logistik

Diese Reihe dient der Veröff entlichung neuer Forschungsergebnisse auf den Gebieten der Produktion und Logistik. Aufgenommen werden vor allem herausragende quantitativ orientierte Dissertationen und Habilitationsschriften. Die Publikationen vermitteln innovative Beiträge zur Lösung praktischer Anwendungsprobleme der Produktion und Logistik unter Einsatz quantitativer Methoden und moderner Informationstechnologie.

Kontakt
Professor Dr. Thomas S. Spengler
Technische Universität Braunschweig
Institut für Automobilwirtschaft
und Industrielle Produktion
Mühlenpfordtstraße 23
38106 Braunschweig

Jens Kuhpfahl

Job Shop Scheduling with Consideration of Due Dates

Potentials of Local Search Based Solution Techniques

Foreword by Prof. Dr. Christian Bierwirth

Springer Gabler

Jens Kuhpfahl
Halle, Germany

Dissertation University of Halle (Saale), 2015

Produktion und Logistik
ISBN 978-3-658-10291-3 ISBN 978-3-658-10292-0 (eBook)
DOI 10.1007/978-3-658-10292-0

Library of Congress Control Number: 2015941508

Springer Gabler

Printed on acid-free paper

Springer Gabler is a brand of Springer Fachmedien Wiesbaden
Springer Fachmedien Wiesbaden is part of Springer Science+Business Media
(www.springer.com)

Foreword

A bulk of research in job shop scheduling is dealing with problems where finding a solution schedule with minimum makespan is desired. Recently, also tardiness objectives received attention because on-time order fulfillment is of increasing importance particularly in pull-oriented supply chain systems. Keeping job due dates is a prerequisite for avoiding stock outs and for serving customers within promised delivery times. In case of highly utilized machine capacities, however, not every jobs might be producible on time. In such situations the minimization of the total tardiness of the jobs turns out as an appropriate objective for production scheduling. The book of Jens Kuhpfahl investigates scheduling methods, more precisely local search based heuristics, for the job shop problem with total weighted tardiness (JSPTWT) as objective. The operation research literature treating this problem class dates back over fifteen to twenty years. In this relatively short period an area of intense research has developed which is systematically reviewed and refined by the author. Based on the well-known disjunctive graph model for the classical minimum makespan problem, he analyses existing neighborhood search operators for the JSPTWT and derives new definitions to modify solution schedules in so far unknown ways. Feasibility guarantees and connectivity properties are proved analytically for these neighborhoods and their performance is compared empirically. To reduce computation times, a fast method for assessing the schedule quality, called Head Updating, is proposed. These and further components are thoroughly integrated into the metaheuristic framework of a Greedy Randomized Adaptive Search Procedure (GRASP) delivering a new state-of-the-art method for the JSPTWT. Although research in deterministic machine scheduling is highly developed, the book succeeds in closing existing gaps and contributing new ideas to the design of scheduling heuristics. I wish it a great success.

Christian Bierwirth

Acknowlegdement

I would like to express my gratitude to the people who supported and helped me achieving this dissertation. Special thanks goes to:

- Prof. Dr. Christian Bierwirth - my PhD supervisor - for his continuous support and guidance, as well as the chance for me to discover the fascinating world of scheduling.

- Prof. Dr. Dirk C. Mattfeld for the enlightening discussions at several conferences and the permission to use the code of its genetic algorithm.

- My colleagues from the chair of Production and Logistics for their support and the pleasant working atmosphere.

- My family for their love, the constant encouragement and the financial support particularly at the beginning of my studies.

- Isabell for her love, warmth, support, motivation and vitality.

Jens Kuhpfahl

Contents

List of Figures

List of Tables

List of Notations

n	-	number of jobs
J	-	set of jobs, $J = \{1, 2, \dots n\}$
m	-	number of machines
M	-	set of machines, $M = \{1, 2, \dots m\}$
o_{ij}	-	the i-th operation of job j
μ_{ij}	-	associated machine of operation o_{ij}
p_{ij}	-	processing time of operation o_{ij}
r_j	-	release time of job j
w_j	-	weight of job j
d_j	-	due date of job j
c_j	-	completion time of job j
t_j	-	tardiness of job j, $t_j = \max\{0, c_j - d_j\}$
u_j	-	tardiness indicator of job j, $u_j = 1$, if $t_j > 0$; $u_j = 0$, else
s_{ij}	-	start time of operation o_{ij}
c_{max}	-	makespan of a schedule, $c_{max} = \max\{c_j \mid j = 1, \dots, n\}$
f	-	due date factor
G	-	(Disjunctive) Graph $G = (N, A, E)$
(i/j)	-	node representing operation o_{ij}
0,1	-	dummy nodes
N	-	set of nodes in graph G
A	-	set of consecutive arcs in graph G
E	-	set of disjunctive arcs in graph G
$v \to w$	-	directed arc leading from node v to node w
B_j	-	completion node of job j
F_j	-	finishing node of job j
$\mathcal{LP}(u, v)$	-	longest path from node u to node v

$\mathcal{CT}(0, F)$	-	critical tree with root 0 and set of leaves F
$L(.)$	-	length of an object
$\#\ \mathcal{LP}$	-	number of longest paths
x	-	solution
X	-	solution set
c	-	cost function
NB	-	neighborhood
$\mathcal{P}(u, v)$	-	path from node u to node v
$\mathcal{C}$	-	cycle in a graph
σ_l	-	length of critical block l, i. e. the number of critical arcs in the block
τ	-	total number of critical blocks
τ_1	-	number of critical blocks with length $\sigma_l = 1$
τ_2	-	number of critical blocks with length $\sigma_l = 2$
τ_3	-	number of critical blocks with length $\sigma_l \geq 3$
$\mathcal{Z}_{NB}$	-	maximum number of neighboring schedules in NB
x_i	-	sequence of solutions $x_i = (x_i^1, x_i^2, \dots)$
Gap(j)	-	average gap of neighborhood operator j
# Eval	-	total number of schedule evaluations
# LOpt	-	total number of computed local optima
# Imp	-	average number of improving steps
Δ Gap	-	Gap difference according to absolute and relative performance quality
$h(v)$	-	head of node/operation v, i. e. $h(v) = L(\mathcal{LP}(0, v))$
$q^j(v)$	-	tail(j) of node/operation v, i. e. $q^j(v) = L(\mathcal{LP}(v, F_j))$
$PJ(v)$	-	predecessor of operation v in the job sequence (on the corresponding machine)
$SJ(v)$	-	successor of operation v in the job sequence (on the corresponding machine)
$PM(v)$	-	predecessor of operation v in the machine sequence of the corresponding job
$SM(v)$	-	successor of operation v in the machine sequence of the corresponding job
$l(v)$	-	level value of operation v
$h'(v)$	-	estimation value of the head of operation v

$q_v'^j$	-	estimation value of tail(j) of operation v
LB	-	lower bound value
$\bar{c}_{max}$	-	exact makespan (of neighboring schedule)
$\overline{TWT}$	-	exact total weighted tardiness value (of neighboring schedule)
$\bar{h}(v)$	-	exact value of the head of operation v (in neighboring schedule)
$\bar{q}^j(v)$	-	exact value of tail(j) of operation v (in neighboring schedule)
$n \mathrm{x} m$	-	size of a problem instance based on n and m
$n\!:\!m$	-	ratio of jobs to machines
P	-	pool of solutions
AHD	-	average hamming distance
$\mathcal{E}$	-	entropy
ω_{mjk}	-	number of bits with value 1
TB	-	time bound
α	-	look-ahead parameter of time bound
β	-	scaling factor
$\mathcal{I}_{LP}$	-	number of iterations in learning phase
$\mathcal{I}_{AP}$	-	number of iterations in Amplifying Procedure
$\mathcal{I}_{PR}$	-	number of iterations in Path Relinking procedure
$\mathcal{I}$	-	total number of iterations
$g(i)$	-	criticality of block i
$R_{k,i,\mathcal{I}_{LP}}$	-	k-th best rule of the learning phase, solving problem instance i with the setting of $\mathcal{I}_{LP}$ iterations
$\delta_{k,i,\mathcal{I}_{LP}}$	-	recurrence indicator of rule $R_{k,i,\mathcal{I}_{LP}}$
$\bar{\delta}_{\mathcal{I}_{LP}}$	-	average recurrence of rule selection using $\mathcal{I}_{LP}$ iterations
$\bar{\delta}^{bv}_{\mathcal{I}_{LP}}$	-	average recurrence of rule selection, based on the criterion of producing the best objective function value (bv), using $\mathcal{I}_{LP}$ iterations
$\bar{\delta}^{av}_{\mathcal{I}_{LP}}$	-	average recurrence of rule selection, based on the criterion of producing the best average objective function value (av), using $\mathcal{I}_{LP}$ iterations
TotalGap	-	total gap
# BKS	-	number of best known solutions computed
# ops	-	number of operations
f_j	-	flow time of job j, $f_j = c_j - r_j$

t_{inst}	-	time limit of the computations on instance $inst$
T_{inst}	-	summarized time limit of the computations on instance $inst$
$Z(o_{ij})$	-	priority value of operation o_{ij}
t	-	time
$\mathcal{SO}$	-	set of schedulable operations

List of Abbreviations

ACO	-	Ant Colony Optimization
ATC	-	Apparent Tardiness Cost rule
BCEI+2MT	-	Backwards Critical End Insert + 2-Machine Transpose neighborhood
BFS	-	Best Found Solution
BKS	-	Best Known Solution
BS	-	Benchmark Set
CE3P	-	Critical End 3-Permutation neighborhood
CE4P	-	Critical End 4-Permutation neighborhood
CET	-	Critical End Transpose neighborhood
CET+2MT	-	Critical End Transpose + 2-Machine Transpose neighborhood
COVERT	-	Cost Over Time rule
CSR	-	Critical Sequence Reverse neighborhood
CT	-	Critical Transpose neighborhood
DD+PT+WT	-	Combination of Due Date, Processing Time and Waiting Time rule
DICEI	-	Double Inwards Critical End Insert neighborhood
DOCEI	-	Double Outwards Critical End Insert neighborhood
ECET	-	Extended Critical End Transpose neighborhood
ECT	-	Earliest Completion Time rule
EDD	-	Earliest Due Date rule
EGRASP	-	Extended GRASP
FCFS	-	First Come First Served rule
FDC	-	Fitness-Distance-Correlation
FPTAS	-	Fully Polynomial-Time Approximation Scheme
GA	-	Genetic Algorithm

GRASP	-	Greedy Randomized Adaptive Search Procedure
ICT	-	Iterative Critical Transpose neighborhood
ILS	-	Iterated Local Search
JSP	-	Job Shop scheduling Problem
JSPTFT	-	JSP with Total Flow Time objective
JSPTWT	-	JSP with Total Weighted Tardiness objective
JSPTWU	-	JSP with weighted number of tardy jobs objective
LOPA	-	Longest Path sorting strategy
LPT	-	Longest Processing Time rule
MDD	-	Modified Due Date rule
ODD	-	Operational Due Date rule
PPC	-	Production Planning and Control
PTAS	-	Polynomial-Time Approximation Scheme
PT+PW	-	Combination of Processing Time and Waiting Time rule
PT+PW+ODD	-	Combination of Processing Time, Waiting Time and Operational Due Date rule
PT+WINQ+SL	-	Combination of Processing Time, Work In Next Queue and Slack rule
RCL	-	Restricted Candidate List
SCEI	-	Single Critical End Insert neighborhood
SL	-	Slack rule
SPT	-	Shortest Processing Time rule
SPT+S/RPT	-	Combination of Shortest Processing Time and Slack per Remaining Processing Time rule
S/OPN	-	Slack per number of operations rule
TS	-	Tabu Search
TFT	-	Total Flow Time
TWT	-	Total Weighted Tardiness
TWU	-	Total Weighted number of tardy jobs
VNS	-	Variable Neighborhood Search
WCR	-	Weighted Critical Ratio rule
WEDD	-	Weighted Earliest Due Date rule
WI	-	Weight rule
WINQ	-	Work In Next Queue rule

WMDD	-	Weighted Modified Due Date rule
WRA	-	Weighted Remaining Allowance rule
WSL+WSPT	-	Combination of Weighted Slack and
	-	Weighted Shortest Processing Time rule
WSPT	-	Weighted Shortest Processing Time rule

Chapter 1

Introduction

Scheduling is a natural working process that humans do every day. Planning daily life by scheduling daily tasks is mostly subconscious and not directed by a specific procedure. For example, when should I wash the dishes? Should I buy the groceries before or after doing the dishes? For scheduling such tasks, several relationships have to be considered. The precedence relations between washing clothes, drying clothes and ironing them are simple to understand. Furthermore, there could be a number of tasks that cannot be performed simultaneously. Showering and cooking should not be done at the same time. Otherwise, the result could lead to an unsatisfying meal.

Scheduling one's daily life is mostly based on experience according to old schedules. Usually, there is neither the application of a sophisticated decision procedure, nor the need of a computation technique. The main reason for this is that there is no real economic objective function to be optimized.

In the scientific theory, scheduling is formalized to the task of allocating finite resources over time to accomplish a given set of tasks [110]. Hence, scheduling is a crucial decision process with the aim of improving the efficiency of operating systems. In business economics, fields of application often arise in manufacturing plants or service systems [109]. Usually, the problem structure incorporates many tasks to be executed as well as other complicated features that have to be considered. Every task is divided into operations that have to be processed on one specific resource or on a resource from a set of feasible resources. Additionally, objective functions take an important role. Tackling these problems in terms of science means finding a schedule that fulfills the objective(s) in the best way possible.

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_1,

The focus in this thesis is on the job shop scheduling problem. This problem is to find a schedule for processing n jobs (tasks) on m machines (resources) such that the technological sequence of each job is adhered to. Moreover, the problem is solved with regard to an objective function so that a schedule is obtained with e. g. minimum total costs of the production.

The research on job shop scheduling problems started a long time ago. The first studies are published in the 1950s (see e. g. [4, 65]). Over time, job shop scheduling problems never lost its attractiveness. Due to the challenging problem instance ft10 [46] which was unsolved for more than 20 years, a lot of research was dedicated to the problem structure, as well as efficient solution procedures.

Due to the fact that job shop problems could be found in the business environment of producing goods, the problem has been tackled with the goal of minimizing the completion time of the last job, also known as the minimization of the makespan. The purpose of this objective is to obtain schedules that maximize the usage of the machine capacities as well as to minimize the idle times of the machines. The associated total costs for processing all jobs become minimal.

In this thesis, the structure of the jobs is extended by the introduction of a due date. This due date is critically important, since violating the due date of a job can cause significant penalty costs. The economic relevance of these due dates is described below with specific examples. However, meeting due dates becomes a more and more challenging task [24]. Many schedules cannot ensure that all jobs are completed on-time. Furthermore, the tardiness of a job can even be comparatively more expensive if the associated penalty costs are higher. As a result, every job has got assigned a weight that indicates its importance among the jobs. In summary, the job shop scheduling problem with minimizing the total weighted tardiness is the considered optimization problem in this thesis.

Other objective functions that take due dates into account minimize the number of tardy jobs or maximize the total punctuality of the jobs (i. e. minimizing the deviation from the due date). However, these objectives are omitted in this thesis, since the total weighted tardiness in a resulting schedule is more significant for the decision-maker. It corresponds to the β-service-level in a logistic system, which measures the average order fill rate.

In the following examples, three applications are presented where job shop scheduling problems are identifiable.

Production Planning and Control. Value creation in industrial companies bases on the process of production planning and control (PPC). In general, PPC systems have a hierarchical structure joining several planning tasks that are based on each other. One planning task, integrated in the operational level of the hierarchical planning process, is the shop floor scheduling [26]. Usually, in the context of manufacturing systems, it is referred to as a machine scheduling problem. After the determination of the master schedule program and the control of the required materials, production orders are released and have to be processed on the machines.

Job shop scheduling problems typically arise in the production environment with a character of high variety and low volume [61], i. e. the entirety of production orders is very diverse. In this problem, the production orders are also denoted as jobs. Due to the specific machine sequence which every job has to comply, machines and workplaces in the production system are not connected, and buffers are used for storing the incoming jobs. Economic success mainly depends on the obtained schedules for processing the jobs on the machines. Already in 2003, an international study showed that the costs of a product and its delivery time are nearly equivalent factors for economic success of the company [6]. High quality schedules according to due date related objectives can improve delivery performance, reduce inventory costs, and are very important to manufacturers in today's time-based competition; therefore, minimizing the total weighted tardiness has become an important objective function for solving the job shop scheduling problem.

Planning and Scheduling in Supply Chains. Supply chains represent networks of interconnected facilities or companies. Players involved in such a network participate in the value-added process of manufacturing several products [34]. The entire planning processes are designed for the complete network in order to achieve the best possible economic success across the companies.

The production planning and scheduling in supply chains are usually decomposed into two planning problems. Medium term planning problems and short term planning problems are solved consecutively [77]. The goal of medium term planning is to determine production quantities at the production facilities. The result produces a master schedule program for each of the connected players in the supply chain. Note that the programs are subject to a planning horizon of several weeks or months. The subsequent short term planning consists of solving a scheduling problem. Here, every player in the supply chain is considered on its own. The scheduling problem,

e. g. job shop scheduling problem, has to be solved based on the given production orders from the master schedule program. The problem structure is more detailed and the results cover, at most, a schedule for a week. It is important that the considered objective function is based on the individual due dates of the production orders. In this way, the interaction of the two planning problems is ensured.

Both medium term and short term planning can be solved sequentially, for example by using a feedback loop. The planning result of the first problem delivers the input of the second planning problem [77].

Single-Track Railway Scheduling. This problem is concerned with scheduling trains between different locations that are connected with only one rail track [27, 104]. Because of the single tracks, the passing of trains is only carried out on stations, sidings or double-track sections. Furthermore, a timetable of the trains is given in advance, i. e. the starting and ending location of the rides, the arrival times at and departure times from each station. Previous work to this scheduling problem (see e. g. [104]) is often related to the objective of minimizing the overall delay. Recent publications (see e. g. [72]) are turned towards the objective function of minimizing the total weighted tardiness.

The problem structure can be transferred into a job shop scheduling problem [104]. For this purpose, the trains represent the jobs. The sequence of visited locations and used track sections indicates the machine sequence of the jobs. Passing one single track section corresponds to the processing of a job on a machine. The planned arrival times on the locations are taken as due dates. They can be seen either as operation-specific dates (arrival time at a location) or as job-specific dates (arrival time at the final destination). Additional constraints, such as blockings of track sections/machines, have to be taken into consideration. After solving the single-track railway scheduling problem, the resulting starting times of the jobs on the machines correspond to the actual departure times of the trains from the stations. Therefore, the total weighted tardiness is an adequate measure for the practicability of the given timetable.

1.1 Aims and Contributions of the Thesis

Even though the problem structure is relatively easy to describe, the search for an optimal schedule is very difficult. Due to the variety of possible job sequences on

the machines, the number of solutions is $n!^m$. For example, if 10 jobs need to be processed on 10 different machines, the cardinality of the solution set is 3.95×10^{65} solutions. Fortunately, this set contains a large subset of infeasible solutions as a result of violated precedence constraints. But, even if it is possible to estimate that e. g. half of them are infeasible and could be discarded, the set of solutions is still extremely large and could not be handled in a complete enumeration algorithm. Actually, modern exact algorithms cannot solve problem instances with more than 15 jobs and 15 machines [24]. For this reason, the main focus in research for solving job shop scheduling problems is on the field of heuristics and metaheuristics.

In the last 30 years, research has produced many metaheuristic search concepts. In general, scientists distinguish between two classes of metaheuristics: population-based and single-solution metaheuristics [48]. The first class, involving e. g. ant colony optimization and genetic algorithms, is mostly based on nature-inspired ideas. The search process is simultaneously carried out from a set of solutions, often also referred to as population. The second class, involving e. g. simulated annealing and tabu search, is based more on theoretic aspects of combinatorics. The search concept is characterized by a single search trajectory. The different solution concepts can be further classified, e. g. based on the number of neighborhood structures or the objective function [21]. In recent years, the combination of several metaheuristic search concepts, also referred to as hybrid metaheuristics, has become popular (see [20] for a survey).

For solving the job shop scheduling problem with minimizing the makespan as objective, Vaessens et al. [134] have studied the performance of several heuristics and metaheuristics proposed in literature. The result of this study is that local search based heuristics commonly produce the best results. This kind of solution procedures is classified into the class of single-solution metaheuristics mentioned above. Even though the study of Vaessens et al. [134] is more than 15 years old and other solution techniques have been developed by now, the results are still relevant [143]. Local search algorithms also play an important role with regard to the objective of total weighted tardiness minimization. Some of the best arguments are the recently published solution procedures for solving this optimization problem (see e. g. [57, 91, 153]). All of them incorporate local search as one elementary component (see Section 3 for the complete literature review).

With this motivation, the primary goal of the thesis is to propose a new solution

procedure based on local search for solving the job shop scheduling problem with total weighted tardiness objective. For the development of a powerful procedure, the following issues are investigated forming the major scientific contributions of this thesis:

1. Adapting the disjunctive graph for modeling the problem structure: Important to note is that some publications introducing the disjunctive graph for the job shop scheduling problem with total weighted tardiness objective already exist (see e. g. [76]); however, there is still a lack between representation form and problem structure. Therefore, this thesis presents a new disjunctive graph representation. Based on this modeling framework, a new structural feature called critical tree is introduced. The critical tree aims at a compact aggregation of critical arcs and blocks which serve as starting points in the application of neighborhoods.

2. Analyzing several neighborhood operators: The neighborhood operator plays a fundamental role within local search. Therefore, existing and newly defined operators are applied to the problem under consideration. The properties and the performance of all neighborhoods will be investigated.

3. Developing and introducing a new concept for the assessment of newly generated schedules: An important feature for the successful application of local search algorithms is to get a fast evaluation whether or not the new schedule is improving or not. The faster and the more accurate, the more efficient is the local search.

4. Developing and implementing of advanced search concepts: A simple local search procedure cannot deliver the performance quality for solving hard and large problem instances effectively. For this reason, several ideas will be proposed for guiding and improving the search process.

All results contribute in the development of a new solution procedure. The comparison with other existing metaheuristics for solving the job shop scheduling problem with total weighted tardiness objective provides an assessment of its competitiveness.

1.2 Overview of the Thesis

This thesis is structured into eight chapters. In the following chapter, the problem structure is introduced in detail. Alongside an explanation of the machine scheduling theory, the mathematical model and the complexity classification are presented for the problem under consideration. The first two aforementioned issues form the bulk of this chapter: the modeling as disjunctive graph and the concept of the critical tree.

In Chapter 3, the previous research work and papers related to the job shop scheduling problem with total weighted tardiness objective are reviewed.

Chapter 4 presents a broad analysis of neighborhood operators, which can be used in local search algorithms for solving the problem presented. Therefore, several neighborhoods are defined and classified according to their perturbation schemes. Important characteristics are investigated analytically. The resulting performance quality within a local search procedure is assessed with the help of a empirical study.

In Chapter 5, the assessment of the solution quality in terms of its objective function value is focused. Different concepts for a fast estimation are known from literature and reviewed in this chapter. Furthermore, a new concept is introduced and the resulting performance is analyzed within a computational study.

Chapter 6 is divided into two topics. First, an overview of different metaheuristic search concepts is presented. A brief review of the fitness landscape theory illustrates the complexity of developing a powerful solution procedure. Implications and recommendations known from the fitness landscape of the job shop scheduling problem with minimum makespan objective are presented. Second, the algorithmic concept of the new solution procedure is described. Additional ideas of search mechanisms are presented in order to obtain more effectiveness in the entire search process.

Chapter 7 provides an extensive computational study. The aim is to assess the performance quality of the proposed solution procedure based on standard benchmark instances. Furthermore, related optimization problems are considered in order to check the range of application.

Chapter 8 summarizes the gained results and concludes the thesis.

Since several computational experiments are performed, the general settings are as follows. The computations are conducted on a Desktop PC with Intel Core i7-2600

(3.4 GHz), 8 GB RAM and Linux openSUSE 11.4 (x86_64) with the exception of CPLEX computations which are conducted on a Desktop PC with Intel Core 2 Duo E8400 (3.0 GHz), 2 GB RAM, using ILOG CPLEX Optimization Studio 12.2. The neighborhoods, the estimation procedures and the new proposed solution procedure are implemented in LiSA 3.0 - Library of Scheduling Algorithms[1]. Moreover, all computations with limited time or resource capacity are carried out on a single processor.

1.3 Publications

Some results of this thesis have already been published by the author. An overview of submitted papers is given in the following list:

- J. Kuhpfahl and C. Bierwirth: A Study on Local Search Neighborhoods for the Job Shop Scheduling Problem with Total Weighted Tardiness Objective, submitted for publication.
- J. Kuhpfahl and C. Bierwirth: A GRASP approach for the Job Shop Scheduling Problem with minimizing the Total Weighted Tardiness, In: Proceedings of 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2013), 613-616.
- R. Zhang and J. Kuhpfahl: Corrigendum to "A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted objective" [Computers & Operations Research 38(2011) 854-867], Computers & Operations Research 40 (2013), 2816.
- J. Kuhpfahl and C. Bierwirth: A New Neighbourhood Operator for the Job Shop Scheduling Problem with Total Weighted Tardiness Objective, In: Applied Mathematical Optimization and Modelling - APMOD 2012 Extended Abstracts, 204-209.
- J. Kuhpfahl and C. Bierwirth: Computational Comparison of Neighbourhoods for the Job Shop Scheduling Problem with Total Weighted Tardiness Objective, In: Proceedings of 5th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2011), 536-538.

[1] http://lisa.math.uni-magdeburg.de/

Chapter 2

Job Shop Scheduling - Formulation and Modeling

Machine scheduling problems arise in different structures and settings. In order to obtain a clear comprehension of the contents in this thesis, the introduction of the job shop scheduling problem with total weighted tardiness objective will be detailed later in this chapter. In the first section, the considered problem structure will be described. After that, the differentiation from other scheduling problems will be presented in Section 2.2. The Section 2.3 will then provide the mathematical model and basic information about the complexity of the problem. The modeling of the problem as a disjunctive graph and the resulting implications are important fundamentals for topics which will later be addressed. For this purpose, Section 2.4 and 2.5 contain the concept of the graph representation, the critical tree and the gained conclusions. Section 2.6 provides an illustration of the presented concepts with the help of the problem instance ft06.

2.1 Problem Structure

Following the introduction of French [47], the job shop scheduling problem (JSP) consists of a finite set of jobs $J = \{1, 2, \dots n\}$ and a finite set of machines $M = \{1, 2, \dots m\}$. More precisely, every job consists of a finite set of operations. The processing of an operation has to be performed on a preassigned machine, i. e. the i-th operation of job j, denoted by o_{ij}, is processed on machine $\mu_{ij} \in M$. The operation order of each job is fix, i. e. the technological machine sequence given for

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_2,

every job has to be taken into account. The aim is to find a schedule for processing these n jobs on the m machines. Further conditions are as follows:

- The processing of the i-th operation of job j takes $p_{ij} > 0$ time units.
- Each operation has to be processed exactly once.
- Preemption is not allowed while operations are being processed.
- Processing operations may not overlap with one another.
- There is no machine-dependent or sequence-dependent setup time.
- Machines must always be available.

Beyond the described structure, the thesis focuses on the standard, dynamic version of JSP with job weights and due dates. In the standard JSP, every job has to be processed on every machine exactly once. This means that every job consists of m operations in which every pair of these operations is processed on different machines. This standardization helps to comprise a broad range of problem instances. If a job is not executed on machine $k \in M$ in a problem instance, the corresponding operation is simply neglected in the technological machine sequence. To be consistent with the assumption that every job passes every machine, one can alternatively set the processing time of these virtual operations to zero and add them at the first position of the technological sequence in order to avoid blockings.

One can furthermore distinguish between static and dynamic JSP. In the dynamic JSP, every job has an assigned release time $r_j \geq 0$ so that the first operation cannot start before r_j. In the static JSP, all jobs are available at the beginning of the planning horizon, i. e. $r_j = 0, \forall j \in J$. Note that the dynamic JSP covers the static JSP.

An additional attribute of a job j is its weight w_j which represents the job's relative importance in comparison to other jobs. Furthermore, every job has a due date $d_j \geq 0$ which should, but does not necessarily have to, be fulfilled in a schedule.

The quality of a solution is assessed by the obtained total weighted tardiness, defined as $TWT = \sum_{j=1}^{n} w_j \cdot t_j$, where $t_j = \max\{0, c_j - d_j\}$ is the resulting tardiness of job j in a schedule, and c_j its completion time. The resulting minimization problem is referred to as JSPTWT. The solution of a problem instance can be illustrated in a Gantt-Chart, as shown in Section 2.6.

The problem structure of the JSPTWT involves two questions: 1. Which job sequences should be determined on the machines? 2. When does the processing of each operation start? In general, both questions are answered at the same time. On the one hand, by fixing the start time of each operation, the job sequences can be derived from this outcome. On the other hand, given the job sequence on every machine, the start times of the operations are calculated by their earliest possible beginning with respect to the precedence relations given by their technological sequence.

In scheduling theory, there are three different classes of schedules [110]: semi-active, active and non-delay schedules. A schedule is called semi-active if and only if no operation could start earlier than its assigned start time without changing the corresponding job sequence on the machine. A schedule is called active if and only if there exist no earlier start of an operation without delaying the start of another operation. Finally, a schedule is called non-delay if and only if no machine is kept idle while an operation is schedulable on this machine at the same time. Based on these definitions, the following inclusions are observed: the class of semi-active schedule includes the set of active schedules, and the class of active schedules includes every non-delay schedule.

Since the total weighted tardiness objective is a regular measure, i. e. it is a non-decreasing function of the completion times, it is clear that at least one optimal solution corresponds to an active schedule, without necessarily having to be a non-delay schedule [47]. For this reason, the contents and the results in this thesis are applied to active schedules.

2.2 Classification into the Scheduling Theory

In general, scheduling problems cover a broad range of practical problems. As mentioned in Chapter 1, the process of production planning and control in manufacturing systems is one motivation. The occurring scheduling problems are summarized in the class of machine scheduling. The considered JSPTWT is also classified as a machine scheduling problem. For this reason, this section will only provide a short introduction to machine scheduling and differentiate between JSPTWT and other related problems. There are, however, other classes and fields of applications in industrial and service settings, e. g. timetabling, sports scheduling, crew scheduling, scheduling in health care, and many more [109].

Since the practical applications are manifold, there are different types of machine scheduling problems (see e. g. [84, 110]). Graham et al. [58] have proposed a classification scheme based on a three field notation $\alpha \mid \beta \mid \gamma$. The α-field describes the machine environment of the considered problem. Processing restrictions and additional constraints are indicated in the β-field. The last field γ specifies the objective function, which should be fulfilled in the best possible way.

The simplest form of the machine environment describes a planning problem with a single machine [110]. Other problem structures involve e. g. identical machines or flexible structures in the technological sequence. Closely related problems to the job shop problem are the flow shop problem and the flexible job shop problem. The flow shop scheduling problem is a specialization of the job shop where all jobs have the same technological sequence. The flexible job shop scheduling problem is a generalization of the job shop. Instead of needing exactly one specific machine to process an operation, there is a set of identical machines available for carrying out this single task.

Further restrictions and constraints occur in e. g. sequence-dependent setup times or blocking constraints. The introduction of setup times describes a scenario in which the setup activities on certain machines strongly depend on the job order given in the schedule. A blocking constraint results from limited buffers in front of the machines, i. e. a job blocks the current machine if the buffer of the subsequent machine is full. More details as well as other additional characteristics can be found in [110]. Note that the consideration of due dates is usually not additionally specified in the β-field.

In the last 5 decades, job shop scheduling has mainly focused on the minimization of the makespan c_{max}. The corresponding optimization problem is also referred to as minimum makespan job shop scheduling problem. This objective function is motivated by the fact that minimizing the makespan directly corresponds to a high utilization of the machines, and thus, decreases production costs. As mentioned before, the considered total weighted tardiness objective leads to a minimization of penalty costs. This objective function belongs to the class of min-sum objectives such as the minimization of the total completion times ($\sum_{j=1}^{n} c_j$), the minimization of the total flow time ($\sum_{j=1}^{n}(c_j - r_j)$) or the number of tardy jobs ($\sum_{j=1}^{n} u_j$).

In this thesis, according to the classification of Graham et al. [58] and the described problem structure in Section 2.1, the considered optimization problem is as follows:

$$J \mid r_j \mid \sum w_j t_j$$

2.3 Mathematical Model and Complexity

Based on the notation introduced in Section 2.1, the mathematical model of the JSPTWT can be formulated as a disjunctive program [2]:

$$\min \sum_{j=1}^{n} w_j t_j \tag{2.1}$$

$$\text{s.t.} \quad s_{ij} + p_{ij} \leq s_{i+1,j} \qquad \forall\, j \in J, \forall\, i \in M \backslash \{m\} \tag{2.2}$$

$$s_{ij} + p_{ij} \leq s_{k,l} \vee s_{kl} + p_{kl} \leq s_{i,j} \qquad \forall\, j,l \in J, \forall\, i,k \in M \backslash \{m\} \tag{2.3}$$

$$\text{with } j \neq l, \text{ and } \mu_{ij} = \mu_{kl}$$

$$t_j \geq s_{mj} + p_{mj} - d_j \qquad \forall\, j \in J \tag{2.4}$$

$$t_j \geq 0 \qquad \forall\, j \in J \tag{2.5}$$

$$s_{1j} \geq r_j \qquad \forall\, j \in J \tag{2.6}$$

The decision variable s_{ij} represents the start time for operation o_{ij}. The constraints (2.2) ensure the technological sequence given for every job. The disjunctive constraints (2.3) capture the sequencing problem on every machine. The constraints (2.4) and (2.5) are part of the program to measure the resulting tardiness of each job. Finally, constraints (2.6) ensure that a job cannot start before its release time, and thus, capture the non-negativity of the decision variables s_{ij}. In the objective function (2.1) the weighted sum of the tardiness has to be minimized.

The disjunctive constraints (2.3) can be reformulated as a set of linear constraints with the help of the M-method [37]. For this purpose, additional binary variables have to be introduced indicating the different precedence relations on the machines. For a problem instance with 10 jobs and 5 machines, this linearization leads to a mixed-integer program with 285 decision variables, of which 225 are binary variables, and 510 constraints.

The above formulated mathematical model of the JSPTWT can be easily converted for solving the minimum makespan JSP. In addition to the swap of the objective function toward minimizing the makespan c_{max}, the following set of constraints replaces constraints (2.4) and (2.5):

$$c_{max} \geq s_{mj} + p_{mj} \qquad \forall\, j \in J \tag{2.7}$$

Although the mathematical models of the JSPTWT and the minimum makespan

JSP are very similar, the JSPTWT is more complex. If the JSPTWT is already restricted to 1 machine (i. e. it becomes a single machine scheduling problem) and the release times of the jobs are set to zero, than the scheduling of n jobs with TWT objective is an $\mathcal{NP}$-complete problem [83, 86]. With the integration of release times $r_j \geq 0$ the complexity result is the same [110]. On the other hand, scheduling n jobs on a single machine without release times while focusing on the minimization of the makespan is easy to solve. Every arbitrary sequence that corresponds to an active schedule is an optimal solution.[2] The problem classification into classes of different complexity is also demonstrated by the following example: processing two jobs on m machines with release times and unit processing times is strongly $\mathcal{NP}$-hard for the TWT objective [133], whereas this problem is solvable in polynomial time with regard to the minimization of the makespan [132]. Further complexity results are presented by Knust [74].

A standard JSP, as defined in Section 2.1, is $\mathcal{NP}$-hard for the TWT objective as well as makespan objective [110]. Since the optimal schedule for the TWT objective depends on the completion time of all jobs, finding this solution and proving its optimality is usually quite hard and takes a considerable amount of time. The comparison of the needed computation time according to TWT objective and makespan objective is additionally demonstrated by solving six small problem instances by using CPLEX 12.2. In the instances la01-la03 10 jobs have to be processed on 5 machines, whereas in the instances abz05, abz06, ft10, 10 jobs have to be processed on 10 machines. Further details of the instances as well as their modification to JSPTWT instances are presented in Section 7.1.1. For these computations, it is necessary to mention that the due dates of the jobs are generated with the help of

			TWT objective		c_{max} objective	
Instance	n	m	Opt. value	Comp. time	Opt. value	Comp. time
la01	10	5	2.299	43,5 s	666	9,6 s
la02	10	5	1.762	74,5 s	655	44,3 s
la03	10	5	1.951	47,6 s	597	16,2 s
abz05	10	10	1.403	6.905,4 s	1.234	73,7 s
abz06	10	10	436	371,8 s	943	9,5 s
ft10	10	10	1.363	3.510,0 s	930	2.730,2 s

Tab. 2.1: Results of CPLEX 12.2 for several problem instances.

[2]Note that every semi-active schedule is also active in this special case.

due date factor $f = 1.3$. Tab. 2.1 shows the objective function value of the optimal solutions and the computation times for both objective functions. The reported computation times for the TWT objective are significantly higher than those of the makespan objective. Even though the problem instances are small in comparison to real world instances, the computation time is up to more than 100 minutes. These results motivate again to develop powerful heuristics and metaheuristics for tackling the JSPTWT.

2.4 The Disjunctive Graph Model

The disjunctive graph model is a fundamental representation form of the minimum makespan JSP [2, 18]. Efficient heuristic concepts are based on this network model. As a consequence, the related literature already provides a disjunctive graph model for the JSPTWT (see e. g. [76, 111]). However, the presented models do not enable to determine the solution quality in terms of the resulting tardiness of a job. For this reason, a proper transformation of the disjunctive graph model for the JSPTWT will be presented in this section.

In general, problem instances of the minimum makespan JSP can be represented by a disjunctive graph $G = (N, A, E)$, where the i-th operation of job j is denoted by node $(i/j) \in N$. The set of nodes is completed by two dummy nodes, namely the source node 0 and the sink node 1. The set of operations of job j is connected by directed arcs $(i/j) \rightarrow (i+1/j) \in A$ representing its technological machine sequence. For every job j, there is one directed arc connecting the source node with the first operation node, $0 \rightarrow (1/j) \in A$, and one further directed arc connecting the last operation node with the finishing node, $(m/j) \rightarrow 1 \in A$. A weight is given for every arc $(i/j) \rightarrow (i+1/j) \in A$ which represents the processing time of operation (i/j). Arcs $0 \rightarrow (1/j)$ have a weight 0. Finally, arcs $(m/j) \rightarrow 1$ are weighted by

	o_{1j}		o_{2j}		o_{3j}				
Job j	μ_{1j}	p_{1j}	μ_{2j}	p_{2j}	μ_{3j}	p_{3j}	w_j	r_j	d_j
1	1	3	2	4	3	4	1	0	12
2	3	2	1	3	2	4	2	2	15
3	1	5	3	5	2	2	1	2	18

Tab. 2.2: Data to the example 3x3 instance.

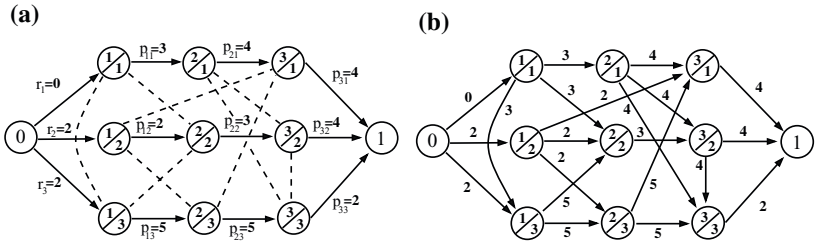


Fig. 2.1: Minimum makespan JSP: (a) Disjunctive graph G (dashed line = pair of disjunctive arcs), (b) feasible solution as directed graph G'.

the processing time of operation (m/j). The precedence relation between operations $(i/j), (k/l) \in N$ of different jobs being processed on the same machine are represented by pairs of disjunctive arcs $\{(i/j) \to (k/l), (k/l) \to (i/j)\} \in E$. Again, a weight is assigned to each arc $(i/j) \to (k/l)$ of the set E that represents the processing time of operation (i/j). By choosing suitable subsets of E, all possible job sequences on the machines can be displayed. Furthermore, by selecting one arc of each pair of disjunctive arcs, a subset E' of E is obtained, thereby describing a feasible schedule if and only if the corresponding graph $G' = (N, A, E')$ is acyclic [2].

Fig. 2.1(a) shows the disjunctive graph for an instance with 3 jobs and 3 machines and data summarized in the first 7 columns of Tab. 2.2. A solution of this problem is shown in Fig. 2.1(b). The makespan of this schedule corresponds to the length of one longest path from source 0 to sink 1. Note that there can be more than one longest path in the directed graph. For the example in Fig. 2.1(b), a longest path is $0 \to (1/1) \to (1/3) \to (2/3) \to (3/1) \to 1$; another one is $0 \to (1/1) \to (1/3) \to (2/2) \to (3/2) \to (3/3) \to 1$. Both have length 17.

For modeling the JSPTWT in an appropriate manner, the described disjunctive graph is modified in the following way. For every job j, an individual completion node B_j and an individual finishing node F_j are introduced (instead of the common sink node 1)[3]. Further arcs $0 \to F_j$ and $B_j \to F_j$ are added to the set A. Weights 0 and $-d_j$ are assigned to these arcs to compute the tardiness of job j as the non-negative distance between its completion time and due date. Additionally, the weight of arc $0 \to (1/j) \in A$ represents the release time of the job j. Fig. 2.2(a) shows the modified disjunctive graph $G = (N, A, E)$ for the considered example above with the additional data of weights, release times and due dates (see the right

[3]The introduction of the completion nodes B_j has already been proposed by Pinedo and Singer [111]. In their disjunctive graph, these nodes are denoted as sink nodes V_j.

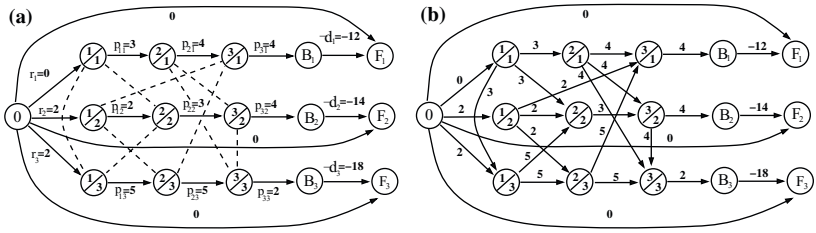


Fig. 2.2: JSPTWT: (a) Disjunctive graph G (dashed line = pair of disjunctive arcs), (b) feasible solution as directed graph G'.

part of Tab. 2.2). Like in the disjunctive graph representation of the minimum makespan JSP, a feasible solution is obtained by selecting one arc of each pair of disjunctive arcs from the set E (see Fig. 2.2(b) for the corresponding directed graph $G' = (N, A, E')$). In order to assess the solution quality of the underlying schedule, a longest path for every job j has to be computed, i. e. starting from 0 and ending in its finishing node F_j. It can be observed that the value of the longest path leading from 0 to B_j represents the completion time of job j, and the length of the longest path from 0 to F_j corresponds to the tardiness of job j. For example, considering job 1 in the solution of Fig. 2.2(b), the longest path leading from 0 to B_1 is $0 \rightarrow (1/1) \rightarrow (1/3) \rightarrow (2/3) \rightarrow (3/1) \rightarrow B_1$ and is of length 17. This value expresses the completion time of the third operation of job 1, and thus, equals to the completion of the whole job. The longest path leading from 0 to F_1 is $0 \rightarrow (1/1) \rightarrow (1/3) \rightarrow (2/3) \rightarrow (3/1) \rightarrow B_1 \rightarrow F_1$ and is of length 5. Accordingly, the tardiness of job 1 is given by $t_1 = \max\{0, c_1 - d_1\} = \max\{0, 17 - 12\} = 5$.

In contrast to other disjunctive graph models for the JSPTWT which have been proposed in the literature (see e. g. [76, 107]), the presented model incorporates the information whether a job is tardy or not.

2.5 The Concept of the Critical Tree

The disjunctive graph representation of the problem structure and the derivation of feasible schedules help to identify important direct precedence relations on machines that determine the solution quality. These kinds of precedence relations are represented by arcs on the longest path, also known as critical arcs.

Following the concept of Nowicki and Smutnicki [101], a longest path can be

decomposed into critical arcs and critical blocks. An arc that (i) belongs to a longest path and (ii) connects two operations that are processed consecutively on a machine is referred to as a critical arc. A critical block is defined as the sequence of operations connected by a maximum chain of critical arcs according to one longest path. The identification of critical arcs is used for the definition of local search based neighborhood operators (see Chapter 4), because the reversal of one critical arc generates a new schedule that does not violate any technological machine sequence of a job. Furthermore, only the reversal of critical arcs can improve the solution. The reversal of arcs that do not belong to any longest path as well as represent precedence relations on machines, do not influence any origin longest path, and thus, cannot reduce their length.

While assessing the total weighted tardiness measure of the schedule represented in G', all its critical arcs and blocks are identified during the construction of the n longest paths. The composition of all longest paths results in a tree graph with root 0 and n leafs. This tree is called the critical tree, and is formally defined as follows:

Definition 1 (Critical Tree)**.** *Given the directed graph G' of a feasible schedule and a weight w_j for each job j. The composition of all longest paths $\mathcal{LP}(0, F_j)$, each leading from 0 to F_j, is called* ***critical tree*** *$\mathcal{CT}(0, F)$. The length of the tree is defined as:*

$$L(\mathcal{CT}(0, F)) = \sum_{j=1}^{n} w_j \cdot L(\mathcal{LP}(0, F_j))$$

The critical tree $\mathcal{CT}(0, F)$ derived for solution G' of the example instance (see Fig. 2.2(b)) is shown in Fig. 2.3. Since there are three jobs, it is composed of three longest paths. For job $j = 1$, path $\mathcal{LP}(0, F_1) = 0 \rightarrow (1/1) \rightarrow (1/3) \rightarrow (2/3) \rightarrow (3/1) \rightarrow B_1 \rightarrow F_1$ is determined which is of length $L(\mathcal{LP}(0, F_1)) = 3{+}5{+}5{+}4{-}12 = 5$. Consequently, job 1 is tardy by 5 time units. Job 2 is also tardy, as indicated by

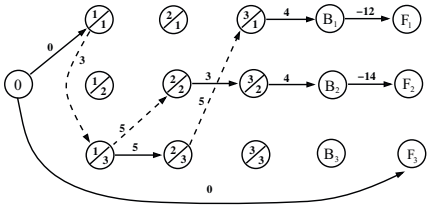


Fig. 2.3: Graph structure of all longest paths resulting in the critical tree $\mathcal{CT}(0, F)$ (dashed arc = critical arc).

the fact that arc $B_2 \rightarrow F_2$ belongs to its longest path. Only job 3 is on-time. The total weighted tardiness of this schedule is determined by the length of the tree:

$$\begin{aligned} L(\mathcal{CT}(0,F)) &= w_1 \cdot \mathcal{LP}(0,F_1) + w_2 \cdot \mathcal{LP}(0,F_2) + w_3 \cdot \mathcal{LP}(0,F_3) \\ &= 1 \cdot (3+5+5+4-12) + 2 \cdot (3+5+3+4-14) + 1 \cdot 0 = 7 \end{aligned}$$

Due to the construction of the longest paths, every critical arc and critical block is included in $\mathcal{CT}$ and causes a tardiness of at least one job. In the presented example, there are three critical arcs which are represented as dashed arcs in Fig. 2.3. Note that critical blocks can be included within each other. Even though the illustration in Fig. 2.3 would lead one to assume that there are only two critical blocks, the graph contains three of them. The critical block $(1/1) \rightarrow (1/3) \rightarrow (2/2)$ on $\mathcal{LP}(0,F_2)$ covers the block $(1/1) \rightarrow (1/3)$ from $\mathcal{LP}(0,F_1)$. However, both blocks are used separately, since it is possible to derive different neighboring solutions from them (e. g. by applying the SCEI neighborhood operator, see Section 4.2).

In summary, the critical tree derived from the schedule G' offers an appropriate framework for the application of local search based neighborhoods to the JSPTWT. In addition, the concept of the critical tree can be useful for assessing other objectives of the represented schedule. For example, the critical tree $\mathcal{CT}(0,B)$ for the solution in Fig. 2.2(b) expresses the value of the total weighted flow time of the schedule. Moreover, the transfer of the concept to the minimum makespan JSP is possible. The additional gain from the construction of $\mathcal{CT}(0,1)$ is, however, comparatively negligible.

2.6 Exemplification on the instance ft06 ($f = 1.3$)

The following example provides a demonstration of the disjunctive graph model and the concept of the critical tree. The problem instance ft06 consists of 6 jobs that have to be processed on 6 machines [46]. The complete instance data is reported in Tab. 2.3. Since due dates, weights and release times are not determined, they are computed according to the specifications of Singer and Pinedo [121] (see Section 7.1.1 for the detailed computations). For this demonstration, the due date factor $f = 1.3$ is used to generate the due date of every job.

The complete data generated for ft06 ($f = 1.3$) enables to solve the problem

	o_{1j}		o_{2j}		o_{3j}		o_{4j}		o_{5j}		o_{6j}				
Job j	μ_{1j}	p_{1j}	μ_{2j}	p_{2j}	μ_{3j}	p_{3j}	μ_{4j}	p_{4j}	μ_{5j}	p_{5j}	μ_{6j}	p_{6j}	w_j	r_j	d_j
1	3	1	1	3	2	6	4	7	6	3	5	6	4	0	33
2	2	8	3	5	5	10	6	10	1	10	4	4	2	0	61
3	3	5	4	4	6	8	1	9	2	1	5	7	2	0	44
4	2	5	1	5	3	5	4	3	5	8	6	9	2	0	45
5	3	9	2	3	5	5	6	4	1	3	4	1	2	0	32
6	2	3	4	3	6	9	1	10	5	4	3	1	1	0	39

Tab. 2.3: Data of the instance ft06 ($f = 1.3$).

instance as standard JSP as well as JSPTWT. Fig. 2.4 shows the Gantt-Chart of an optimal schedule based on the minimum makespan objective. This best solution has a makespan of 55 indicated by the completion of the operation (6/1). The TWT value of the presented schedule is 161. In Fig. 2.5, the Gantt-Chart of an optimal schedule based on the TWT objective is presented. In this optimal solution, the completion of the jobs leads to a total weighted tardiness of 52. The makespan of this

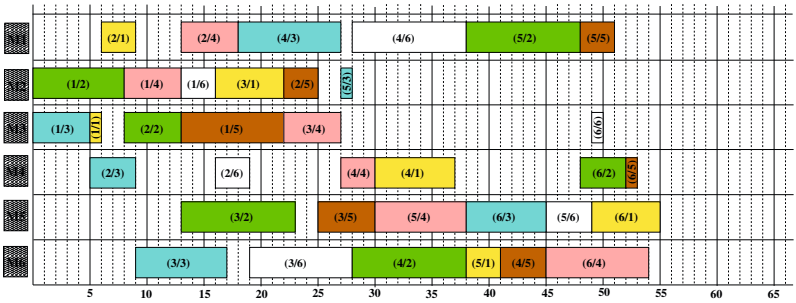


Fig. 2.4: Gantt-Chart of the optimal schedule with makespan objective.

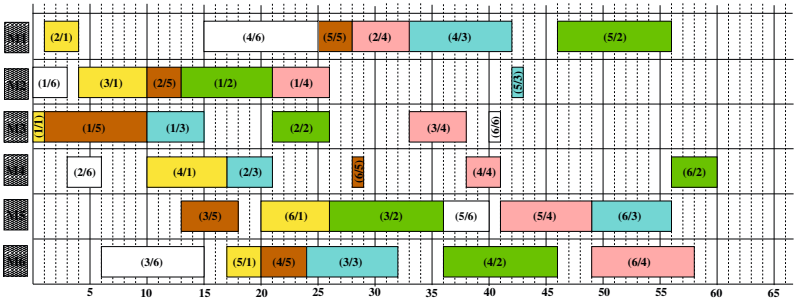


Fig. 2.5: Gantt-Chart of the optimal schedule with TWT objective.

schedule is 60 caused by the completion of operation (6/2). Comparing Fig. 2.4 and Fig. 2.5, the crucial feature between both schedules is the set of different precedence relations on the machines. 31 of 90 precedence relations have opposite values which indicate the discrepancy between these optimal schedules. Even though more than one optimal schedule can exist for a problem, this comparison already shows that a simple transfer of methods developed for the standard JSP to the JSPTWT does not appear promising.

Using ft06 ($f = 1.3$) and representing the problem structure of the JSPTWT as disjunctive graph $G = (N, A, E)$ lead to a set of 49 nodes. The arc set A includes 54 directed arcs. In the arc set E, there are 90 pairs of disjunctive arcs. Unfortunately, the total number of arcs of 234 is too large for clearly illustrating the graph G.

A feasible schedule is derived with the help of the First Come First Served rule (FCFS). The priority rule sorts the competing operations based on their arrival times: the earliest operation is processed first. If there is more than one operation with the same arrival time, then the operation with the lower job index j is prioritized. In this way, operation orders, and thus, job orders on the machines are determined as shown in Tab. 2.4.

The corresponding directed graph G' is illustrated in Fig. 2.6. Note that this figure only shows directed arcs that represent precedence relations of adjacent operations.

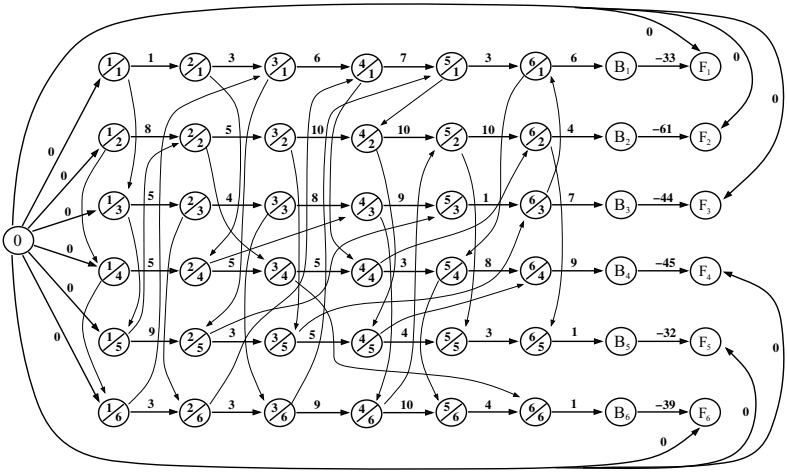


Fig. 2.6: Directed Graph G' of the ft06 ($f = 1.3$) schedule based on FCFS rule.

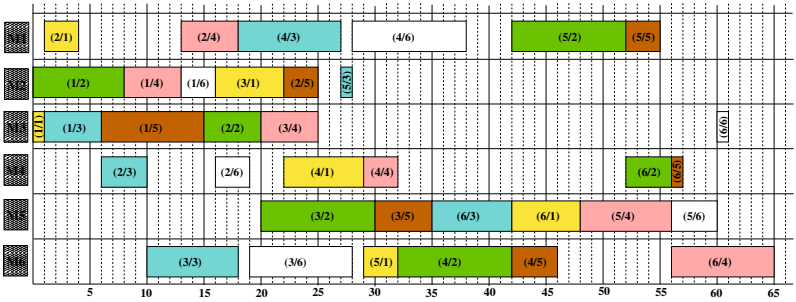


Fig. 2.7: Gantt-Chart of the ft06 ($f = 1.3$) schedule based on the FCFS rule.

Furthermore, the weights of the arcs representing the precedence relations on the machines are skipped. The Gantt-Chart of the FCFS schedule is shown in Fig. 2.7.

Mach.	Job order
1	$1 \to 4 \to 3 \to 6 \to 2 \to 5$
2	$2 \to 4 \to 6 \to 1 \to 5 \to 3$
3	$1 \to 3 \to 5 \to 2 \to 4 \to 6$
4	$3 \to 6 \to 1 \to 4 \to 2 \to 5$
5	$2 \to 5 \to 3 \to 1 \to 4 \to 6$
6	$3 \to 6 \to 1 \to 2 \to 5 \to 4$

Tab. 2.4: Job orders of the FCFS solution.

For measuring the total weighted tardiness of this schedule, all longest paths that constitute the critical tree $\mathcal{CT}$ are constructed, see Fig. 2.8. One can observe that the longest paths $\mathcal{LP}(0, F_j)$ for $j = 1, 4, 5, 6$ include the arc $B_j \to F_j$. Therefore, the corresponding jobs are tardy. Conversely, the jobs 2 and 3 are on-time. By

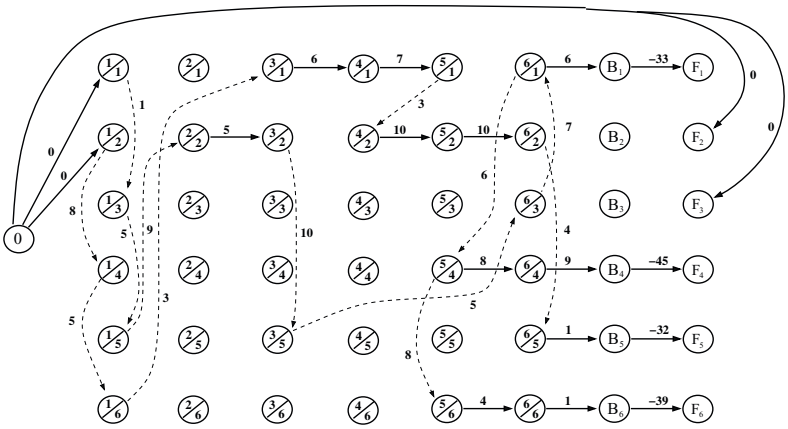


Fig. 2.8: $\mathcal{CT}(0, F)$ of the ft06 ($f = 1.3$) schedule based on FCFS rule.

Block no.	Mch.	# $\mathcal{LP}$	Operation Order
1	3	3	$(1/1) \rightarrow (1/3) \rightarrow (1/5) \rightarrow (2/2)$
2	5	3	$(3/2) \rightarrow (3/5) \rightarrow (6/3) \rightarrow (6/1)$
3	5	2	$(3/2) \rightarrow (3/5) \rightarrow (6/3) \rightarrow (6/1) \rightarrow (5/4)$
4	5	1	$(3/2) \rightarrow (3/5) \rightarrow (6/3) \rightarrow (6/1) \rightarrow (5/4) \rightarrow (5/6)$
5	2	1	$(1/2) \rightarrow (1/4) \rightarrow (1/6) \rightarrow (3/1)$
6	6	1	$(5/1) \rightarrow (4/2)$
7	4	1	$(6/2) \rightarrow (6/5)$

Tab. 2.5: Overview of the critical blocks for the FCFS solution of ft06 ($f = 1.3$).

measuring the length of the critical tree, the total weighted tardiness value of the presented schedule is $L(\mathcal{CT}(0, F)) = 172$.

In the next step, all critical arcs and blocks included in the critical tree are identified. Tab. 2.5 summarizes the resulting seven critical blocks and thirteen included critical arcs. As a result of the generated job orders, it can be observed that block no. 4 includes block no. 3 and no. 2. The list of critical arcs and blocks forms the basis for applying a neighborhood operator and starting a local search procedure to this solution.

Chapter 3

Literature Review

In comparison to other scheduling problems, particularly the minimum makespan job shop scheduling problem, the JSPTWT has only received scholarly attention in the last 15 years. For the following sections, the publications related to the JSPTWT are distinguished according to the presented solution method. Within each section, the publications are listed in chronological order.

3.1 Exact Algorithms

The Branch-and-Bound approach proposed by Singer and Pinedo [121] has been the only existing exact solution method for JSPTWT for a long time. In their work, they investigate different configurations for branching strategies and bounding. For example, they propose a branching scheme based on fixing disjunctive arcs. A branching scheme is supplemented by a bottleneck machine selection rule: based on the shifting bottleneck heuristic [111], that machine is selected where the last arc has been fixed in the branching step. Bounds are adapted from techniques developed for the minimum makespan JSP [28, 80].

The Langrangian relaxation of the Mathematical Model has been considered for the "just-in-time" JSP in [30, 61]. In this kind of problem, the JSP is solved with the objective of minimizing the sum of weighted tardiness and weighted earliness of the jobs. This objective function is transformed into the total weighted tardiness objective by setting the weights attached to the earliness to zero. The solution of the Langrangian relaxation helps to improve the lower bound to the considered optimization problem. Despite further enhancements [12], there is still an open gap for finding optimal solutions by means of optimization software.

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_3,

In 2011, Lancia et al. [82] proposed two alternative formulations for the Mathematical Model of the JSPTWT. The first formulation is based on the definition of schedule patterns, the second one relies on network flows. However, even small instances with up to 10 jobs and 5 machines could not be solved optimally within 1.800 seconds. The benefit of these two formulations is their general application to various objective functions intended for solving the JSP.

3.2 Dispatching Rules

Due to simple implementation, there are a lot of publications in the field of dispatching rules, also referred to as priority rules. Different dispatching rules have been proposed for constructing schedules so that due dates are taken into account in the generation process. A large number of publications include a computational study that incorporates the simulation of the dynamic JSPTWT, see e. g. [79]. The most famous dispatching rules with regard to due dates are the Earliest Due Date rule (EDD) [100], the Apparent Tardiness Cost rule (ATC) [139] and the Cost Over Time rule (COVERT) [29]. Further due date related dispatching rules can be found in surveys, see e. g. [63, 66]. However, no rule outperforms the other by means of generating schedules with a significant better total weighted tardiness value.

3.3 Shifting Bottleneck Heuristic

The adaption of the shifting bottleneck procedure to the JSPTWT has been presented by Pinedo and Singer in 1999 [111]. Here, the origin problem with m machines is decomposed into subproblems where the sequencing problem on one machine is solved. Already scheduled operations on other machines determine the release times and operational due dates for the schedulable operations. With the help of partial enumeration, the current subproblems are solved and the bottleneck machine is identified. The machine with the highest increase of the total weighted tardiness objective is referred to as bottleneck. The associated best sequence on this machine is fixed and the whole process is started again with updated release times and due dates. With the help of reoptimizing already scheduled machines, as well as changing the selection of the bottleneck machine in further enumeration steps, the shifting bottleneck procedure becomes more powerful.

3.4 Local Search based Algorithms and Techniques

The first heuristic concept for solving the JSPTWT was proposed by Armentano and Scrich [8]. They developed a tabu search approach that is based on a disjunctive graph model. New schedules are computed with the help of the neighborhood operator of van Laarhoven et al. [137]. However, the considered problem structure in this work does not include job weights, and thus, only a total tardiness objective is tackled.

The disjunctive graph model has also been used in the large step random walk by Kreipl [76]. This iterative local search procedure is divided into two parts: firstly, an improvement phase based on local search for finding a local optimum; secondly, a perturbation phase for generating new feasible start solutions near the current one.

In [35], de Bontridder presents a tabu search algorithm whose neighborhood search strategy benefits from the dual formulation of the problem. The dual problem corresponds to a maximum cost flow problem. With the additional collected information, the solution quality of the new schedule could be deduced. New schedules are generated by reversing one critical arc at the beginning or end of a block [101].

Zhang and Wu [151] have proposed a simulated annealing algorithm for solving the JSPTWT. The algorithm also uses information about network flows observed in the resulting dual problem. The obtained results help to identify non-improving neighborhood moves.

Tackling job shop scheduling problems with a local search involves a time-consuming computation for the objective function value of the newly generated schedules. In 2011, Mati et al. [91] presented an enhancement of Taillard's estimations for a lower bound to the total weighted tardiness objective. Their proposed iterated local search benefits from this fast determination within the neighborhood search.

Recently, Braune et al. [22] proposed further enhancements for the evaluation of the solution quality of neighboring schedules. In contrast to the work of Mati et al. [91], new schedules are generated by using the SCEI neighborhood, which represents a more extensive operator (cf. Chapter 4).

3.5 Other Heuristic Approaches

Asano and Ohta [9] have proposed a problem specific heuristic for solving the JSPTWT. This heuristic relies on a tree search procedure. In this tree, every node

represents a subproblem of the origin problem. In the subproblems, the job shop scheduling problem is solved with minimizing the maximum tardiness as objective. By fixing some arcs in the current solution, the successor node in the tree tackles the next subproblem.

Several genetic algorithms, that have been proposed in literature, are designed for solving the JSP with various objective functions (see e. g. [42, 88]). At this point, only the recent genetic algorithm of Mattfeld and Bierwirth [94] is briefly sketched. In their implementation, the encoding of schedules is based on a permutation which represents the priorities among the operations. The application of the precedence preservative crossover operator results in the generation of new solutions. With the help of an effective schedule builder, decoding the permutations produces active schedules.

Several decomposition methods have been presented for solving large-scale problem instances [120, 145, 150]. The idea is to decompose the problem into subproblems with regard to the planning horizon. Single subproblems are solved with approved techniques, for example using the shifting bottleneck heuristic [120].

3.6 Hybrid Approaches

Over the last few years more and more hybrid solution techniques have been presented for solving the JSPTWT. Applying different successful search procedures should further improve the performance of the solution procedure.

The combination of a genetic algorithm and local search was proposed by Essafi et al. [41]. After every generation of a new set of solutions, an iterated local search algorithm is started to improve the quality of the produced schedules.

Zhou et al. [154] proposed a hybrid genetic algorithm for tackling a special case of the JSPTWT. The considered problem structure does not include job's release times. The proposed procedure combines a genetic algorithm with simulation.

The JSPTWT with sequence-dependent setup times and machine availability constraints is subject of the work of Tavakkoli-Moghaddam et al. [131]. In this extension of the problem structure, setup activities on the machines have to be respected, whereas an opposite operation sequences may cause different setup cost, respectively setup times. The availability of machines is influenced by preventive maintenance activities. By setting setup times and maintenance operations to zero, the orig-

inal JSPTWT results again. The hybrid metaheuristic presented by the authors combines a simulated annealing algorithm with an electromagnetic-like mechanism. The latter is a population-based metaheuristic inspired by the behavior of electrically charged particles.

In [19], Bülbül has introduced a shifting bottleneck heuristic incorporating tabu search in the reoptimization phase. The tabu search component benefits from the application of two different neighborhood operators. Different settings of the shifting bottleneck heuristic are studied with the help of a tree search where different orders of bottleneck machines are assessed.

In the work of González et al. [57], the problem structure of the JSPTWT is also extended by the consideration of sequence-dependent setup times. For solving this problem, the authors proposed a combination of a genetic algorithm with a tabu search. The presented computational results show that this hybrid metaheuristic represents one of the best performing solution procedures.

Another hybrid solution technique has been proposed by Zhang and Wu in [152]. In their genetic algorithm several scheduling rules are implemented and the best working rule is identified and selected. Applying local search to the obtained solutions yields to further improvements. However, their work is related to the static JSPTWT, i. e. the presence of release times of the jobs is not considered.

Recently, two further hybrid metaheuristics have been proposed for solving the JSPTWT [149, 153]. Both approaches combine an evolutionary algorithm (artificial bee colony algorithm, particle swarm optimization algorithm) with a local search algorithm. In order to improve the efficiency of the algorithmic concept, both hybrid metaheuristics benefit from the results of Zhang and Wu [151] by identifying neighborhood moves which cannot yield an improvement in the local search process.

3.7 Summary

Even though there are already several publications and solution procedures for the JSPTWT, their number is still moderate. A compact overview is shown in Tab. 3.1. It can be observed that the examination of the JSPTWT has increased in recent years.

It is interesting that the reported solution techniques are still diverse. In order to get more powerful solution procedures, the combination of several metaheuristic

Authors	Year	Ref.	Contribution	Benchm.
Hoitomt et al.*	1993	[61]	Exact Algorithm	
Lin et al.	1997	[88]	Metaheuristic	
Singer & Pinedo	1998	[121]	Branch and Bound	
Pinedo & Singer	1999	[111]	Shifting Bottleneck Heuristic	
Wu et al.	1999	[145]	Decomposition Method	
Armentano & Scrich†	2000	[8]	Metaheuristic	
Kreipl	2000	[76]	Metaheuristic	
Singer	2001	[120]	Decomposition Method	
Asano & Ohta	2002	[9]	Heuristic Approach	
Chen & Luh*	2003	[30]	Exact Algorithm	
Mattfeld & Bierwirth	2004	[94]	Metaheuristic	✓
de Bontridder	2005	[35]	Metaheuristic	
Essafi et al.	2008	[41]	Hybrid Metaheuristic	✓
Zhou et al.†	2009	[154]	Hybrid Metaheuristic	
Tavakkoli-Moghaddam et al.‡	2009	[131]	Hybrid Metaheuristic	
Zhang & Wu†	2010	[150]	Decomposition Method	
Lancia et al.	2011	[82]	Mathematical Model	
Zhang & Wu†	2011	[151]	Metaheuristic	
Mati et al.	2011	[91]	Move Evaluation, Metaheuristic	✓
Bülbül	2011	[19]	Hybrid Metaheuristic	✓
González et al.‡	2012	[57]	Hybrid Metaheuristic	✓
Zhang & Wu†	2012	[152]	Hybrid Metaheuristic	
Zhang & Wu†	2013	[153]	Hybrid Metaheuristic	
Zhang et al.†	2013	[149]	Hybrid Metaheuristic	✓
Braune et al.	2013	[22]	Move Evaluation Scheme	✓

Tab. 3.1: Literature overview without listing dispatching rules (* originally considering the minimization of the unpunctuality, † considering only special cases of JSPTWT, ‡ additionally extending the problem structure).

concepts has attained particular attention in research. However, only a few publications have explained their motivation to use the selected techniques and analyzed the behavior of these techniques. There is still a lack of knowledge about the functionality of elementary components and their efficiency. Therefore, the following chapters provide new analytical and empirical results to close these gaps. Moreover, the gained results contribute to the algorithmic design of a new solution procedure for solving the JSPTWT. The presented benchmarking of this new solution procedure relies on the comparison with various computational results reported in literature (see the last column "Benchm." of Tab. 3.1). Even though each of the listed publications provides some computational results, the benchmarking focuses on solution procedures which deliver a superior solution quality.

Chapter 4

Neighborhood Definitions for the JSPTWT

The vast majority of solution methods for solving machine scheduling problems exploits the concept of neighborhood search, like tabu search, iterated local search etc. In the development of heuristics for the JSPTWT, it is observable that local search techniques are often used (cf. Section 3). One essential task in the design of local search procedures is the structure of the neighborhood and its operator. However, the different solution methods proposed in literature have been developed incorporating different neighborhood operators [19, 91]. There is little known about the impact on the solution quality of different neighborhoods. In the first part of this chapter, after introducing the general concept of the neighborhood search (Section 4.1), five known neighborhoods will be adapted to the JSPTWT (Section 4.2), and six new neighborhoods defined (Section 4.3). In Appendix A, all perturbation schemes of the neighborhoods are demonstrated on the problem instance ft06 ($f = 1.3$) and its FCFS solution (see Section 2.6). In the second part of this chapter, the proposed neighborhoods will be investigated analytically (Section 4.4) as well as empirically (Section 4.5). The chapter concludes with a short summary of the gained results.

4.1 The Basic Concept of Neighborhood Search

In general, the considered JSPTWT belongs to the class of combinatorial optimization problems. Such a problem is described by its solution set X and a cost function

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_4,

c, which assigns to each solution $x \in X$ a value $c(x)$. Instead of a complete enumeration and assessment of all solutions in X, a neighborhood search algorithm considers a small subset of solutions and determines the best solution among them. The formal definition of a neighborhood is as follows:

Definition 2 (Neighborhood [1])**.** *Let (X, c) be an instance of a combinatorial optimization problem. A neighborhood function is a mapping $NB : X \to 2^X$, which defines for each solution $x \in X$ a set $NB(x) \subseteq X$ of solutions that are in some sense close to x. The set $NB(x)$ is the neighborhood of solution x and each $x' \in NB(x)$ is a neighbor of x. We shall assume that $x \in NB(x)$ for all $x \in X$.*

In other words, the neighborhood defines a subset of solutions that is similar to one feasible solution x. The neighboring solutions are generated by using a specific perturbation scheme applied to the original solution x. The cost function c helps to define the following important characteristic for the solutions in $NB(x)$:

Definition 3 (Locally Optimal Solution [1])**.** *Let (X, c) be an instance of a combinatorial optimization problem and let NB be a neighborhood function. A solution $x \in X$ is locally optimal (minimal) with respect to NB if*

$$c(x) \leq c(x'), \qquad \forall x' \in NB(x).$$

In general, the solutions of the neighborhood differ in their cost function value $c(x)$. The aim of a neighborhood search algorithm is to move from a current feasible solution to a new and hopefully better feasible solution. The local search heuristic repeats this search from the new obtained solution. The heuristic ends in a solution that is locally optimal.

There is no common approach to design the perturbation scheme, more specifically, select an appropriate neighborhood. Papadimitrioiu and Steiglitz have stated that this "choice is usually guided by intuition"[4]. For this reason, the motivation of this chapter is to analyze several different neighborhoods to determine whether or not they are suitable and efficient for solving the JSPTWT.

[4]cf. Papadimitrioiu and Steiglitz: Combinatorial Optimization: Algorithms and Complexity [107], page 455

Abbrev.	Name	Reference
CT	Critical Transpose	van Laarhoven et al. [137]
CET	Critical End Transpose	Nowicki, Smutnicki [101]
CET+2MT	Critical End Transpose + 2-Machine Transpose	Matsuo et al. [92]
CE3P	Critical End 3-Permutation	Dell'Amico, Trubian [36]
SCEI	Single Critical End Insert	Dell'Amico, Trubian [36]
CSR	Critical Sequence Reverse	Zhang, Wu [151]

Tab. 4.1: List of existing neighborhood operators.

4.2 Existing Neighborhoods

Various neighborhoods have been proposed for the minimum makespan JSP. They originally base on critical blocks observed in the disjunctive graph representation of the minimum makespan JSP. These neighborhoods are directly adapted to the JSPTWT by applying the respective perturbation schemes to all critical blocks observed in the critical tree. For simplicity, the general sequence of operations within the critical block is denoted by $u_1, u_2, \dots u_n$. A list of the existing operators is presented in Tab. 4.1.

CT. The *critical transpose* neighborhood, introduced by van Laarhoven [136], is the first known approach to generate a neighboring solution. The operator performs the reversal of exactly one critical arc. This structure is also known as swap [91] or interchange [92] of two adjacent operations. This perturbation can only reduce the length of a longest path provided the critical arc is positioned at the beginning or end of a critical block. Otherwise, when the reversed arc is inside the critical block, it does not change the starting times of the succeeding operations within that critical block. Thus, the starting time of the last operation on this path still depends on the scheduling of this block and cannot be reduced in the new schedule [92].

CET. Since the reversal of an arc inside a critical block cannot yield an improvement, Nowicki and Smutnicki [101] have proposed a reduced CT neighborhood called the *critical end transpose* neighborhood. In the perturbation scheme, one critical arc at the beginning or end of a critical block is reversed. Fig. 4.1 shows an example with a critical block consisting of three critical arcs, respectively four operations u_1

to u_4. The CET operator reverses either $u_1 \rightarrow u_2$ or $u_3 \rightarrow u_4$. In this example, a new schedule is generated by reversing the last arc, depicted as gray colored arc in Fig. 4.1.

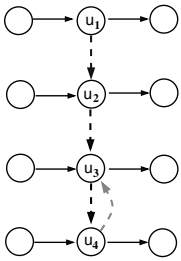


Fig. 4.1: CET move.

Nowicki and Smutnicki [101] have proposed further reductions of the CET neighborhood. They discard the critical arc $u_1 \rightarrow u_2$ at the beginning of the first critical block of the longest path if the source 0 is the only preceding node of u_1 and this critical block consists of more than one critical arc, i. e. containing more than the two operations u_1, u_2. The reversal of this first critical arc cannot yield an improvement since the starting time of the third operation within this block cannot be changed. Based on a similar argument, Nowicki and Smutnicki have detected a second possibility for reducing the set of neighboring solutions: the reversal of the critical arc at the end of the last critical block, i. e. the sink is the only succeeding node, and this block consists of more than one critical arc. The reversal of this last critical arc cannot reduce the longest path, either. However, these proposals of neighborhood reductions cannot be applied to the JSPTWT, since the reversal of the first arc can improve the solution caused by the impact of the release times r_j. Furthermore, reversing the last critical arc could affect two longest paths, and thus, the tardiness value of two jobs.

CET, however, defines an efficient subset of the CT neighborhood so that the CT neighborhood is omitted in the following study.

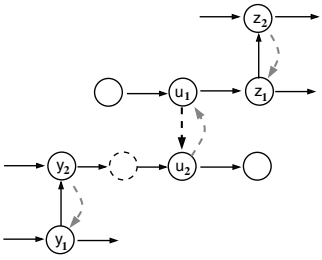


Fig. 4.2: CET+2MT move.

CET+2MT. Matsuo et al. [92] have enhanced the idea of the CET neighborhood by reversing additional arcs related to predecessors or successors in the machine sequences (see Fig. 4.2). Based on the reversal of the critical arc $u_1 \rightarrow u_2$, two further machine arcs might be reversed simultaneously, provided the following conditions hold. The machine arc $z_1 \rightarrow z_2$ is reversed (with z_1 being the direct subsequent operation of the machine sequence related to u_1) if and only if (i) z_1 is started before the completion of u_2 and (ii) z_1 is completed directly before the start of z_2. The machine arc $y_1 \rightarrow y_2$ (with y_2 being one preceding operation of the machine sequence related to u_2) is reversed if and only if (i) the direct job

predecessor of u_2 is not completed before the start of u_1, (ii) y_2 is started immediately after the completion of y_1, and (iii) all operations between y_2 and u_2 are processed without idle time. The resulting operator is called *critical end transpose + 2-machine transpose* operator.

These accompanying arc reversals attempt to break up blockings in the longest path which result from the reversal of the critical arc $u_1 \rightarrow u_2$. Note that the two additional reversed arcs do not have to be critical arcs. It is only necessary that the above described conditions are fulfilled.

CE3P. The reversal of one critical arc can be viewed as a 2-permutation of two adjacent operations. Dell'Amico and Trubian have generalized this idea defining the *critical end 3-permutation* operator. In addition to the two operations at the beginning or the end of a critical block, either their directly preceding or directly subsequent operation on the machine is taken into account, even this operation does not belong to a longest path. A neighboring schedule is generated by changing the order of the three consecutive operations by means of a 3-permutation. However, only 3 of the 6 existing 3-permutations are taken into consideration, namely those which reverse the two operations at the beginning (or at the end) of the critical block. Fig. 4.3 shows an example: in addition to the first two operations based on the critical arc $u_1 \rightarrow u_2$, their direct succeeding operation u_3 is used for generating a neighboring schedule. The CE3P operator creates the following 3-permutations displayed as arc sequences: $u_2 \rightarrow u_1 \rightarrow u_3$, $u_2 \rightarrow u_3 \rightarrow u_1$ or $u_3 \rightarrow u_2 \rightarrow u_1$. The last perturbation is shown in Fig. 4.3 with the help of the gray colored arcs.

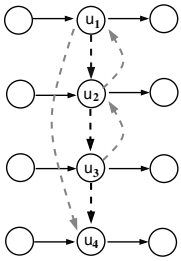


Fig. 4.3: CE3P move.

Based on the two selection possibilities of the considered critical arc $u_1 \rightarrow u_2$, permutations with the subsequent operation (u_3 in the above example) as well as preceding operation (it would be u_0 in the above example) are possible candidates. However, there are only five 3-permutations in total, since the exclusive reversal of $u_1 \rightarrow u_2$ is delivered by focusing on u_0, as well as u_3.

SCEI. A further neighborhood operator proposed by Dell'Amico and Trubian [36] attempts to shift an operation within a critical block. The new position of this opera-

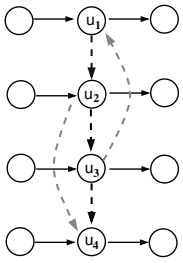


Fig. 4.4: SCEI move.

tion in the block is the most distanced insert position that still yields a feasible schedule. This operator is called the *single critical end insert* operator. In order to ensure that a length of the longest path is affected by a perturbation, operations inside a critical block are only shifted to the first or the last position in the block. This is because shifting an inner operation inside a block does not change the starting time of the last operation, and thus, does not reduce the corresponding longest path. In the example of Fig. 4.4, the operation u_3 is shifted to the front of the critical block, resulting in the new arc sequence $u_3 \rightarrow u_1 \rightarrow u_2 \rightarrow u_4$.

Based on the same argumentation like in CET, two approaches for the reduction of the SCEI neighborhood are known. The shifting of the first operation u_1 in the first critical block to a position inside this block can be discarded if the source 0 is the only preceding node of u_1. The same situation is given by the last operation in the last critical block. However, these proposals for a reduction of the neighborhood are only applicable for the minimum makespan JSP. For the adaptation to the JSPTWT, which incorporates release times and bases on multiple longest paths, these perturbations can also significantly affect the solution quality.

A variant of SCEI is defined by Balas and Vazacopoulos [11]. They propose to reduce the neighborhood in advance by only allowing perturbations that rely on special path conditions. The consequence is that insert positions do not have to be the most distanced position. The operator performs shift moves to insert positions for which feasibility can be guaranteed in advance. This variant of SCEI is faster, but can fail in finding the most improving SCEI move. Another variant of the SCEI neighborhood is introduced by Zhang et al. [147]. The defined operator restricts the insertion of the first or last operation to the second or second to last position respectively. This perturbation move corresponds to a reversal of one critical arc, just like CET. However, the following study concentrates on the pure exclude-insert operator defined by Dell'Amico and Trubian [36].

CSR. The idea to reverse a connected sequence of operations has been introduced for the minimum makespan JSP by Steinhöfel et al. [124], and is already used for the JSPTWT by Zhang and Wu [151]. Obviously, this perturbation guarantees the generation of feasible schedules for sequences of only two operations, which corres-

Abbrev.	Name	derived from
ECET	Extended Critical End Transpose	CET
ICT	Iterative Critical Transpose	CET
CE4P	Critical End 4-Permutation	CE3P
DOCEI	Double Outwards Critical End Insert	SCEI
DICEI	Double Inwards Critical End Insert	SCEI
BCEI+2MT	Backwards Critical End Insert + 2-Machine Transpose	SCEI, CET+2MT

Tab. 4.2: List of new neighborhood operators.

ponds to an ordinary CT perturbation [148]. Applying the operator to longer sequences requires a feasibility test to be executed in each step, i. e. , after a single arc has been reversed. In the example of Fig. 4.5, four operations of the critical block $u_1, \dots u_4$ are considered as the sequence to be reversed. The new sequence $u_4 \to u_3 \to u_2 \to u_1$ establishes the reversal of all critical arcs between u_1 and u_4. The described operator is called *critical sequence reverse* operator.

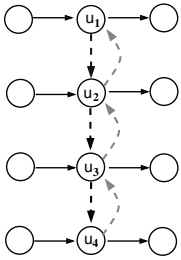


Fig. 4.5: CSR move.

4.3 New Neighborhoods

In this section, six new neighborhoods are introduced, which extend and supplement the above described neighborhood concepts. Tab. 4.2 presents an overview of these neighborhoods.

ECET. The transpose-based neighborhood operator CET is concentrated only on one critical arc, either the arc at the beginning or at the end of a critical block. The *extended critical end transpose* neighborhood enlarges this perturbation scheme. Another perturbation for generating a neighboring schedule is to reverse these two critical arcs simultaneously: the reversal of the arc at the beginning of the block as well as the reversal of the arc at the end of a critical block. A necessary condition is that both arcs have no operation in common, i. e. the block consists of at least three arcs. For this reason, the single reversal of a critical arc is the only perturbation that can be performed in blocks

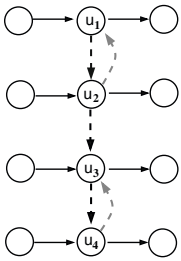


Fig. 4.6: ECET move.

with, at most, two critical arcs, i. e. the perturbation scheme is identical to the CET neighborhood. The example in Fig. 4.6 shows the additional perturbation move, reversing both critical arcs $u_1 \to u_2$ and $u_3 \to u_4$ in one step.

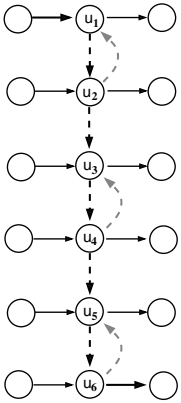


Fig. 4.7: ICT move.

ICT. Large critical blocks essentially determine the length of a longest path. The idea of the *iterative critical transpose* neighborhood is to break up long critical blocks by reversing every second arc, starting with the first arc in a block. This perturbation scheme always results in a feasible schedule (as shown in Section 4.4.1). Fig. 4.7 shows an ICT move for a block of length 5, reversing the first, the third, and the fifth arc.

Note that there can only be one neighboring schedule derived from one critical block. After the determination of the considered critical block, the predefined reversals are performed without exception or flexibility.

CE4P. Like CE3P, this perturbation scheme considers either the first or the last arc of a critical block. But different to CE3P, both the direct preceding and the direct succeeding operations of the critical arc are involved in the perturbation which leads to the *critical end 4-permutation* neighborhood. Consider four such consecutive operations $u_1, \ldots u_4$, as shown in Fig. 4.8. To ensure that at least the order of the two inner operations u_2 and u_3 is inverted (which represents the initial critical arc), the operator can generate a total of twelve different 4-permutations. The depicted 4-permutation in the figure corresponds to the new sequence of arcs $u_3 \to u_2 \to u_4 \to u_1$.

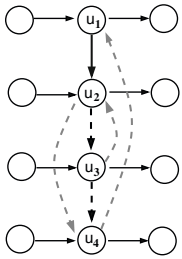


Fig. 4.8: CE4P move.

Note that the only permutations taken into consideration are those which yield a feasible schedule. However, regardless of the necessary feasibility, the perturbation scheme of CE4P extends the permutation-based neighborhood CE3P in a ratio 12:5 since all neighboring schedules of CE3P are also generated by CE4P.

DOCEI. The *double outwards critical end insert* neighborhood selects two consecutive operations in a critical block and inserts the former operation at the end

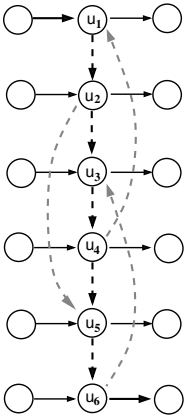


Fig. 4.9: DOCEI move.

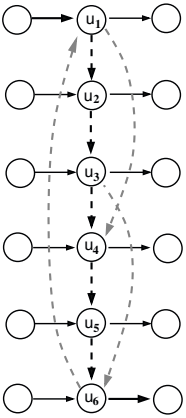


Fig. 4.10: DICEI move.

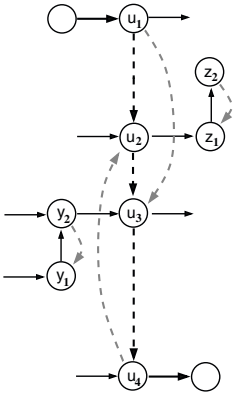


Fig. 4.11: BCEI+2MT move.

of the critical block and the latter operation at the beginning of the critical block. Note that this perturbation scheme extends the concept of the SCEI operator by performing a second exclude-insert perturbation move. Infeasible schedules are omitted. The example in Fig. 4.9 shows a block of length 5 where the consecutive operations u_3 and u_4 are first excluded and then reinserted at the end and the beginning of the critical block. The resulting operation sequence is $u_4 \rightarrow u_1 \rightarrow u_2 \rightarrow u_5 \rightarrow u_6 \rightarrow u_3$.

Note that the selection of the first two operations (respectively last two operations) in the block leads to a neighboring schedule that is identical to a SCEI move of the first operation to the end of the block (respectively last operation to the front of the block).

DICEI. The perturbation by the *double inwards critical end insert* operator works like DOCEI with the exception that the selected consecutive operations are not shifted outwards to the border of the critical block, but the border operations are shifted inwards between the selected operations. Fig. 4.10 shows the same example considered for DOCEI before, now leading to the new operation sequence $u_2 \rightarrow u_3 \rightarrow u_6 \rightarrow u_1 \rightarrow u_4 \rightarrow u_5$.

BCEI+2MT. The *backwards critical end insert + 2-machine transpose* neighborhood operator, first introduced in [78], combines a SCEI move with a two machine transpose move (like this in CET+2MT). In other words, the perturbation scheme

joins the shift of an operation with two additional arc reversals, provided a feasible schedule can be generated in this way. An example is illustrated in Fig. 4.11, where operation u_2 is shifted to the end of the critical block in SCEI manner. Furthermore, to break up possible blockings in the resulting schedule, two additional arcs ($z_1 \rightarrow z_2$ and $y_1 \rightarrow y_2$) are reversed. These arcs belong to the job successor of the shifted operation u_2 and the job predecessor of its succeeding operation u_3 in this block.

Preliminary computational experiments have shown that shifting an operation to the beginning of the block rarely leads to improvements with the exception of the second operation. The latter shift move corresponds to a CET move. This observation is explained by the resulting starting times of the operations after the perturbation move. Shifting an operation to the front of a block leads to an earlier starting time of one operation while the processing of operations inside the block is delayed. As a consequence, the completion of more than one job is negatively effected. Therefore, the operator is designed by always shifting operations to the end of the block, or by shifting the second operation to the front of the block. Moreover, the conditions for the two accompanying arc reversals are the same as those in CET+2MT, but with the exception that y_2 has to be the direct predecessor of u_3.

Based on the above definitions, the following inclusions of the neighborhood operators are apparent:

$$\begin{aligned} &\text{CET } (\subseteq \text{ CT }) \subseteq \text{CSR} \\ &\text{CET} \subseteq \text{CE3P} \subseteq \text{CE4P} \\ &\text{CET} \subseteq \text{ECET} \\ &\text{CET} \subseteq \text{SCEI} \end{aligned}$$

A classification of the proposed neighborhoods is provided by the underlying principle of generating new schedules using transpose-based, insertion-based and

	Transpose-based	Insertion-based	Sequence-based
on single machine	CET ECET ICT	SCEI DOCEI DICEI	CE3P CE4P CSR
on multiple machine	CET+2MT	BCEI+2MT	——

Tab. 4.3: Classification of neighborhoods based on perturbation schemes.

sequence-based perturbations. Moreover, changing the order of operations in a schedule on a single or on multiple machines can be distinguished. Tab. 4.3 shows the corresponding scheme. Note that the operators CE3P and CE4P appear in the class of sequence-based operators because their generation scheme is based on sequences of three and four operations, respectively.

4.4 Characteristics of the proposed Neighborhoods

The following features represent characteristics used to describe a neighborhood definition more precisely [93]:

(1) Correlation: A neighboring solution should be very similar to the origin solution x, i. e. they only differ by a few attributes.

(2) Feasibility: Neighboring solutions derived from a feasible solution should always be feasible again. If this feature is not immediately guaranteed, it has to be checked after generating the neighboring solution.

(3) Connectivity: With the help of a connected neighborhood the search process, which is started from an arbitrarily solution, is potentially able to attain a globally optimal solution within a finite number of performed perturbations.

(4) Size: The neighborhood NB creates a set of neighboring solutions that, on the one hand should be small and manageable, but on the other hand, large enough to provide an improving solution.

(5) Improvement: The probability of improving solutions within the neighborhood set should be as high as possible. This feature determines the success of the local search procedure.

The first feature reflects the core of the defined perturbation scheme of the neighborhoods (see the previous two sections). Features (2) and (3) can be investigated analytically; therefore, the existing knowledge about feasibility guarantees and the connectivity properties are reviewed. Some new analytical results are presented in this section. The feature (4) is analyzed according to the estimate of an upper bound of neighboring solutions. However, the size of a neighborhood can be smaller, as presented in this analysis. Therefore, this feature as well as feature (5) are investigated empirically (see Section 4.5 for the corresponding computational experiments and results).

4.4.1 Feasibility Property

A schedule is called feasible if it fulfills all conditions of the problem structure of the JSPTWT, e. g. the technological machine sequence of every job. From the set of neighborhood operators, those are preferred, which automatically guarantee schedule feasibility. Within the search procedure, there would be no need to check all required conditions of the problem.

The most basic perturbation of a schedule is performed by CET which simply reverses one critical arc. This always results in a feasible solution [137]. Lemma 1 extends this result towards the reversals of adjacent non-critical operations, provided these operations are immediately started on the same machine without idle time. Lemma 1 is used to prove Lemma 2 and Theorem 1. Theorem 1 proves the feasibility guarantee for CET+2MT. Based on Lemma 2, feasibility guarantees are derived for ECET and ICT.

Lemma 1. *Let G' be a directed graph representing a feasible schedule. Furthermore, let v and w be two not necessarily critical, successive operations on a machine that are processed immediately after each other, i. e. without intermediate idle time on the machine. Then, no other path can exist from v to w in G', and the reversing of arc $v \rightarrow w$ always produces a feasible solution.*

Proof. Assume there is an additional path leading from v to w. This path P contains at least two more operations, denoted by $o_1, \dots o_r$ ($r \geq 2$). Due to the represented precedence relations on this path $P = v \rightarrow o_1 \rightarrow \cdots \rightarrow o_r \rightarrow w$, the processing of w starts at the earliest after the completion of o_r. Since $v \neq o_r$, $p_{o_r} > 0$ and the processing of o_r starts after the completion of v, too, the assumption that there is no intermediate idle time between the processing of v and w on the machine is no longer fulfilled. Since no further path exists between v and w, reversing arc $v \rightarrow w$ creates no cycle. □

Lemma 2. *Let G' be a directed graph representing a feasible schedule. Furthermore, let $v_1 \rightarrow w_1$ and $v_2 \rightarrow w_2$ be two not necessarily critical arcs in G' that represent precedence relations on machines and have no operations in common. Additionally, there is no intermediate idle time between the processing of v_1 and w_1 as well as v_2 and w_2 on their machine, and a path P consisting of directed arcs leads from w_1 to v_2. By reversing both arcs simultaneously, a feasible solution is produced.*

Proof. Suppose that reversing both arcs $v_1 \to w_1$ and $v_2 \to w_2$ creates a cycle $\mathcal{C}$. This cycle has to contain both reversed arcs, because otherwise, the exclusive reversal of one arc would produce a cycle already precluded by Lemma 1. Let $\mathcal{C} = w_1 \to v_1 \to P_1 \to w_2 \to v_2 \to P_2 \to w_1$ with P_1 and P_2 being paths connecting v_1 and w_2 as well as v_2 and w_1. Formally, P_1 and P_2 represent subsets of operations. In case that v_1 and w_2 as well as v_2 and w_1 are connected by only one arc, these subsets are empty, i. e. each path, P_1 and P_2, consists of only one arc. However, P_1 and P_2 also exist in G', and thus, the arc sequence $w_1 \to P \to v_2 \to P_2 \to w_1$ must form a cycle in G'. This contradicts the feasibility of G'. □

Based on Lemma 2, it is clear that the feasibility guarantee holds true for the neighborhood operator ECET. The argumentation of Lemma 2 can also be applied to a chain of consecutive arcs to be reversed in a new schedule. In this way, the feasibility guarantee for neighborhood operator ICT is derived.

Theorem 1. *Let G' be a directed graph representing a feasible schedule. Then, neighborhood operator CET+2MT always produces feasible solutions.*

Proof. W. l. o. g. consider a situation where the operator generates a new solution by reversing all three relevant arcs (see Fig. 4.2). For simplicity, the arcs have been named $z_1 \to z_2 = d$, arc $u_1 \to u_2 = e$ and arc $y_1 \to y_2 = f$. These are the arcs taken into account in the original schedule, whereby d', e' and f' are the corresponding reversed arcs in the new schedule.

Assume that the newly generated schedule includes a cycle $\mathcal{C}$. In the following, four cases are considered as to how this cycle could look like.

Case 1: Cycle $\mathcal{C}$ contains none of the reversed arcs. However, $\mathcal{C}$ must already exist in G' which contradicts the feasibility of the origin schedule.

Case 2: Cycle $\mathcal{C}$ contains exactly one of the reversed arcs. Since, for all arcs it holds, there is no immediate idle time in the processing of the corresponding adjacent operations, a contradiction to Lemma 1 arises.

Case 3: Cycle $\mathcal{C}$ contains exactly two of the reversed arcs.

(a) $\mathcal{C}$ contains d' and e' (see Fig. 4.12, left side). The cycle consists of two sub-paths, P_1 and P_2, which connects the two reversed arcs: $\mathcal{C} = e' \to P_1 \to d' \to P_2 \to e'$. Both sub-paths already exist in G', since they are not affected by one of the three reversals. But then, the arc sequence

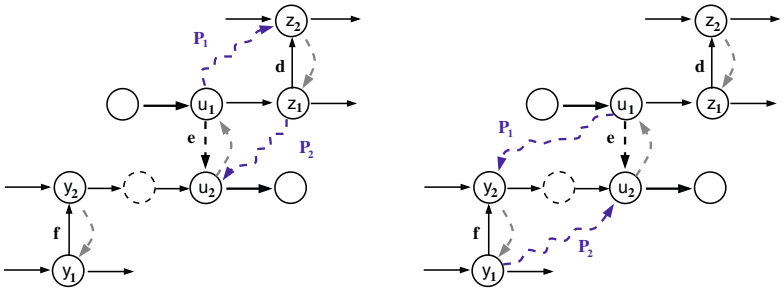


Fig. 4.12: Course of a potential cycle produced by CET+2MT: Left: $d', e' \in \mathcal{C}$ (Case 3a), Right: $e', f' \in \mathcal{C}$ (Case 3b).

$u_1 \to z_1 \to P_2$ is a longer path from u_1 to u_2 in G' as the arc e. Thus, e cannot be a critical arc in G'. $\lightning$

(b) $\mathcal{C}$ contains e' and f' (see Fig. 4.12, right side). Again, the cycle consists of two sub-paths, P_1 and P_2, which connects the two reversed arcs: $\mathcal{C} = e' \to P_1 \to f' \to P_2 \to e'$. Both sub-paths, P_1 and P_2, already exist in G', since they are not affected by one of the three reversals. But then, the sequence $u_1 \to P_1 \to y_2 \to \cdots \to u_2$ is a longer path from u_1 to u_2 in G' as the arc e. Thus, e cannot be a critical arc in G'. $\lightning$

(c) $\mathcal{C}$ contains d' and f'. The argumentation in this case is identical. The course of the cycle would imply that e is no critical arc in G'.

Case 4: Cycle $\mathcal{C}$ contains all three reversed arcs. These arcs are connected by three sub-paths P_1, P_2 and P_3, that complete the cycle $\mathcal{C}$. Two possibilities exist regarding what the course of $\mathcal{C}$ looks like:

(a) $\mathcal{C} = f' \to P_1 \to e' \to P_2 \to d' \to P_3 \to f'$ (see Fig. 4.13, left side): all sub-paths P_1, P_2, and P_3 already exist in G', since they are not affected by any of the three reversals. But then, the arc sequence $u_1 \to z_1 \to P_3 \to y_2 \to \cdots \to u_2$ is a longer path from u_1 to u_2 in G' as the arc e. Thus, e cannot be a critical arc in G'. $\lightning$

(b) $\mathcal{C} = f' \to P_1 \to d' \to P_2 \to e' \to P_3 \to f'$ (see Fig. 4.13, right side): all sub-paths P_1, P_2, and P_3 already exist in G', since they are not affected by one of the three reversals. But then, the sequence $u_1 \to z_1 \to P_2$ is a longer path from u_1 to u_2 in G' as the arc e. Thus, e cannot be a critical arc in G'. $\lightning$

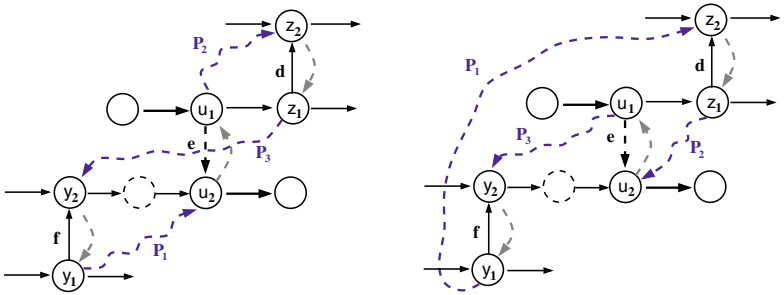


Fig. 4.13: Course of a potential cycle produced by CET+2MT: Left: $d', e', f' \in \mathcal{C}$ (Case 4a), Right: $d', e', f' \in \mathcal{C}$ (Case 4b).

With the above contradictions, the feasibility guarantee of schedules generated by the neighborhood operator CET+2MT has been proven. □

Tab. 4.5 summarizes the above findings on the feasibility guarantee. It can be concluded from the table that the feasibility guarantee only holds for the transpose-based operators CET, ECET, ICT and CET+2MT. For the rest of the neighborhood operators, the feasibility of the generated schedule has to be checked either within the generation scheme (like for SCEI), or after the schedule has been built (like for CSR). For doing this, a feasibility test is required. Such a test detects possible cycles through iteratively determining the starting times of all operations due to their predecessors. The run-time complexity of this test is $\mathcal{O}(nm)$, see [128].

4.4.2 Connectivity Property

The connectivity property of a neighborhood affects the behavior of a local search procedure to reach every schedule by successively replacing the current solution with a neighboring solution.

Definition 4 (Connectivity [25])**.** *A neighborhood NB is called connected if it is possible to transform each solution x into each other solution $\bar{x}$ by a finite number of moves in NB, i. e. if a sequence of solutions $x = x_0, x_1, x_2, \ldots x_k = \bar{x}$ exists with $x_t \in NB(x_{t-1})$ for $t = 1, \ldots k$.*

With respect to optimization, a slightly weaker definition is sufficient to assess the capabilities of a neighborhood.

Definition 5 (Opt-Connectivity [25])**.** *A neighborhood NB is called opt-connected if it is possible to transform each solution x into an optimal solution x^* by a finite number of moves in NB, i. e. if a sequence of solutions $x = x_0, x_1, x_2, \dots x_k = x^*$ exists with $x_t \in NB(x_{t-1})$ for $t = 1, \dots k$.*

From current literature, it is known that the neighborhood CT, which performs the reversal of one critical arc, is opt-connected for the minimum makespan JSP [137] as well as for the JSPTWT [41, 75]. Since CSR includes CT, CSR is also opt-connected for both optimization problems. Furthermore, Dell'Amico and Trubian [36] have shown that SCEI is opt-connected for the minimum makespan JSP. Their proof can be transferred directly to the JSPTWT.

In the following, a counterexample illustrates that some of the considered neighborhoods are not opt-connected.

Theorem 2. *The neighborhoods CET, ECET, ICT, CET+2MT, CE3P and CE4P are not opt-connected.*

Proof. Consider the problem instance in Tab. 4.4 consisting of four jobs and two machines. Note that the jobs have a common due date and follow the same technological machine sequence (actually, it is a flow shop problem). An optimal schedule x^* is obtained from Johnson's rule, shown on the left side of Fig. 4.14. Note that multiple optimal schedules to this problem instance exist in which job 4 is always the one processed at last on machine 1.

Given the schedule x_0, shown on the right side of Fig. 4.14, it is clear that it cannot be transferred into x^* by means of CET moves. CET merely allows reversing the two leftmost and the two rightmost operations of the critical block $(1/1) \to (1/4) \to (1/3) \to (1/2)$. Therefore, operation $(1/4)$ will never take the last position in the block, which means that job 4 cannot be processed as the last job on

	o_{1j}		o_{2j}				
Job j	μ_{1j}	p_{1j}	μ_{2j}	p_{2j}	w_j	r_j	d_j
1	1	4	2	2	1	0	16
2	1	4	2	3	1	0	16
3	1	4	2	3	1	0	16
4	1	4	2	1	1	0	16

Tab. 4.4: Data to the example 4x2 instance.

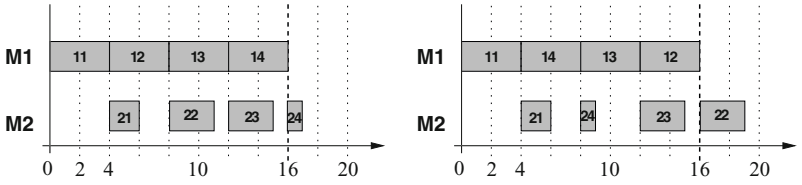


Fig. 4.14: Gantt-Chart of the optimal schedule x^* (left) and the initial schedule x_0 (right).

machine 1. This, however, is a necessary condition for schedule optimality in this problem instance.

Taking a look at the schedule perturbation schemes performed by ECET, ICT and CET+2MT, it becomes apparent that the above example disproves the opt-connectivity property for these neighborhood definitions as well. Each of these neighborhood operators cannot transfer the presented schedule x_0 into another schedule with operation (1/4) as the last one on machine 1.

For the CE3P and CE4P neighborhoods, a similar counterexample is obtained by simply adding two further jobs with processing time $p_{1j} = 4$ and $p_{2j} = 2$ to the above flow shop instance and increasing the common due date to 24. Sequencing these additional jobs between job 4 and job 3 on each machine, the initial schedule x_0 contains one critical block on machine 1. It consists of $(1/1) \rightarrow (1/4) \rightarrow (1/5) \rightarrow (1/6) \rightarrow (1/3) \rightarrow (1/2)$. Both neighborhoods produce the same set of neighboring schedules, either based on the permutation of the first three or the last three operations of this block. However, in all generated schedules in the complete neighborhood search process, job 4 cannot enter the last position (not even a position in the latter half) of machine 1. But, the optimal schedule of this extended problem instance has job 4 on the last position of machine 1 again. □

Even though this single counterexample is sufficient, it is possible to create further counterexamples by scaling the job data or including further jobs. The counterexample presented above is not applicable for the remaining neighborhood operators DOCEI, DICEI and BCEI+2MT. Therefore, the opt-connectivity property is indicated as "open" for these operators. The analytical results regarding the feasibility guarantee and connectivity property of neighborhoods are summarized in Tab. 4.5. It is interesting to note that there is no neighborhood definition which is opt-connected and provides a feasibility guarantee at the same time.

	opt-connected	not opt-connected	open
Feasibility Guarantee	—	CET ECET ICT CET+2MT	—
No Feasibility Guarantee	SCEI CSR	CE3P CE4P	BCEI+2MT DICEI DOCEI

Tab. 4.5: Classification of neighborhoods based on feasibility and connectivity.

4.4.3 Estimate of the Size of the Neighborhoods

The size of a neighborhood refers to the number of neighboring schedules that can be generated by applying the operator to one solution. However, only feasible schedules are accepted in the search process so that infeasible schedules are discarded immediately. Moreover, only an approximation of the size by means of a maximum number of neighboring schedules or upper bound can be indicated. The actual number of neighbors can be smaller, since some of the perturbations can lead to the same neighboring schedule. For example, critical blocks can overlap each other if two longest paths merge at a certain node. In this case, both blocks contain at least one critical arc, namely the first critical arc (cf. the example of the FCFS schedule of ft06 ($f = 1.3$) in Section 2.6). Apparently, the application of the CET operator to this critical arc produces the same solution in both blocks.

For a characterization of the size of the neighborhoods, the structure of the critical tree $\mathcal{CT}$ has to be specified. Therefore, let τ denote the number of critical blocks, and σ_l the length of block $l = 1, \ldots \tau$ according to the number of included critical arcs. For simplicity, the critical blocks are given in ascending order of their length. Three subsets are distinguished based on parameter σ_l:

- τ_1 - number of blocks with $\sigma_l = 1$
- τ_2 - number of blocks with $\sigma_l = 2$
- τ_3 - number of blocks with $\sigma_l \geq 3$

Based on this specification of critical blocks, it is possible to count the number of neighbors $\mathcal{Z}$ according to the perturbation scheme of the operator. For example, in the CET neighborhood, every critical block with $\sigma_l \geq 2$ is tackled twice: the reversal of the first or the last arc is performed for generating a new schedule. For

a blocks with $\sigma_l = 1$, there is only one critical arc, and thus, the operator can only generate one new schedule. The number of neighbors is estimated by:

$$\mathcal{Z}_{CET} = \tau_1 + 2\tau_2 + 2\tau_3$$

In the ECET neighborhood, there is an additional perturbation move available for all critical blocks with $\sigma_l \geq 3$: the simultaneous reversal of the first and last arc. The number of neighbors increases slightly to:

$$\mathcal{Z}_{ECET} = \tau_1 + 2\tau_2 + 3\tau_3$$

With the ICT operator, each critical block results in exactly one neighboring solution by reversing every second arc inside the block:

$$\mathcal{Z}_{ICT} = \tau_1 + \tau_2 + \tau_3 = \tau$$

The CET+2MT neighborhood has the same upper bound as CET, since the two additional arc reversals are always performed if the corresponding conditions are fulfilled:

$$\mathcal{Z}_{CET+2MT} = \tau_1 + 2\tau_2 + 2\tau_3$$

The CE3P neighborhood enhances CET. As already stated in the description of the operator, every critical arc at either the beginning or the end of a block leads to, at most, five different 3-permutations, and thus, five neighboring schedules. However, two of the ten 3-permutations in a block of length $\sigma_l = 2$ are identical, since the reversal of the two arcs is obtained by focusing CE3P on the first arc (in combination with the last operation) and by focusing it on the second arc of the block (in combination with the first operation):

$$\mathcal{Z}_{CE3P} = 5\tau_1 + 9\tau_2 + 10\tau_3$$

The size of the CE4P neighborhood can be approximated by the same analysis as with CE3P. The application of the CE4P operator leads to, at most, twelve 4-permutations based on one chosen critical arc. Furthermore, there are two identical neighboring solutions in a block with length $\sigma_l = 2$. The reversal of the two critical arcs is a 4-permutation that is derived from the first critical arc as well as from the second (last) critical arc:

$$\mathcal{Z}_{CE4P} = 12\tau_1 + 23\tau_2 + 24\tau_3$$

In the SCEI neighborhood, neighbors are generated by excluding one operation and

inserting it at a new position. Every inner operation of a block can be shifted in two directions. Border operations are only shifted in one direction. The size of SCEI can be derived from the number of operations, and thus, from the number of critical arcs:

$$\mathcal{Z}_{SCEI} = \tau_1 + \sum_{l=\tau_1+1}^{\tau} 2\sigma_l$$

Both neighborhoods DOCEI and DICEI are also based on the set of critical arcs, since either two adjacent operations in a block are shifted (DOCEI) or the border operations are repositioned between these adjacent operations (DICEI):

$$\mathcal{Z}_{DOCEI} = \sum_{l=1}^{\tau} \sigma_l \quad \text{and} \quad \mathcal{Z}_{DICEI} = \sum_{l=1}^{\tau} \sigma_l$$

In the BCEI+2MT neighborhood, the essential perturbation is the shift move of an operation to the end of a block. The number of available shift moves in a block corresponds to the number of critical arcs, since the last operation is already on the last position of its block. An alternative to the backward shift move is the shift of the second operation to the front of the block, which is additionally considered in blocks of length $\sigma_l \geq 2$. Like in CET+2MT, the two additional arc reversals are always performed if all conditions for these moves are fulfilled:

$$\mathcal{Z}_{BCEI+2MT} = \sum_{l=1}^{\tau} \sigma_l + \tau_2 + \tau_3$$

The CSR operator performs a reversal of a connected operation sequence in a block. Based on a critical block of arbitrarily length, σ_l neighbors are derived by inverting 2 adjacent operations, $\sigma_l - 1$ neighbors are obtained by inverting 3 adjacent operations, ... 1 neighbor by inverting all operations. Using the Gaussian totals formula, the number of neighbors is estimated by:

$$\mathcal{Z}_{CSR} = \sum_{l=1}^{\tau} \frac{\sigma_l \cdot (\sigma_l + 1)}{2}$$

The analytical results show that the number of neighboring schedules can either be dependent on the number of critical blocks or the number of critical arcs, or on both citeria simultaneously. Based on the dependency, the following proportions are determined:

$$\mathcal{Z}_{ICT} \leq \mathcal{Z}_{CET} = \mathcal{Z}_{CET+2MT} \leq \mathcal{Z}_{ECET} \leq \mathcal{Z}_{CE3P} \leq \mathcal{Z}_{CE4P}$$
$$\mathcal{Z}_{DOCEI} = \mathcal{Z}_{DICEI} \leq \mathcal{Z}_{BCEI+2MT} \leq \mathcal{Z}_{SCEI}$$

Proportions between permutation-based and insertion-based neighborhoods cannot be stated in general. On the one hand, the critical tree of a schedule could contain a high number of blocks with length $\sigma_l = 1$. Then, the size of permutation-based neighborhoods is larger than those of insertion-based neighborhoods. On the other hand, the critical tree of a schedule may contain only a few blocks, all consisting of a high number of critical arcs. In this situation, the insertion-based neighborhoods produce many more neighboring schedules than permutation-based neighborhoods.

4.5 Performance Analysis

In this section, a comprehensive computational study focuses on the performance of the proposed neighborhood operators. First, the used test suite is described briefly and then the outcome of local search algorithms employing a single operator as well as a combination of different operators is presented and analyzed.

4.5.1 Test Suite

The computational tests are done using 53 benchmark instances from Beasley's OR-Library [13]. The test suite contains 18 instances each with 10 jobs and 10 machines (abz05-abz06, la16-la20, ft10 and orb01-orb10). The suite is supplemented by 35 instances with 10 to 30 jobs and 5 to 15 machines (la01-la15 and la21-la40), where the job-machine-ratio varies from 1:1, 3:2, 2:1, 3:1 to 4:1. Together, eight different classes of instances are given in the test suite where the size $n \mathrm{x} m$ is constant in each class.

Due dates and job weights are computed for all instances according to the procedure of [121], presented in Section 7.1.1. In this study, the due date factor $f = 1.3$ is exclusively used. Essafi et al. [41] have reported the objective function value of the so far best known solutions to all 53 test instances. Note that none of these instances have been solved with a total weighted tardiness of zero so far. Less tight due date factors of $f = 1.5$ and $f = 1.6$, which are also considered in [41, 121], are not considered within this section. In this way, at least one job of a feasible schedule is tardy, which enables a reliable assessment of the neighborhood operators.

4.5.2 Local Search with a Single Neighborhood Operator

The eleven neighborhood operators proposed in Section 4.2 and 4.3 are assessed in the following way. First, a sufficiently large sequence $x_i = (x_i^1, x_i^2, \dots)$ of random start solutions is computed for each of the 53 benchmark instances $i = 1, \dots 53$. The neighborhood operators $j = 1, \dots 11$ are incorporated in a steepest descent algorithm. Given a start solution, this algorithm computes the objective function value of all neighboring solutions with respect to the used neighborhood operator. Provided that the best neighboring solution shows a lower total weighted tardiness than the start solution, the best neighboring solution replaces the start solution and the search restarts from the new solution. Otherwise, if no improved solution is found in the neighborhood of the start solution, the steepest descent algorithm terminates and a new steepest descent algorithm is started with the next random start solution taken from the sequence x_i. The entire procedure stops with one of two alternative termination criteria. With the first termination criterion, the steepest descent algorithm is repeated for neighborhood $j = 1, \dots 11$ and instance $i = 1, \dots 53$ until a prescribed number of local optima is generated. With the second termination criterion, the algorithm stops after a prescribed number of neighboring schedules is evaluated for every operator and every problem instance.

Using the described schedule generation procedure, two experiments are performed on our test suite. In both experiments, the same sequence x_i of start solutions is used for the problem instances by each of the neighborhood operators. In the first experiment, the performance quality of the neighborhood operators is assessed without considering how many evaluations a steepest descent walk takes. This is achieved by terminating the procedure after a prescribed number of steepest descent walks (termination criterion 1). In the second experiment, the performance quality of the neighborhood operators is assessed for a certain level of computational effort. This is reached by terminating the procedure after a prescribed number of schedule evaluations have been carried out (termination criterion 2). The number of evaluations is chosen with regard to the size of an instance, i. e. the number of jobs and machines involved. Note that the number of produced local optima usually differs for the competing neighborhood operators in this second experiment. This kind of comparison allows assessing neighborhoods under a common limitation of the computing capacity.

The different settings in the experiments enable to measure two kinds of per-

formance qualities of the neighborhoods. The fixed number of steepest descent walks provides the evaluation of the total performance quality, i. e. assessing the full potential of a neighborhood and its associated set of neighboring solutions in the complete local search algorithm. On the other hand, the fixed number of the available schedule evaluations leads to the assessment of the absolute performance quality, i. e. the potential of a neighborhood operator with respect to the achieved effort. This approach explains the pure potential of the neighborhood operator when it creates one neighboring solution.

Let $TWT(BFS_i^j)$ denote the objective function value of the best solution found by neighborhood operator j for problem instance i. Furthermore, let $TWT(BKS_i)$ denote the objective function value of the best known solution for problem instance i. In order to assess the solution quality achieved by a neighborhood operator, the average gap to the best known solutions is computed:

$$\text{Gap}(j) = \frac{1}{53}\sum_{i=1}^{53}\frac{TWT(BFS_i^j) - TWT(BKS_i)}{TWT(BKS_i)} \quad [\text{in } \%] \quad \text{for } j = 1,\ldots 11$$

Experiment 1: For every neighborhood operator and every problem instance, a total of 100 locally optimal schedules is generated. Hence, over the entire test suite, a total of 5.300 locally optimal schedules is produced by every neighborhood operator. With this common bound of generated local optima, the total performance quality of each neighborhood operator is measured. Based on the best schedule found for a problem instance, an average gap is computed and reported in the left part of Tab. 4.6. The results indicate that the operators perform very differently on the test suite. Five of the operators (SCEI, CET+2MT, CE4P, BCEI+2MT and CSR)

	Experiment 1				**Experiment 2**			
Operator	Gap	# Eval	# Imp	Rank	Gap	# LOpt	# Imp	Rank
SCEI	73.69	26.85	17.57	1	71.14	6575	12.14	4
CET+2MT	73.98	3.07	17.13	2	60.17	27169	14.35	1
CE4P	75.34	31.71	15.81	3	84.09	2331	12.63	11
BCEI+2MT	76.70	11.42	13.06	4	69.26	12509	9.55	3
CSR	78.44	75.50	16.05	5	80.90	4781	10.41	9
CE3P	81.83	14.40	12.29	6	78.53	6023	10.44	7
CET	81.84	2.68	15.10	7	67.03	28947	13.43	2
ECET	83.32	4.70	14.50	8	72.45	18657	12.40	6
DOCEI	96.13	3.37	4.57	9	81.18	28266	4.42	10
DICEI	96.70	3.10	4.30	10	80.63	28918	4.30	8
ICT	100.02	0.53	3.39	11	72.29	108700	3.79	5

Tab. 4.6: Results of the computational study using a single neighborhood operator.

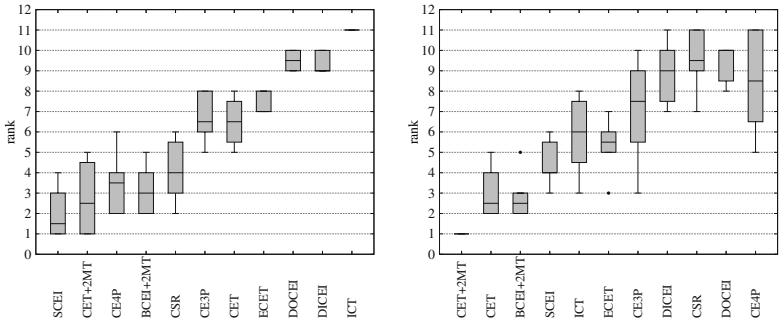


Fig. 4.15: Ranks of neighborhood operators according to the eight instance classes in Experiment 1 (left) and Experiment 2 (right).

show a gap between 73% and 78%. This group is followed by three operators (CE3P, CET and ECET) with a gap of approximately 82%. The remaining three operators (DOCEI, DICEI and ICT) form a group which performs with a gap above 95%. The rank achieved by the operators is also reported in the table. As one might expect, the rank correlates quite closely with the average number of improving steps (# Imp) that the operators have realized during the steepest descent walks. The coefficient of correlation is -0.86. This is also illustrated by the dominant downward slope of the curve belonging to Experiment 1 in the left diagram of Fig. 4.16. It means that, the more improving steps an operator executes, the better the produced solution quality is in turn. Considering the total number of schedule evaluations # Eval (in million) made by the operators, there is no clear correlation to the rank obtained. For example, operator CE3P, which belongs to the medium performing group of operators, consumes more than four times the amount of evaluations needed by the second best performing operator CET+2MT. The most schedule evaluations ($75{,}5\times10^6$) are made by CSR, which takes rank 5 among all operators.

The total performance quality of a neighborhood operator varies significantly between instances of different classes. However, the ranks derived from the gaps based on an instance class show that the operators can be arranged quite consistently. In the left part of Fig. 4.15, the ranks obtained for the eight instance classes are illustrated by using box plots[5]. The top and and the bottom of the boxes is mostly stretched over two or three ranks. The ICT operator always delivers the worst performance for every instance class so that the corresponding box vanishes.

[5]The ends of the whiskers represent the lowest and highest rank within the 1.5-times interquartile range.

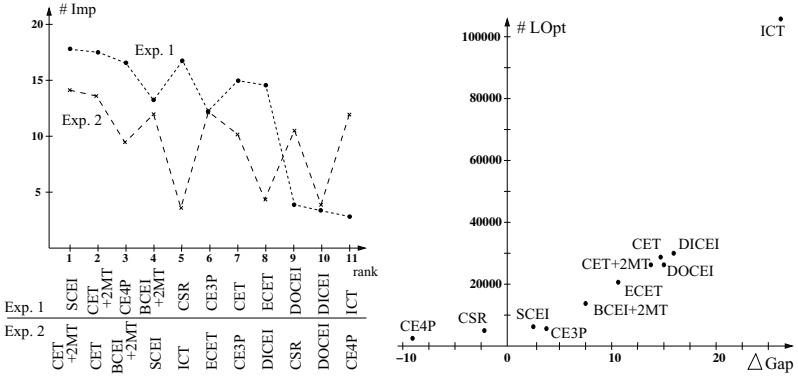


Fig. 4.16: Correlation between rank and # Imp (left); correlation between # LOpt and ΔGap (right).

Experiment 2: This experiment is driven by the observation that in Experiment 1, the number of schedule evaluations varies strongly for the operators. Therefore, in Experiment 2, a common bound on the number of schedule evaluations is introduced for the neighborhood operators. The idea of the experiment is to measure the absolute performance quality of each neighborhood operator with regard to a certain computational effort. Since the size of the instances of the test suite varies, a size-dependent limit of $10 \cdot n^2 \cdot m^2$ schedule evaluations is imposed. This means that the total number of evaluations (# Eval) is constant for every test instance in this experiment. E. g., for an instance with $n = 10$ jobs and $m = 10$ machines, a total of 100.000 schedules are evaluated by each of the eleven operators. A total of 12.86 million schedule evaluations is made by each operator on the entire test suite. Like before, the assessment and comparison of an operator are based on the average gap, the associated rank, and the average number of improving steps. Furthermore, the total number of local optima (# LOpt) generated by the operators for the entire test suite is counted. The corresponding results are reported in the right part of Tab. 4.6. Like in Experiment 1, the operators perform very differently for the test suite. Again, the gained results enable to sort them into three groups. The best performing operators (CET+2MT, CET, BCEI+2MT) show a gap between 60% and 69%. This group is followed by four operators (SCEI, ICT, ECET and CE3P) with a gap between 71% and 78%. The group of weaker operators contains four operators (DICEI, CSR, DOCEI, CE4P) with a gap above 80%. As shown in the left diagram

of Fig. 4.16 by the curve belonging to Experiment 2, the correlation between the rank and the average number of improving steps realized by the operators is much weaker than in the previous experiment. The column # LOpt indicates that the operators exploit the available capacity for schedule evaluations very differently. While, for example, operator SCEI produces about 6.600 locally optimal schedules, operator ICT comes along with more than tenfold that amount, yet both operators differ by only one rank.

In the right part of Fig. 4.15, the ranks obtained for the eight instance classes are illustrated again by using box plots. The achieved ranks of the operators vary stronger between the instance classes as in Experiment 1. In particular, the performance of the permutation-based operators CE3P and CE4P show a high variance between instances of different size. In contrast, the CET+2MT operator always achieves rank 1 in each instance class.

Comparing the outcome of both experiments, it is striking that nearly all operators have reached a better gap in Experiment 2. With the exception of CSR and CE4P, the operators have also generated more local optima in this experiment than in Experiment 1. Remember that a fixed limit of 5.300 locally optimal schedules is prescribed in Experiment 1. While CSR and CE4P have performed less steepest descent walks in Experiment 2 than in Experiment 1, the other operators have actually produced more locally optimal schedules here (see column # LOpt). The larger number of restarts performed by the majority of operators in Experiment 2 explains the better gap observed in Experiment 2. It can also be observed from Tab. 4.6 and the right diagram of Fig. 4.16 that the operators benefit quite consistently from an increased number of restarts. For example, operators CET+2MT, CET, DOCEI and DICEI make about 27.000 and 29.000 restarts in Experiment 2, which respectively leads to an improvement (ΔGap) of 14% to 16% compared to Experiment 1.

The solution quality achieved by the operators in the two experiments is very different. By dividing the operators into weak, medium and strong performing operators, different groupings are obtained in the two experiments. Fig. 4.17 indicates that merely CET+2MT and BCEI+2MT perform strongly in both experiments. In both experiments, two operators perform on a medium level and two on a weak level. The rest of the operators perform differently between the two experiments.

Nevertheless, even the best performing operators produce an average gap above

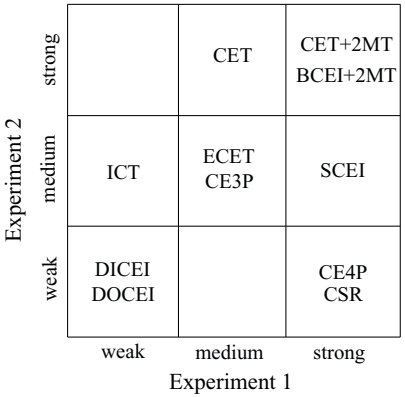


Fig. 4.17: Grouping of operator performance in Experiments 1 and 2.

60%, meaning that the total weighted tardiness of the best found solutions deviate significantly from the total weighted tardiness of the best known solution. In other words, a local search with a single operator generates a relatively weak solution quality. Unfortunately, this is not alleviated by simply increasing the run-time of the local search. For example, doubling the number of schedule evaluations for the best performing operator in Experiment 2 yields a 7.0% reduction of the gap, and tripling yields an 8.8% reduction. Apparently, multiplying the computational effort does not pay off.

4.5.3 Local Search with Pairs of Neighborhood Operators

Taking for granted that different neighborhood operators possess individual capabilities, it might be profitable to combine them in a local search procedure. In this section, pairs of operators which complement one another in a favorable fashion are studied. For this purpose, the above experiments are repeated with the modification of jointly applying two neighborhood operators in the steepest descent search algorithm instead of just a single one. The neighborhood operators are used alternately until neither of them can improve the current schedule anymore. In this way, the algorithm ends up with a schedule which is locally optimal with respect to two different neighborhood definitions. The alternation of the neighborhoods shall provide a diversity in the local search process. Therefore, as long as both neighborhood operators produce at least one improving solution, the steepest descent algorithm

switches between the neighborhoods after a descent step, i. e. if the last improvement has been achieved by operator 1, operator 2 is given the chance to make the next improvement.

In the experiments, only combinations of operators are considered which appear to be compatible. Two neighborhood operators are called incompatible if one operator includes the other (like CSR includes CET) or if both operators belong to the same perturbation class (like CET and ICT, see Tab. 4.3). In total, 42 combinations of the eleven neighborhood operators are investigated. The results produced by these combinations of operators are shown in Tab. 4.7. Here, the operator that performed better in Experiment 1 is named as operator 1. Note that Experiments 3 and 4 are designed just like Experiments 1 and 2, and that all performance indicators are computed in the same way.

Experiment 3: The gap observed for the considered neighborhood operator combinations vary in a range from 62% to 94%. While the worst operator combinations are composed from the two worst performing single operators (DICEI and ICT), the best operator combination is formed by CET+2MT and CE4P which achieve rank 2 and 3 individually. The second best neighborhood operator combination applies the two best single operators SCEI and CET+2MT. In the next two positions, quite similar combinations are met, whereby only BCEI+2MT takes the role of CET+2MT. It is also worth mentioning that CET+2MT and BCEI+2MT are always involved in the top 17 operator combinations. Apparently, perturbing schedules on multiple machines (+2MT) is advantageous in combinations with all other operators considered. However, applying the two operators together yields merely rank 10. It can be also observed from Tab. 4.7 that combining operators leads to a significant increase in the number of improving steps (# Imp) made until a local optimum is reached compared to a local search employing just a single operator. The operator combination that made the largest number of improving steps achieved the second best gap. The best combination (CET+2MT and CE4P) improves the gap of the best single operator SCEI by over 10%. The following 22 combinations also achieve gaps better than SCEI alone; however, not all tested neighborhood operator combinations work well together. Some combinations obtain a gap which improves the gap achieved by the better of the two operators only marginally. In some cases, combinations perform even worse (like SCEI and ICT).

		Experiment 3				**Experiment 4**			
Operator 1	Operator 2	Gap	# Eval	# Imp	Rk	Gap	# LOpt	# Imp	Rk
CET+2MT	CE4P	62.64	28.45	20.45	1	70.03	2965	17.55	21
SCEI	CET+2MT	63.43	19.90	22.12	2	61.65	7464	15.87	3
SCEI	BCEI+2MT	64.31	28.97	21.84	3	62.84	5652	15.49	7
CE4P	BCEI+2MT	64.49	39.81	21.13	4	69.48	2427	17.33	19
CET+2MT	CE3P	64.65	13.82	20.23	5	62.13	6084	17.52	5
CET+2MT	CSR	65.15	54.93	20.86	6	65.14	5684	13.63	10
BCEI+2MT	CE3P	65.40	21.89	20.12	7	72.04	4696	16.22	24
BCEI+2MT	CSR	65.47	64.15	21.30	8	66.64	4639	13.82	13
BCEI+2MT	CET	67.33	12.16	21.32	9	61.78	10617	15.84	4
CET+2MT	BCEI+2MT	67.65	11.76	20.14	10	60.43	11213	14.52	1
CET+2MT	CET	68.85	3.70	20.06	11	60.88	19506	17.58	2
BCEI+2MT	ECET	68.93	13.48	20.79	12	63.68	9268	15.72	8
CET+2MT	ECET	69.58	5.83	19.64	13	62.46	15548	16.79	6
CET+2MT	DOCEI	70.33	13.23	19.42	14	66.66	11694	14.24	14
BCEI+2MT	DICEI	70.56	17.47	16.70	15	65.42	8556	12.69	11
BCEI+2MT	DOCEI	70.71	17.18	16.83	16	68.04	8548	12.77	17
CET+2MT	DICEI	71.27	13.40	19.14	17	67.57	11750	14.07	16
SCEI	CE3P	71.74	26.84	18.80	18	78.26	4369	14.39	34
CET+2MT	ICT	71.84	4.22	19.11	19	63.69	20788	16.13	9
SCEI	CE4P	72.55	39.28	18.23	20	77.49	2711	14.29	32
SCEI	CSR	72.70	60.29	18.46	21	73.95	4830	11.77	28
BCEI+2MT	ICT	72.93	11.02	16.93	22	67.53	11912	13.05	15
CE4P	DOCEI	73.57	36.42	16.73	23	80.95	2634	14.16	41
SCEI	ECET	73.76	17.62	18.09	24	69.12	8456	12.96	18
CE4P	ICT	73.94	30.04	16.31	25	80.82	2922	14.19	40
CSR	DICEI	74.20	67.05	17.22	26	76.50	4976	11.48	30
CE3P	ECET	74.32	5.11	16.28	27	73.17	6298	14.69	25
SCEI	ICT	74.50	22.91	19.12	28	69.83	7813	13.21	20
CE4P	DICEI	74.56	36.39	16.50	29	81.26	2636	13.97	42
CSR	DOCEI	74.76	68.05	17.54	30	77.77	4976	11.54	33
CE4P	ECET	75.52	23.98	15.75	31	78.50	3397	14.06	36
CSR	ICT	76.74	60.12	17.08	32	79.11	6027	11.13	39
CSR	ECET	78.55	42.45	16.11	33	75.58	6753	10.83	29
CET	DOCEI	78.72	10.68	16.53	34	71.14	12920	12.54	22
ECET	DOCEI	79.12	11.92	16.10	35	73.21	11041	12.44	26
ECET	DICEI	79.16	11.83	15.68	36	73.30	11072	12.27	27
CET	DICEI	79.60	10.51	16.05	37	71.54	12962	12.37	23
CE3P	DOCEI	80.44	16.92	13.76	38	78.35	6556	10.81	35
CE3P	DICEI	80.61	16.86	13.42	39	66.33	6573	10.65	12
CE3P	ICT	81.09	11.28	13.24	40	76.74	7826	11.22	31
DOCEI	ICT	93.55	2.78	5.52	41	79.09	30370	5.50	38
DICEI	ICT	93.82	2.55	5.18	42	78.85	31258	5.31	37

Tab. 4.7: Results of pairs of neighborhood operators applied jointly.

Experiment 4: The outcome of this experiment is quiet similar to the outcome of Experiment 3. The gap achieved by the operator combinations is, in most cases, a little better than in Experiment 3. If all operator combinations make the same number of schedule evaluations, the combination of CET+2MT with BCEI+2MT

performs best with a gap of about 60%. Six of the top 10 combinations from Experiment 3 are among the top 10 combinations in Experiment 4. However, no operator combination of Experiment 4 achieved a better gap than the best performing single operator CET+2MT in Experiment 2. Apparently, the number of operator combinations which fail in this experiment is much higher than in Experiment 3. This is explained by the observation that the weaker operator often consumes the vast majority of schedule evaluations and prevents the better operator from unfolding its pure potential.

To summarize, combining two adequate neighborhood operators has a clear impact on the gained total performance quality. However, the achievable quality is not yet satisfying. Moreover, the absolute performance quality decreases even slightly. The best achieved gap over all experiments so far is shown by the single application of CET+2MT in Experiment 2.

4.5.4 Local Search with all Neighborhood Operators

It is expected that the repetition of the experiments with combinations of more than two neighborhood operators will confirm the results. Whereas the total performance quality would get a little better, the absolute performance quality will hardly improve further on. In this section, the limitations of combining neighborhood operators are investigated. For this purpose, the experiments are repeated by using all eleven operators in the steepest descent search algorithm together. The obtained solutions are local optima with regard to each of the neighborhoods. Tab. 4.8 presents the outcomes of the corresponding Experiments 5 and 6. Note that the alternation concept of the neighborhood is adjusted by using the neighborhoods in ascending order of the achieved ranks in Experiments 1 and 2 respectively. In this way, a rotation of the neighborhoods in use is produced. If the current neighborhood produces at least one improving solution, the best improvement is selected and the local search is restarted with the next neighborhood operator. The rotation starts with the best neighborhood, followed by the second best and so on. After the search has used the operator with rank 11, the rotation returns to the operator with rank 1. For example, the rotation in Experiment 5 follows the order SCEI, CET+2MT, CE4P, ... ICT, SCEI, ... This handling concept is motivated again by the expected increase of diversity in the neighborhood search.

Experiment 5: The total performance quality is further improved with the combined use of all neighborhoods. However, the additional gain according to the best pair of neighborhoods of Experiment 3 is only 3.14%. Obviously, the benefit of using another neighborhood in the steepest descent algorithm vanishes quickly. Compared to the best single operator and the best pair of neighborhood operators, the computational effort expressed by the number of schedule evaluations (# Eval) nearly doubles, but the average number of improving steps does not grow in the same manner.

The neighborhoods progress in a different way than in the previous experiments. The percentage distribution of the improvements the neighborhoods carried out is shown by the bold line in the left plot of Fig. 4.18. The strong correlation to the ranking produced in Experiment 1 is caused by the used rotation scheme. Furthermore, it is interesting to aware which of the neighborhoods contributes the last improvement because this neighborhood can be viewed as the strongest one in the considered setting. The percentage distribution of the last performed improvement is shown by the dashed line in Fig. 4.18 (left). The strong correlation between the neighborhood ranking of Experiment 1 and the neighborhood ranking induced by the last performed improvement indicates that the implemented order of the neighborhoods affects the performance of the steepest descent algorithm.

Experiment 6: The deterioration of the absolute performance quality observed in this experiment is not striking. Weak performing neighborhood operators consume a significant portion of schedule evaluations without contributing to the performance quality in the same ratio. For this reason, the gap achieved in this experiment is only 65.88%. The analysis of the percentage distribution of the made improvements, as well as the last made improvement to the neighborhoods in this experiment, is shown in Fig. 4.18 (right). Again, there is a strong correlation to the used order of the neighborhoods derived from the provided ranks from Experiment 2.

To summarize, applying all eleven neighborhoods together produces the best

	Experiment 5			**Experiment 6**		
	Gap	# Eval	# Imp	Gap	# LOpt	# Imp
All Operators	59.50	50.79	26.03	65.88	2774	19.91

Tab. 4.8: Results of all neighborhood operators applied jointly.

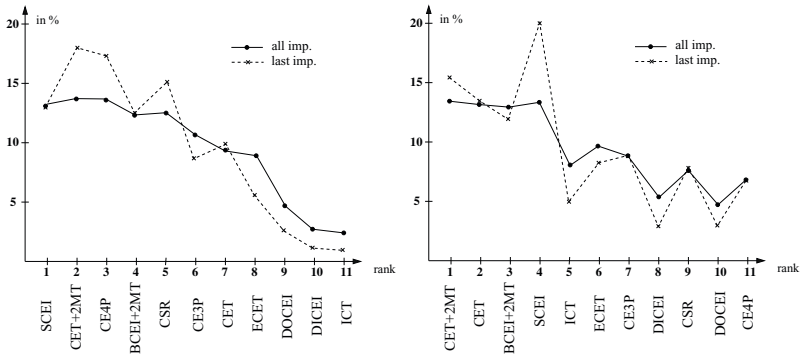


Fig. 4.18: Percentage distribution of improvements/last improvements to the neighborhoods in Exp. 5 (left) and Exp. 6 (right).

average gap of 59.50% in all experiments. However, regarding the enormous computational effort which is demonstrated by the number of made schedule evaluations, Experiment 5 has shown that the search quality of the neighborhoods is still unsatisfying.

4.6 Summary

In this chapter, five existing as well as six new neighborhoods are considered for the JSPTWT. For all eleven neighborhoods, theoretical properties and their empirical performance qualities are analyzed. The presented results are summarized in Tab. 4.9.

Generally, desirable is to have a neighborhood at hand which generates no infeasible schedule and provides a connected neighborhood at the same time. As stated before, none of the known neighborhoods for the JSPTWT fulfills both criteria, see the left part of Tab. 4.9. Here, "+" denotes the fulfillment of the property, "−" the nonfulfillment and "◎" shows that the specific classification is open.

The size of a neighborhood is a criterion for which it is hard to derive strong recommendations. In general, the size "should be within useful bounds"[6]. If the neighborhood is too small, the search process may get stuck in the early phases, thus resulting in local optima of poor quality. On the other hand, checking large

[6]cf. D. C. Mattfeld: Evolutionary Search and the Job Shop: investigations on genetic algorithms for production [93], page 29.

	Theoretic Properties		**Performance Quality**	
Operator	Feasibility Guarantee	Connectivity Property	Final Rank in Single App.	Final Rank in Paired App.
CET+2MT	+	−	1	1
SCEI	−	+	2	4
BCEI+2MT	−	◎	3	2
CET	+	−	4	3
ECET	+	−	5	6
CSR	−	+	6	7
CE4P	−	−	7	9
CE3P	−	−	8	8
ICT	+	−	9	11
DOCEI	−	◎	10	10
DICEI	−	◎	11	5

Tab. 4.9: Compact results of neighborhood operator.

neighborhoods requires a high computational effort. The presented analytical investigation of the sizes by means of the maximum number of neighboring schedules (cf. Section 4.4.3) is confirmed by the results of the performance analysis in Section 4.5.2. The proportions stated for the neighborhoods (e. g. $\mathcal{Z}_{ICT} \leq \mathcal{Z}_{CET}$) are confirmed by the number of schedule evaluations (# Eval) made in Experiment 1.

The crucial feature of a neighborhood is its search quality and its ability to guide the search to good local optima. The results in the Experiments 1-4 have shown that none of the proposed neighborhoods outperforms all other ones. The detailed results for the 53 problem instances confirm that each of the eleven neighborhood operators can potentially yield the best result. Furthermore, Experiments 5 and 6 have demonstrated that the search quality of neighborhoods, even when all operators are applied together, is very limited.

The performance results are summarized by means of final ranks that the neighborhoods achieved in Experiments 1-4. These final ranks are based on the mean value of the gaps obtained in Experiments 1 and 2 (single application of the neighborhoods), as well as in Experiments 3 and 4 (paired application of the neighborhoods). The final ranks are given in the right part of Tab. 4.9. As an overall result, some neighborhoods (in particular CET+2MT) are recommendable for implementing them in a local search based metaheuristic, while other neighborhoods are not (like DOCEI).

For solving a specific JSPTWT instance, it is worth the effort to test different

neighborhoods, possibly within different combinations. Furthermore, varying the order of neighborhoods can also have an impact on the solution quality. Therefore, it is interesting to know how to exploit all these findings within the algorithmic desgin of a local search based metaheuristic.

Chapter 5

Neighbor Evaluation Procedures in Local Search based Algorithms for solving the JSPTWT

Local search algorithms perform iteration processes in which a neighborhood search is repetitively executed. The computational effort of the neighborhood search should be as low as possible and as high as needed in order to make continuously improvements over a long period of time. The generation and assessment of neighboring solutions are major steps in this process and consume most of the computation time. The generation of new schedules depends on the complexity of the perturbation scheme of the used neighborhood operator (see Chapter 4). The assessment of the generated solutions, i. e. the determination of the resulting total weighted tardiness value, can be performed exactly or approximately.

The exact assessment of the quality of a solution is obtained by scheduling all operations, i. e. the start times s_{ij} of the operations are determined one by one. This result is obtained e. g. by applying Bellman's labeling algorithm [14], which has a time complexity of $\mathcal{O}(n \cdot m)$. On the other hand, an approximation procedure can be used for the estimation of the solution quality of a neighboring schedule. The general idea of such an approximation procedure is to compute a lower bound value for the neighboring schedule and to compare this value with the objective function value of the original solution. More precisely, the approximation procedure assesses the solution quality on the basis of the performed perturbation move, and thus, only focusing on the changes in the schedule made by the neighborhood operator. The

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_5,

resulting lower bound value is always feasible with regard to the exact objective function value of the neighboring solution, i. e. since the considered problem is a minimization problem, the estimation produces a lower bound value less than or equal to the real total weighted tardiness of the neighboring solution. As a result, the comparison of this value with the total weighted tardiness of the original solution enables to evaluate if the neighboring solution can be discarded immediately. If the lower bound value exceeds this TWT value, the newly generated schedule is no improving solution. Otherwise, the solution quality has to be checked exactly in order to verify whether this new schedule is really improving or not. The latter means that the computed lower bound value has underestimated the real solution quality. However, these approximation procedures, also called lower bound procedures, are profitable and lead to a fast decision about the potential of a neighboring schedule.

In this chapter, an alternative approach for the assessment of neighboring solutions is introduced which is called Heads Updating procedure. Two important features, a fast procedure and an exact assessment of the neighboring schedules, are tackled simultaneously. In this way, an additional computation of the real TWT value of the neighboring schedule is not necessary. Moreover, the procedure is applicable for all kinds of neighborhood operators.

The chapter is organized as follows: In the first section, the principles of lower bound procedures are introduced. Furthermore, the basic concept originally introduced for the minimum makespan JSP is briefly outlined. In Section 5.2, the lower bound procedure of Mati et al. [91] is described, which is applicable in a local search using the CET neighborhood. Afterwards, Section 5.3 sketches the lower bound procedure of Braune et al. [22]. This procedure has been developed for the evaluation of schedules generated by the SCEI operator. In Section 5.4, the Heads Updating procedure for the assessment of neighboring solutions is presented and theoretical characteristics are derived. Finally, Section 5.5 provides a brief performance comparison of the Heads Updating procedure with the other two presented lower bound procedures. A brief summary of the results concludes this chapter.

5.1 Basic Principles

The concept of labeling is widely used in network theory and computer science [138]. The introduction of labels helps to define attributes for nodes and edges/arcs. One

example of a well known label is the degree of node u which indicates the number of edges incident with node u. Furthermore, the processing time attached to an arc represents an arc label.

For the evaluation of the solution quality by a lower bound procedure, two additional labels named head and tail are introduced for each operation. Taillard defined these labels as well as proposed the general interrelationships in [129]. The head label corresponds to the earliest starting time of an operation. The tail label describes the follow-up time needed to finish one specific job:

Definition 6 (Head [129])**.** *The length of the longest path leading from the source 0 to operation v is equal to the earliest starting time of operation v. The value is called head of v and is denoted by $h(v)$.*

Definition 7 (Tail(j))**.** *The longest path leading from operation v to the finishing node F_j indicates the follow-up time between v and F_j. The length of this path is called tail(j) of operation v and is denoted by $q^j(v)$. If there is no path leading from v to F_j, then $q^j(v) = -\infty$.*

In the original approach of [129], there is only one tail introduced for each operation due to the single sink node 1 in the disjunctive graph representation of the minimum makespan JSP. By considering the JSPTWT, the graph representation contains n finishing nodes $F_1, \ldots F_n$. Therefore, n tails are introduced for each operation, $(n+1) \cdot n \cdot m$ labels in total for the problem instance.

The following denotations are also important for the general concept. Focusing on operation v, its preceding and succeeding operations are denoted by:

- $PJ(v)$ - predecessor of v in the job sequence on the corresponding machine
- $SJ(v)$ - successor of v in the job sequence on the corresponding machine
- $PM(v)$ - predecessor of v in the machine sequence of the corresponding job
- $SM(v)$ - successor of v in the machine sequence of the corresponding job

Any path starting in the source 0 and leading to operation v has to include one of its predecessors $PJ(v)$ or $PM(v)$. Therefore, the longest path to operation v has to lead through one of these operations (see Fig. 5.1). The head of the operation as the length of the longest path depends on the heads of its preceding operations. For this reason, the head label is calculated recursively:

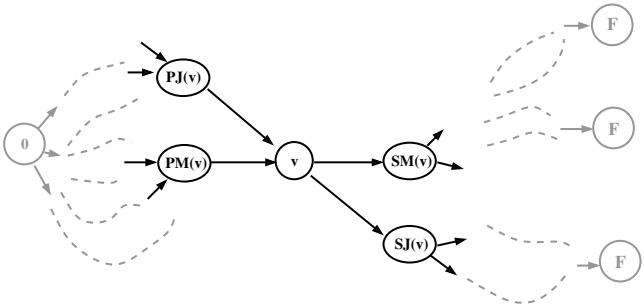


Fig. 5.1: Predecessors and successors of an operation v in G'.

$$h(v) = \max\{h(PJ(v)) + p_{PJ(v)}, h(PM(v)) + p_{PM(v)}\} \tag{5.1}$$

The same conclusion follows for the longest path leading from the operation v to a specified finishing node F_j. The tail(j) of the operation depends on its direct succeeding operations $SJ(v)$ and $SM(v)$ and their tails(j):

$$q^j(v) = \max\{q^j(SJ(v)) + p_v, q^j(SM(v)) + p_v\} \tag{5.2}$$

If the operation v belongs to the longest path from source 0 to the sink F_j, its head and tail(j) represent a segmentation of $\mathcal{LP}(0, F_j)$. The tardiness value of job j results from:

$$t_j = L(\mathcal{LP}(0, F_j)) = h(v) + q^j(v) \tag{5.3}$$

Another label for the characterization of an operation is defined by Mati et al. [91]. It is called the level of an operation:

Definition 8 (Level [91])**.** *The maximum number of operations on a path leading from the source 0 to operation v is called the level of v and is denoted by $l(v)$.*

Like the head label, the level of an operation depends on the level value of its preceding operations. For the course of such a path, see Fig. 5.1. It holds:

$$l(v) = \max\{l(PJ(v)) + 1, l(PM(v)) + 1\} \tag{5.4}$$

The level label helps to identify different paths leading from the source 0 to a finishing node F_j. Operations with the same level value cannot belong to the same

path [91]. Otherwise, the level value of one operation would be smaller than the other one. As a consequence for a lower bound procedure, assessing the path and its length for more than one operation can lead to a more accurate calculation of the lower bound value.

Basic principles of a lower bound procedure. The original concept has been introduced for solving the minimum makespan JSP. The objective function value is equal to the length of the longest path $\mathcal{LP}(0,1)$. Therefore, an operation v on the longest path leads to the calculation:

$$c_{max} = h(v) + q^1(v) \tag{5.5}$$

The first approach for estimating the objective of a neighboring solution relies on schedules generated by the single reversal of a critical arc [129]. Therefore, it can be used for local search procedures incorporating the CET neighborhood. The main idea of the estimation scheme is to recalculate the head and tail(1) of the affected operations adjacent to the reversed arc $u \rightarrow v$, since the starting time of these operations changes definitely in the new schedule. The following two heads and one tail(1) are updated based on the corresponding new predecessors and successors:

$$h'(v) = \max\{h(PM(v)) + p_{PM(v)}, h(PJ(u)) + p_{PJ(u)}\} \tag{5.6}$$

$$h'(u) = \max\{h(PM(u)) + p_{PM(u)}, h'(v) + p_v\} \tag{5.7}$$

and

$$q'^1(u) = \max\{q^1(SM(u)) + p_u, q^1(SJ(v)) + p_u\} \tag{5.8}$$

The estimation for the objective function value of the neighboring solution is obtained by assessing the maximum length of a path leading through u. The achieved value is called the lower bound value of the new solution, denoted by LB:

$$LB = \max\{h'(v) + p_v + q'^1(u), h'(u) + q'^1(u)\} \tag{5.9}$$

However, the overall longest path $\mathcal{LP}(0,1)$ in the graph representation of the newly generated schedule does not necessarily lead through the operation u. In this case, the value LB is lower than the length of $\mathcal{LP}(0,1)$. Thus, $LB \leq \bar{c}_{max}$ holds, with $\bar{c}_{max}$ as the exact objective function value of the neighboring schedule.

The time complexity of the estimation scheme is $\mathcal{O}(1)$ since there are exactly four values to be calculated.

Balas and Vazacopoulos [11] enhanced this basic concept for the use of the SCEI neighborhood. This neighborhood operator generates a new schedule by excluding an operation and inserting it at a new position in the same critical block. With this perturbation move, a set of operations receive a new position in the job sequence of the associated machine. For example, in the critical block $Q = \{s_1, u, r_1, \dots r_k, v\}$, operation u should be excluded and reinserted behind operation v. Heads and tails(1) of the affected operations $r_1, \dots r_k, v, u$ are recalculated in the new schedule according to their new predecessors and successors. For these recalculations, it is necessary to respect the sequence in the update process of the heads and tails(1) since the obtained values build off of each other, cf. Equations (5.6) and (5.7). The corresponding value of the lower bound for the minimum makespan JSP is the result of the composition of recalculated heads and tails:

$$LB = \max_{z \in \{r_1, \dots r_k, v, u\}} \{h'(z) + q'^1(z)\} \tag{5.10}$$

The time complexity of this lower bound procedure is $\mathcal{O}(n)$, since there are, at most, n operations in the critical block affected by the perturbation move.

5.2 A Lower Bound Procedure for the Application of the CET Neighborhood

The neighborhood operator CET [101] performs a reversal of a critical arc $u \to v$ at the beginning or end of a critical block, i. e. the corresponding adjacent operations u and v are swapped. The original concept of Taillard [128] for estimating the resulting solution quality in the new schedule has been adapted and enhanced by Mati et al. [91]. First, the calculation scheme of Taillard is adapted to the needs of the disjunctive graph model of the JSPTWT. Second, the authors have proposed a new approach for assessing the length of the longest paths. In combination, the both approaches return the lower bound value for the generated neighboring schedule.

The first part focuses on the calculation scheme for the tails. Due to the multiple finishing nodes in the graph representation it is necessary to update all tails(j) of operation u:

$$q'^{j}(u) = \max\{q^{j}(SM(u)) + p_u, q^{j}(SJ(v)) + p_u\} \qquad \forall j = 1, \ldots n \tag{5.11}$$

Heads of the operations u and v are updated according to Equations (5.6) and (5.7). Therefore, the first lower bound value for $\mathcal{LP}(0, F_j)$ results from:

$$LB_1(j) = \max\{h'(v) + p_v + q'^{j}(u), h'(u) + q'^{j}(u), 0\} \tag{5.12}$$

Note that the computation of the lower bound value is based on three components so that $LB_1(j) \geq 0$ holds as a result that the tardiness value of a job will never be negative.

The proposal for a second calculation scheme is motivated by the fact that the arc reversal does not have to affect all finishing nodes. For example, if there is no path leading from the operations u or v to the finishing node $F_{j^\star}$, the corresponding tail($\mathrm{j}^\star$) has the value $q'^{j^\star}(u) = -\infty$ and the estimation of $LB_1(j^\star)$ will be zero. On the other hand, the exact scheduling could lead to the late completion of job $j^\star$, and thus, produces a tardiness value. In summary, the estimation for job $j^\star$ would cause an underestimation which directly influences the total estimation of the objective function value for the neighboring schedule. This could lead to an exact assessment of this solution in the local search procedure, even though it is not necessarily an improving solution.

The second approach for calculating a lower bound value presented by Mati et al. [91] is based on alternative paths in the disjunctive graph representation. Alternative paths also lead from the source 0 to the finishing node F_j, but include exactly one operation with the same level like operation u:

$$LB_2(j) = \max_{w \in S}\{h(w) + q^{j}(w)\} \quad \text{whereas } S = \{w \neq u \mid l(w) = l(u)\} \tag{5.13}$$

At most, $\min\{n, m\}$ operations need to be checked, since each machine as well as every technological sequence features this characteristic in, at most, one operation.

The combination of both lower bound calculations ends up in the estimation of the tardiness value of job j. The estimation of the TWT value is given by the following calculation of the lower bound value which is denoted by LB_{MDL}[7]:

$$LB_{MDL} = \sum_{j \in J} w_j \cdot \max\{LB_1(j), LB_2(j)\} \leq \overline{TWT} \tag{5.14}$$

[7]MDL is the abbreviation for the research group of Mati, Dauzère-Pérès and Lahlou which have developed this lower bound procedure [91].

In case that $m \leq n$ holds, the time complexity is $\mathcal{O}(n \cdot m)$. More precisely, the calculation scheme of LB_1 needs $\mathcal{O}(n)$ due to the recalculation of n tails(j). The calculation of LB_2 needs $\mathcal{O}(n \cdot m)$ due to the check of n different tails(j) on, at most, m alternative paths.

5.3 A Lower Bound Procedure for the Application of the SCEI Neighborhood

The SCEI neighborhood operator [36] removes an operation from a critical block and inserts it again at the beginning or at the end of the block. Often, the direction of this shifting is specified as follows [22]: a backward shift move inserts the chosen operation at the front of the critical block. In a forward shift move, the operation is shifted to the end of the block. As mentioned in Section 5.1, based on the direction of the performed perturbation move, the sequence of heads and tails(j) has to be taken into account during the updating process.

Braune et al. [22] proposed the following lower bound procedure for the JSPTWT, which is based on the calculation scheme of Balas and Vazacopoulos introduced for the minimum makespan JSP [11]. The following aspects which enhance the former scheme are taken into consideration. For simplicity, the explanations are only illustrated on the forward shift move based on the critical block $Q = \{u, r_1, \ldots r_k, v\}$, i. e. the operation u is excluded and inserted behind v.

1. For the recalculation of a head, the corresponding preceding operation in the machine sequence $PM(.)$ is only taken into account if it is guaranteed that there is no additional parallel path. A parallel path is a second path leading from an operation in the block to one of its successor. For example, a parallel path starts in r_1 and ends in r_k without leading through another operation $r_t,\ \forall t = 2, \ldots k-1$. Obviously, the length of this parallel path is smaller than the path through the critical block Q, and the parallel path includes $PM(r_k)$. Updating the head of r_k could yield to an overestimation of $h'(r_k)$, since the head of $PM(r_k)$ also changes (due to the parallel path) and can be smaller. Therefore, the existence of parallel paths has to be checked in advance. Braune et al. [22] proposed several conditions in order to check if such parallel paths can be ruled out. If the existence of a parallel path is still open, the calculation

of the head is only based on the preceding operation in the job sequence (in the example: r_{k-1}).
The existence of parallel paths also influences the recalculation of tails(j) in an analogous manner.

2. Since the perturbation move in the critical block Q could affect the next succeeding operations $s_1, \ldots$ on this machine as well, i. e. the starting time of $s_1, \ldots s_l$ increase, these operations are also considered in the calculation scheme of heads and tails(j).
$\Rightarrow$ set $Q \cup A = \{u, r_1, \ldots r_k, v, s_1, \ldots s_l\}$

3. The calculations of heads and tails(j) are enhanced by the successors of the respective machine sequences, $SM(.)$. The calculation of the lower bound value is based on the recalculated heads and tails(j) of these successors. The reason for these "shifted" calculations is the possible invariance of the succeeding operations according to the performed perturbation move. In this way, the accuracy of the calculated lower bound value is improved further on.

4. The length of the longest path $L(\mathcal{LP}(0, F_j))$ in the origin schedule is taken into account if the perturbation move is executed in a critical block that does not belong to $\mathcal{LP}(0, F_j)$ and if there is still no path found to the finishing node F_j, i. e. $q'^j(z) = -\infty, \forall z \in Q \cup A$ in the new schedule.

5. Additionally, alternative paths leading through operations with the same level value like u or v are also checked (see Section 5.2).

The mentioned aspects and ideas lead to the following calculation scheme for a lower bound value [22]. The denotations $\bar{h}(.)$ and $\bar{q}^j(.)$ represent the exact value of the head and tails(j) in the new schedule:

$$h'(SM(z)) = \begin{cases} \max\{h(PJ(SM(z))) + p_{PJ(SM(z))}, h'(z) + p_z\}, \\ \qquad\qquad \text{if } \bar{h}(PJ(SM(z))) \geq h(PJ(SM(z))) \\ h'(z) + p_z \end{cases} \qquad (5.15)$$

$$\forall z \in Q \cup A$$

$$q'^j(SM(z)) = \begin{cases} q^j(SM(z)), & \text{if } \bar{q}^j(SM(z)) \geq q^j(SM(z)) \\ -\infty \end{cases} \quad \forall z \in Q \cup A \qquad (5.16)$$

The lower bound value LB_{BZA}[8] is decomposed into LB_1 and LB_2 as defined below:

$$LB_1(j) = \begin{cases} \max_{z \in Q \cup A}\{h'(SM(z)) + q'^j(SM(z)), 0\}, \\ \qquad\qquad\qquad \text{if } z \text{ is on the longest path of job } j \\ t_j \end{cases} \tag{5.17}$$

$$LB_2(j) = \max_{w \in S}\{h(w) + q^z(w)\} \quad \text{whereas } S = \{w \neq u \vee v \mid l(w) = l(u) \vee l(v)\} \tag{5.18}$$

$$\Rightarrow LB_{BZA} = \sum_{j \in J} w_j \cdot \max\{LB_1(j), LB_2(j)\} \leq \overline{TWT}$$

In case of $m \leq n$ holds, the time complexity for calculating LB_{BZA} is $\mathcal{O}(n^2)$. The calculation of the value LB_1 needs a time complexity of $\mathcal{O}(n^2)$, since the set $Q \cup A$ includes, at most, n operations. For each operation, n different tails(j) have to be checked. As already stated in Section 5.2, the calculation of LB_2 only needs a time complexity of $\mathcal{O}(n \cdot m)$.

Since the considered SCEI neighborhood includes all neighboring schedules generated by the CET operator (see Chapter 4), the lower bound procedure of Braune et al. [22] is also applicable for a local search employing the CET neighborhood. In comparison to the application of the lower bound procedure of Mati et al. [91], the accuracy of the lower bound values gets better. However, the computational effort is higher using the procedure of Braune et al. [22].

5.4 A new approach: Heads Updating

The following section introduces a new concept for assessing the quality of a neighboring solution. The concept is called Heads Updating. The basic idea is to introduce the head label to all nodes that are included in the disjunctive graph representation, in particular to B_j and F_j, $\forall j \in J$. The recalculation scheme only involves the heads of nodes which are directly or indirectly affected by the perturbation. A directly affected node represents an operation which receives a new position on its machine as a result of the applied perturbation scheme. The set of all succeeding nodes to these directly affected nodes constitutes the set of indirectly affected nodes. In contrast to the previously described procedures, the label tail(j) is not used.

[8] BZA is the abbreviation for the research group Braune, Zäpfel, Affenzeller which has developed this lower bound procedure [22].

Based on the definition in Section 5.1, the head of the completion node B_j represents the length of the longest path from 0 to B_j. Since B_j starts immediately after the completion of the last operation of job j, the head corresponds to the completion time of j. In this way, the head of the finishing node F_j corresponds to $L(\mathcal{LP}(0, F_j))$, i. e. the length of the longest path of job j, and thus, its tardiness value. As a result, the scheduling is directly given by the heads.

For the Heads Updating procedure, the following preliminary findings are important. As already mentioned in the introduction of the level label, there is no path between nodes with the same level value [91]. This result could be further extended:

Lemma 3. *Let G' be a directed graph which corresponds to a feasible schedule. Furthermore, let v and w be two different nodes with $l(v) \leq l(w)$. Then, there is no path leading from w to v in G'.*

Proof. Suppose, there is a path in the graph G' leading from w to v. Since $l(w)$ is the maximum number of arcs on a path to node w, there is at least one more arc on the path to node v. Thus, $l(w) < l(v)$ holds which creates a contradiction. □

Lemma 3 shows that a topological order of the nodes in the graph G' can be derived from the level label. The ordering enables a further extension of the principle:

Lemma 4. *Let G' be a directed graph which corresponds to a feasible schedule. Furthermore, let v and w be two different nodes with $w \neq SM(v)$, $w \neq SJ(v)$ and $l(v) + 1 = l(w)$. Then, there is no path leading from v to w in G'.*

The proof of this lemma follows the same argumentation as before. The additional information of Lemma 4 provides further relationships among the nodes.

In the Heads Updating procedure, a perturbation-dependent selection of directly and indirectly affected nodes is determined at the beginning. Therefore, based on the performed perturbation move, the level value of the first affected operation $z^\star$ has to be identified. Due to Lemma 3, the head of nodes with a level value lower or equal to $l(z^\star)$ cannot change, since paths from $z^\star$ to these nodes are precluded. The result of Lemma 4 helps to extend this node set, which can be omitted in the Heads Updating procedure. For all other nodes $z \in G'$ with $l(z) > l(z^\star) + 1$ as well as the two immediately subsequent nodes, heads are recalculated by means of Eq. (5.1). In doing this, the set of considered nodes includes the directly and indirectly affected nodes. Even though, the identified set contains some nodes whose head does not

Algorithm 1 Heads Updating Procedure

1: Initialization: feasible schedule G', perturbation move with operation $z^\star$ is the first affected node
2: Store all nodes z with $l(z) > l(z^\star) + 1$ in list L
3: Append $SM(z^\star)$ and $SJ(z^\star)$ to L (if not already done)
4: Sort the list L according to the level value
5: **for** the nodes z in the sorted list L **do**
6: $\quad h(z) = \max\{h(PM(z)) + p_{PM(z)}, h(PJ(z)) + p_{PJ(z)}\}$
7: **end for**

have to be recalculated necessarily, the computational effort of this approach is appropriate. In order to obtain a correct recalculation, the sequence of updated heads follows the topological order in this node set. This order based on the level value is necessary, since possible recalculations of the heads of preceding operations have to be executed first. The pseudo-code in Algorithm 1 shows the entire process of the Heads Updating procedure.

The example instance of Section 2.4 provides an illustration of the Heads Updating procedure. The feasible solution G' takes the role of the initial solution of the neighboring search (see the left part of Fig. 5.2). In this figure, the arc weights are skipped, but the level of each node is additionally given by a number above each node. Applying the CET operator to the arc $(2/3) \rightarrow (3/1)$ produces the neighboring schedule shown on the right part of Fig. 5.2. Since the first affected node in this perturbation is $(2/3)$ with $l(2/3) = 3$, all gray colored nodes are identified in the Heads Updating procedure for the recalculation of their heads. Note that the head of nodes B_2 and F_2 will have the former value, since there is no recalculation of node $(3/2)$ as the only preceding node.

The new procedure computes the exact head value of each node in the generated

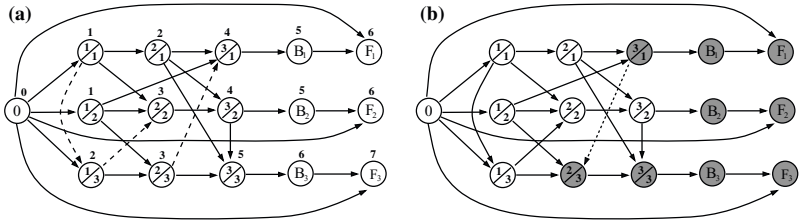


Fig. 5.2: Heads Updating: (a) feasible schedule G', (b) neighboring schedule through arc reversal, recalculation of heads for all gray colored nodes.

neighboring solution. Moreover, since the heads of the finishing nodes F_j are equal to the tardiness values of the jobs, the total weighted tardiness value of the neighboring schedule is easily determinable. In total, the Heads Updating procedure provides the following theoretical characteristics:

- Due to the exact assessment of the newly generated solution, the local search procedure is very effective. There is no misdirection in the search caused by non-improving solutions. The complete search process is reliable.
- In the worst case, $n \cdot (m + 2)$ heads have to be updated. In this scenario, the level of the first affected node is 1, and thus, all nodes may have to be integrated in the recalculation scheme. Hence, the time complexity is $\mathcal{O}(n \cdot m)$.
- The procedure is independent of the perturbation scheme of the used neighborhood operator. Therefore, it is applicable for all kinds of local search procedures.
- The storage of n tails for each of the $n \cdot m$ operations is dispensed.

For the assessment of neighboring schedules in a local search procedure, the Heads Updating procedure seems to provide more advantages than traditional lower bound procedures. However, the result of the worst case analysis shows a similar computational effort of the procedures. Anyway, it is difficult to compare the approaches analytically. With the help of a lower bound procedure, the estimation needs an exact determination of the objective function value, since recalculated values can differ from exact values. In the Heads Updating procedure, all recalculated heads are exact at every time. On the other hand, the average number of recalculated labels can be smaller using a traditional lower bound procedure. Therefore, the following section provides a brief performance test of the Heads Updating procedure in comparison to the described lower bound procedures.

5.5 Performance Test

The test suite in this performance test consists of three instance classes, each containing five randomly selected instances. The problem instances within a class have the same size $n \,\mathrm{x}\, m$: abz05, la16, la18, ft10 and orb02 with size 10 x 10, la21-la25 with size 15 x 10 and la26-la30 with size 20 x 10. Due dates and job weights are computed for all instances according to the procedure of Singer and Pinedo [121]

(cf. Section 7.1.1). The strictest due date factor $f = 1.3$ is exclusively used in the following experiment so that many jobs will be tardy in a schedule. As a result, the number of longest paths leading through the node set of G' will be comparatively high. Hence, the set of critical arcs and critical blocks will be larger and the number of neighboring schedules to be evaluated increases.

The different procedures for evaluating the objective function value of neighboring solutions are assessed in the following way. First, 1000 start solutions are generated randomly for each of the 15 problem instances. In the same manner as the performance analysis in Section 4.5, a steepest descent algorithm, applied to each generated start solution, delivers the associated local optimum. Each steepest descent algorithm is applied four times. In Runs 1 and 2, the neighborhood search uses the CET operator exclusively. In Runs 3 and 4, the generation of neighboring solutions is based on the application of the SCEI neighborhood. Furthermore, in Runs 1 and 3, the proposed lower bound procedures calculating LB_{MDL} [91] and LB_{BZA} [22] respectively provide an estimation of the solution quality of the newly generated schedules. The Heads Updating procedure (HUP) delivers an assessment of the total weighted tardiness of the neighboring solutions in Runs 2 and 4.

Using a lower bound procedure implies that if the return value of the procedure is lower than the TWT value of the current schedule, the exact objective function value is computed in the next step, i. e. the starting times of the operations are determined. Otherwise, if the return value of the lower bound procedure is larger than the total weighted tardiness of the current start solution, the neighboring solution is immediately rejected. The comparison with the objective function value of the best found solution so far yields the result if a new best solution is identified. Fig. 5.3(a) shows the neighbor evaluation process using a lower bound procedure.

The Heads Updating procedure reduces the computational steps of the neighbor evaluation process (see Fig. 5.3(b)). Due to the exact assessment, the comparison with the best found solution is immediately performed.

Another important difference between the approaches is the recalculation of the labels after each improving step of the steepest descent algorithm. After identifying the best improving solution in the set of neighboring schedules, the labels head and tails(j) have to be recalculated exactly while using a lower bound procedure. This computational step is necessary, since otherwise the next evaluation of a neighboring schedule could not yield the correct lower bound value. In contrast to

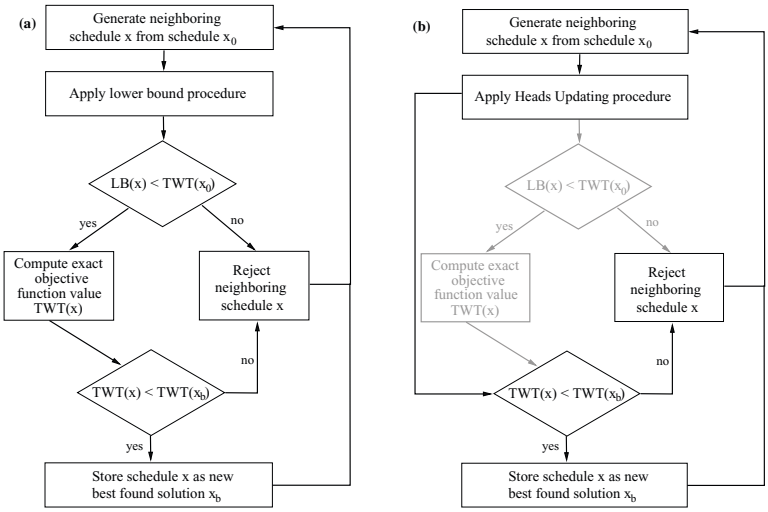


Fig. 5.3: Neighbor evaluation process in the steepest descent algorithm: (a) using a lower bound procedure, (b) using the Heads Updating procedure (x_0 - current start solution, x_b - best improving solution, x - neighboring schedule).

the lower bound approaches, the Heads Updating procedure does not require this computational step. The head labels are correctly recalculated after each improving step.

Since the difference between the accuracy of the lower bound procedures and the accuracy of the Heads Updating procedure is already clarified, the aim of the following experiment[9] is to measure the resulting computational effort caused by the different neighbor evaluation procedures.

Experiment 7: The aim of this experiment is to assess the implications on the computational effort by means of the run-time. Therefore, all generated neighboring schedules in the steepest descent algorithms are assessed by either a traditional lower bound procedure (LB_{MDL} in Run 1 and LB_{BZA} in Run 3) or the Heads Updating procedure (in Runs 2 and 4). Tab. 5.1 presents the total computation times for every of the three instance classes as well as for every run. The Heads Updating procedure clearly outperforms the lower bound procedures. For example, using the lower bound procedure of Mati et al. (LB_{MDL}) for solving the instances with size 10 x 10 takes

[9]The enumeration of Section 4.5 is continued at this point.

CET neighborhood				**SCEI neighborhood**			
Run | Eval.	10 x 10	15 x 10	20 x 10	Run | Eval.	10 x 10	15 x 10	20 x 10
1 | LB_{MDL}	19.6	140.8	1445.1	3 | LB_{BZA}	26.9	196.6	1817.8
2 | HUP	5.0	10.1	19.7	4 | HUP	8.2	24.8	76.6

Tab. 5.1: Computation times (in seconds) regarding the different assessment approaches.

19.6 seconds whereas the Heads Updating procedure consumes only 5.0 seconds. The main handicap of the lower bound procedures is the exact determination of heads and tails(j) after each improvement step of the steepest descent algorithm. In particular, the recalculation of the $n \cdot (n \cdot m)$ tails(j) is very time-consuming. This disadvantage becomes more relevant if larger problem instances have to be solved.

5.6 Summary

The assessment of neighboring solutions by means of their objective function value is an important part in a neighborhood search procedure. Two different approximation procedures known from the subject literature provide an estimation of the solution quality by calculating a lower bound value. On the other hand, this chapter contains a new concept which is called Heads Updating. In the corresponding procedure, the representation of a schedule as a disjunctive graph forms the basis for the assessment of newly generated schedules. In contrast to the calculation schemes of traditional lower bound procedures, the main features of Heads Updating are the exact assessment of the objective function value and its applicability on any neighborhood perturbation scheme. The performance comparison of the evaluation procedures has shown that the implementation of the Heads Updating procedure leads to a significant acceleration in the neighborhood search. For this reason, applying Heads Updating can be strongly recommended for solving the JSPTWT using a local search algorithm.

Chapter 6

Solving the JSPTWT - a new Solution Procedure

The primary goal of this thesis is to present the concept of a new solution procedure for solving the JSPTWT. As shown in Chapter 3, various solution procedures published in the related literature already exist (see e. g. [19, 41, 152]). Research results from the last few years show that these procedures are basically well performing. Their development is mainly based either on the adaptation of algorithms introduced for the minimum makespan JSP or on the combination of several different metaheuristics in a hybrid approach. There is no explanation why these solution procedures are also successful for solving the JSPTWT. Moreover, among the plenitude of metaheuristic concepts, there is no solution procedure which dominates the other approaches, cf. e. g. Appendix C.

This chapter deals with two different subjects. The first section provides an overview of different metaheuristic concepts. This class of solution procedures includes powerful algorithms and techniques which are able to find solutions of high quality. In the second section, a new solution procedure for solving the JSPTWT is introduced called EGRASP. This section provides a description of the algorithmic concept and the motivation behind.

6.1 Metaheuristic Concepts

As aforementioned, operations research scientists have developed many metaheuristic search concepts over the last 30 years. The motivation behind is the today's

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_6,

consideration of more complex combinatorial optimization problems. For example, the search space is typically incredible large in real-world problems, and the number of constraints induces a complexity that sometimes hinders building even one feasible solution [97]. For this reason, metaheuristics are often the method of choice once the considered problem belongs to the class of $\mathcal{NP}$-hard problems [116].

6.1.1 Basic Concept of Metaheuristics

In operations research literature, there are various definitions for the term metaheuristic. The following definition proposed by the metaheuristic network community is related to the area of application:

Definition 9 (Metaheuristic[10])**.** *A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem.*

In 1996, Osman and Laporte [106] presented a fairly different definition which refers to the design and the behavior of a search procedure:

Definition 10 (Metaheuristic [106])**.** *A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions.*

It is widely accepted that metaheuristics form a class of solution techniques which can be understood as an upper level methodology [130]. A metaheuristic guides the search process of a subordinate heuristic, which is designed to solve a specific problem. As a result, metaheuristics cannot guarantee to find an optimal solution. However, they often deliver high-quality solutions in relatively short time [116] since they strengthen the power of the subordinate heuristic.

Following the classification offered by Talbi [130], metaheuristics belong to the family of heuristics, see Fig. 6.1. Heuristics themselves belong to the group of approximate algorithms in the totality of solution procedures. The classification is

[10] cf. Metaheuristics Network Website: http://www.metaheuristics.org/

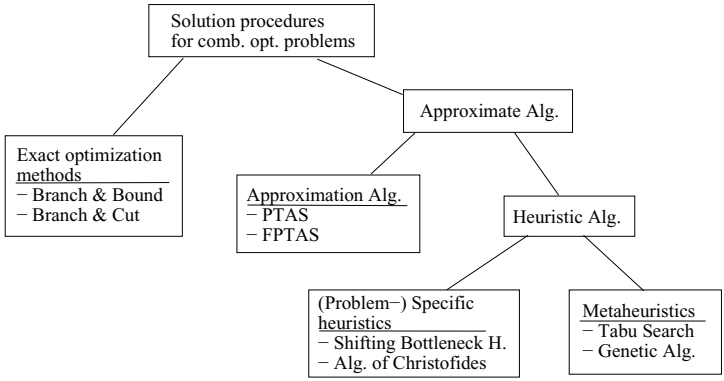


Fig. 6.1: Classification of solution procedures according to Talbi [130].

based on the following criteria: finding an optimal solution or not (→ exact method or approximate algorithm), providing a provable solution quality (→ approximation algorithm or heuristic algorithm) and the area of application (→ problem-specific heuristic or metaheuristic).

The class of metaheuristics itself involves different types of solution techniques. This subclassification results from several characteristics, such as an inspiration by a natural optimization process, using a set of different neighborhoods, incorporating memory or being memory-less [21]. Fig. 6.2 shows the corresponding morphological structure of metaheuristics. In this figure, two metaheuristics are classified in order to provide an illustration: tabu search and genetic algorithms. Details of these two metaheuristic concepts and reasons for their classification are presented in the subsequent subsection.

A frequently used characterization of metaheuristics addresses the criterion of the number of solutions in process which yields either a population-based or a single-solution metaheuristic. A population-based metaheuristic handles a set of solutions in parallel. While a single-solution metaheuristic progresses with one solution by changing it systematically. The subordinate heuristic in a single-solution metaheuristic is typically a local search procedure which performs modifications already introduced as perturbations moves (see Chapter 4).

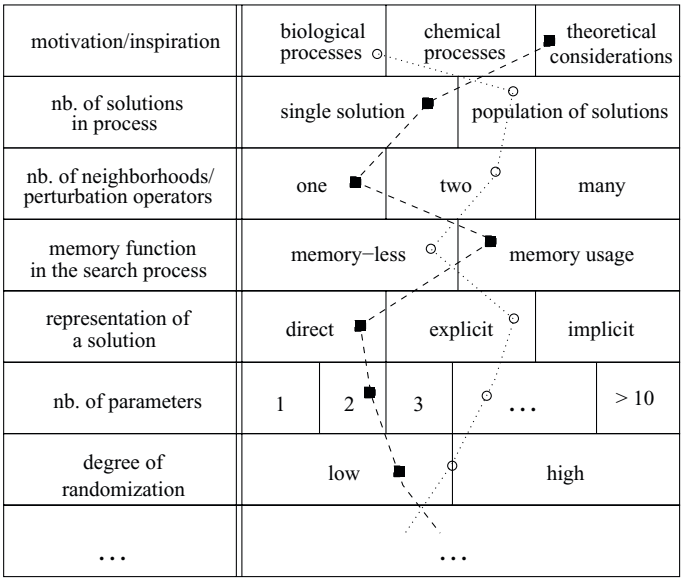


Fig. 6.2: Morphology of metaheuristics, derived from Blum and Roli [21], with classification of tabu search(■) and genetic algorithm (o).

6.1.2 Some Metaheuristics

In order to exemplify different search concepts, four single-solution metaheuristics and two population-based metaheuristics will briefly be sketched. A detailed description as well as other metaheuristics can be found in the handbook from Glover and Kochenberger [53] and the book of Zäpfel et al. [146].

Tabu Search (TS). The basic idea of TS was first introduced by Glover [51, 52]. TS performs a local search procedure which replaces the current solution by its best neighboring solution. In order to avoid visiting already known solutions, the metaheuristic uses a tabu list. In this list, perturbation moves or specific attributes of a solution are stored and then prohibited from being reused in the next steps. The deletion of old list elements is managed by a control strategy so that after few iterations, prohibited attributes become reachable again.
For this reason, tabu search is classified as follows (see Fig. 6.2). TS is motivated by theoretical considerations how the search process is able to escape from a local

optimum. It belongs to the class of single-solution metaheuristics. TS employs one neighborhood operator to generate new solutions etc.
Tabu search became very popular in the 1990s for solving scheduling problems, in particular the minimum makespan JSP; e. g. by Nowicki and Smutnicki's famous publication "A Fast Taboo Search Algorithm for the Job Shop Problem" [101] or in the work of Dell'Amico and Trubian [36] as well as in the work of Taillard [129].

Iterated Local Search (ILS). As already indicated by its name, this metaheuristic aims at an enhancement of a basic local search procedure. The core forms a hill climbing procedure which iteratively improves the solution quality by replacing the current solution with a new and better neighboring solution. As a result, this procedure ends up in a local optimum with no further possible improvement. The idea of an ILS as proposed by Stützle [89, 127] is to perturb the found local optimum again. For this purpose, the neighborhood operator is applied several times without regard to the resulting solution quality. In this way, the search procedure jumps to a new adjacent region of the search space and the search process is continued.
Kreipl [76] as well as Mati et al. [91] have used this relatively simple metaheuristic for solving JSPTWT instances. Further applications as well as variants of iterated local search are reported by Lourenço et al. [90].

Variable Neighborhood Search (VNS). This metaheuristic concept was introduced by Mladenovic and Hansen in 1997 [99]. It is motivated by the fact that a local optimum generated with one neighborhood operator is not necessarily a local optimum for other neighborhood operators. Therefore, the basic idea of VNS is to incorporate different neighborhoods which are used iteratively. If one neighborhood operator cannot detect further improvements for the current solution, the operator is replaced by another neighborhood operator. The overall search process terminates if a solution is found that is locally optimal for all implemented neighborhoods.
An overview of variants and a list of applications for this metaheuristic is presented in the work of Hansen et al. [59]. An example application is described in the paper of Driessel and Mönch [40] using the VNS for solving a parallel machine scheduling problem.

Greedy Randomized Adaptive Search Procedure (GRASP). GRASP consists of two phases, a construction phase for generating solutions and an improvement phase that applies a local search procedure to improve the initial solution [112]. This

process is iteratively repeated until a predefined termination criterion is reached. The algorithmic concept was developed and introduced by Feo and Resende [43]. The design of the GRASP as well as variants and extensions are reported in [44], and an overview of applications is provided in [45]. A GRASP algorithm designed for solving the minimum makespan JSP is described in [17].

Further single-solution metaheuristics which incorporate local search as subordinate heuristic are e. g. simulated annealing (proposed by Kirkpatrick et al. [73]) and guided local search (developed by Voudouris and Tsang [140]).

Genetic Algorithm (GA). The metaheuristic concept of the genetic algorithm is one of the most famous and most often referenced approaches for tackling combinatorial optimization problems. The basic idea relies on the work of Holland [62], and is continued by Goldberg [54, 55] and Mitchell [98]. Initially, a population of chromosomes which represents a set of different solutions to a problem is created. Usually, a chromosome is a string of binary values such that a decoding and encoding scheme is necessary for transferring the chromosome into a solution and vice versa. The main part of a genetic algorithm is the selection and generation process for a new population. Based on a selection rule and a crossover operator, two current chromosomes are recombined in order to produce a new solution. The mutation operator creates small changes of solutions in order to diversify the search further on. Afterwards, the new population replaces the old one and the whole process is repeated until a specific termination criterion is met. It is noteworthy that with regard to scheduling problems, permutation representations of solutions have received attention; often replacing the binary representation is leading to superior results [16].
As a result, genetic algorithms are classified into the morphological structure as follows (see Fig. 6.2): GA are inspired by the natural evolution process. They belong to the class of population-based metaheuristics. In a GA, a crossover operator and a mutation operator are applied as perturbation operators in order to obtain new solutions etc.
Portmann [114] provides an overview of genetic algorithms with applications to scheduling problems. In several publications, the job shop scheduling problem is tackled with variants of genetic algorithms, see e. g. [42, 94].

Ant Colony Optimization (ACO). Natural ants look for the shortest path between their nest and a new source of food. They benefit from pheromone trails on sub-paths whose intensity signals the frequency of use by other ants having already taken the same path. Dorigo has adapted this behavior for the development of the ant colony optimization algorithm [38, 39]. In this metaheuristic, one solution is produced by an ant which successively chooses the next element to be incorporated into its current partial solution. The selection process is guided by some heuristic information and the amount of pheromones associated with this element. The pheromones represent an assessment which measures the frequency of the considered element in previous iterations of the algorithm. Each iteration ends with an updating process of the pheromones in order to control the search process additionally. Even though ant colony optimization is a comparatively new metaheuristic, there already exist some applications to scheduling problems [64, 87].

Further population-based metaheuristics are e. g. scatter search (introduced by Glover [50]), particle swarm optimization (developed by Kennedy and Eberhart [71]) and artificial bee colony algorithm (proposed by Karaboga and Basturk [69]).

6.1.3 The Fitness Landscape: a brief Side Trip

The development of solution procedures should take the search behavior related to a given problem into account [142]. For combinatorial optimization problems, representing the problem structure as a fitness landscape is a common approach. This framework is well suited for describing the search process of heuristics applied when solving these problems [113]. An effective search procedure should incorporate a navigation strategy that guides the search according to information taken from the fitness landscape [142].

The theory of fitness landscapes has its origins in the biological sciences [70]. The description of the dynamic of nature forces has led to the introduction of the notion and its framework [96]. Several statistical investigations and studies published to this topic are applicable to combinatorial optimization problems (see e. g. [123, 144]). Investigations based on the problem structure of specific optimization problems and analyses of search processes have been published over the last twenty years. However, in general, research results are relatively rare. Nowicki and Smutnicki try to push for further research, since "it is not enough to search, one should know where to

search'[11].

This subsection provides an overview of the fitness landscape concept, research results for the minimum makespan JSP and derived implications for the fitness landscape of the JSPTWT. A broad fitness landscape analysis of the JSPTWT is not subject of this thesis. The subsection predominantly aims at presenting a brief insight in the theory of fitness landscapes.

The Concept of the Fitness Landscape

The fitness landscape is a representational form for a problem instance of a combinatorial optimization problem with the aim of visualizing the problem structure. Following the introduction to fitness landscapes of Merz and Freisleben [96], the set of all feasible solutions is called the search space. A metric defines a distance value for each pair of solutions. As a result, the minimum distance value creates a neighborhood for each solution. Therefore, the perturbation operator implemented in a neighborhood search procedure represents the connectivity among these solutions, and thus, is strongly related to the distance metric. The fitness of a solution is given by its objective function value. In summary, the fitness landscape is composed by the search space, the chosen distance metric and the fitness function. The term "fitness landscape" is sometimes synonymous with the search space [142]. In scientific community, the fitness landscape is also referred to the visualization of the search space as a graph [142].

For the representation of local search algorithms, the distance between two solutions in the landscape is equal to the minimum number of required executions of the chosen perturbation operator. Different operators yield different connectivities in the search space, and thus, create different surface structures of the fitness landscape. The resulting connectivity among the solutions enables generating a walk in the landscape passing different solutions. Therefore, the search process of a solution procedure is representable as walking in the associate fitness landscape. The knowledge about the landscape structure helps to understand the search behavior of different solution procedures.

With respect to a minimization problem, the process of local search corresponds to a downhill walk in the landscape ending up in a local optimum. But the obtained solution does not have to be the global optimum. The navigation strategy for

[11] cf. E. Nowicki, C. Smutnicki: Some new ideas in TS for Job Shop Scheduling [103], page 181.

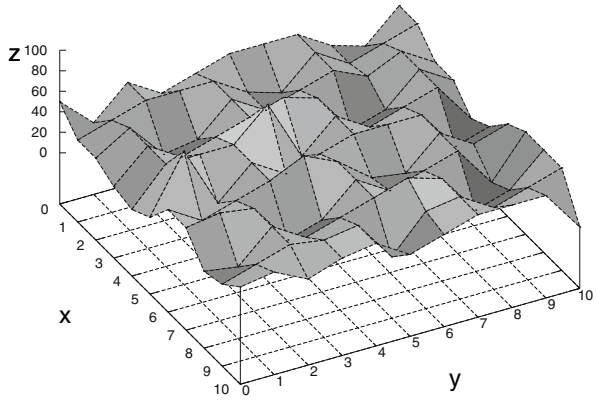


Fig. 6.3: Fictive example of a fitness landscape.

escaping from a local optimum has to be capable concerning the surface structure of the landscape. Otherwise, the process may revert back to the already known solution.

Fig. 6.3 illustrates the concept of the fitness landscape by a fictive example. In this example, the problem is to determine two variables x and y so that the value $z = f(x, y)$ is minimal.[12] For each variable, an integer value in the range $[0, 10]$ is feasible. Therefore, every integer 2-tuple (x, y) with $0 \leq x, y \leq 10$ represents one of 121 feasible solutions. In this example, the used distance metric is the Manhattan metric. Two solutions are connected with each other if the values from exactly one of the two variables differ by one. As a result, the connectivity among the solutions is represented by the grid in the figure. The fitness values are plotted according to the z-axis creating a surface structure. The search process can be described as a single walk over the edges in the pictured landscape.

The fitness landscape is characterized with the help of a specific terminology adapted from geographical structures [142]. A valley within the landscape denotes a solution (or a set of adjacent solutions) when directly connected (neighboring) solutions show a poorer solution quality. The term "valley" is used to represent local optima found in the search process. A special valley in the landscape is called

[12]The values $f(x, y)$ are randomly chosen in the range $[10, 80]$.

the big valley which indicates a vicinity for the location of the global optima as well as other high-quality solutions. A plateau denotes a connected set of solutions with equal fitness. One purpose of the landscape analysis is to discover the solution backbone of the represented problem instance. The backbone is defined as set of attributes that is common in all global optima. The term "ruggedness" is used to characterize the surface structure of the landscape. In general, a very rugged surface of the fitness landscape induce a high effort to find solutions of high quality.

The aim of the fitness landscape representation is to investigate its surface structure in order to obtain information about the search process of the solution procedure. The surface structure depends on the chosen problem instance so that a common implication for the problem structure and the search process is available by empirical tests using a set of problem instances. Research results in the literature provide several statistics and tests for the evaluation of e. g. the location of the local and global optima. The following list shows some approaches for obtaining information about the fitness landscape:

- The entropy measures the average occurrence of a solution attribute in a pool of solutions [95]. The comparison of a pool of random solutions with a pool of locally optimal solution by means of their entropies provides an assessment about the distribution of local optima. Strongly distributed local optima are more difficult to handle by a solution procedure since there is no information about the location of the global optima.
- The Fitness-Distance-Correlation (FDC) in a pool of local optima measures the correlation between the distance of two arbitrary local optima and the difference in their fitness values. A high FDC indicates that nearby local optima have similar fitness values. The global surface of the fitness landscape shows an inclination towards the global optima which can be exploited by a solution procedure. On the other hand, a low FDC implies that high-quality solutions can be found in nearly every region of the landscape [142].
- The Fitness-Distance-Correlation between a local optimum and its nearest global optimum is highly correlated to the success of a search algorithm [143]. Measuring a high FDC implies that the identification of a high-quality solution has led the search algorithm into a very promising region containing a global optimum. Therefore, the solution procedure should intensify the search process close to this high-quality solution.

- An evaluation of the surface structure of the landscape provides the autocorrelation measured in a random walk which is performed on the landscape. The autocorrelation is a statistic for investigating the stationary of a time series. It measures the similarities in a time series by comparing the series with the "time-shifted" series [23]. If the autocorrelation vanishes fast the global surface of the fitness landscape is more rugged and mountainous [70, 95].

Findings on the Fitness Landscape of the minimum makespan JSP

In the context of the job shop scheduling problem, previous studies have already focused on the fitness landscape of the minimum makespan JSP, see e. g. [95, 102, 126, 141]. The published results and implications are little which additionally demonstrate the difficulty to derive exact and useful information.

Without a doubt, local optima are spread nearly uniformly over the entire landscape [95]. Moreover, the Fitness-Distance-Correlation in a pool of locally optimal solution is relatively low. Therefore, high-quality solutions can be found in nearly every region of the landscape.

Based on a study on the fitness landscapes of several benchmark instances, recommendations have been derived for the application of navigation strategies [95]. The guidance of search should be based mainly on a single-solution search concept because the observed landscape structure may pose an obstacle for a successful application of population-based search using recombination operators like in genetic algorithms.

Streeter and Smith have presented the result that the ratio of the number of jobs to the number of machines ($n:m$) influences the shape of the fitness landscape [126]. For problems with a high or low ratio (for example $5:1$ or $1:5$), the ruggedness of the landscape vanishes. Therefore, a solution procedure can find high-quality solutions easier.

Nowicki and Smutnicki have shown that there is a big valley structure in the fitness landscape of the minimum makespan JSP [103]. The existence of the big valley structure has been exploited by improving the successful tabu search procedure of Nowicki and Smutnicki [102]. The idea is to determine the location of the big valley and then to search primarily through this part of the landscape.

Pardalos et al. [108] have proposed a procedure which relies on the backbone property. In this procedure, a set of arcs occurring in the best found solutions

is selected and fixed for the next generation of new schedules. This procedure is embedded in a competitive metaheuristic called distance based search.

Moreover, the concept of path relinking is used more and more in recently published metaheuristics, see e. g. the advanced tabu search algorithm of Nowicki and Smutnicki [102]. Path relinking guides the search process into promising regions of the landscape by exploring the search space between high-quality solutions. Therefore, this solution technique is adequate for exploring a big valley structure [103].

To summarize, the fitness landscape of the minimum makespan JSP consists of a multitude of local optima. They exist in nearly every region of the landscape, and it is very difficult to derive the accurate information for guiding strategies to direct the search to a global optimum. The global surface provides less information about the location of the big valley structure. To overcome this handicap, special solution techniques like path relinking are recommended.

Implications for the Fitness Landscape of the JSPTWT

It is expected that the fitness landscape derived from the JSPTWT does not provide an easier surface structure than the landscape derived from the minimum makespan JSP. This expectation is motivated by two reasons. First, the classification of the JSPTWT into the class of $\mathcal{NP}$-hard problems has shown the difficulty for finding optimal solutions. Classifying special cases like the problems $1 \mid\mid \sum w_j t_j$ and $1 \mid\mid c_{max}$ has demonstrated the increased complexity resulting from the TWT objective. Second, the search space of the JSPTWT is identical with the search space of the minimum makespan JSP. The set of feasible solutions given by the problem structure is independent from the considered objective function. As a matter of course, the comparison of the fitness landscapes is based on the same distance metric. The surface structure of the two landscapes differ only by the associated fitness function. For this reason, it is very likely that the surface structures of the two fitness landscapes are similar to each other.

The following experiment is used to get more information about the fitness landscape derived from the JSPTWT in order to illustrate the mentioned expectation. The experiment aims at determining the location and distribution of the local optima in the fitness landscape. Moreover, by providing a comparison with the distribution of local optima in the fitness landscape derived from the minimum makespan JSP, the expectation about the similarities of the surface structures is checked empirically.

For the following experiment, three different sets of problem instances have been randomly chosen. The first set contains the five instances orb01-orb05, each with 10 jobs and 10 machines. The second set contains the problem instances la36-la40 with size 15 x 15. The instances tai21-tai25 with size 20 x 20 constitute the third set. For all 15 problem instances, due dates and weights are calculated in advance using the procedure of Singer and Pinedo [121]. The due date factor $f = 1.3$ produces the due dates of the instances in these experiments.

In the first step, a pool P_{rand} of 1000 random schedules is generated for each of the 15 problem instances. Afterwards, a steepest descent algorithm is applied to every random solution. Due to its basic character as perturbation operator, the critical transpose neighborhood operator [136] is employed in the steepest descent algorithm. In doing this, the computations yield a pool P_{twt} of 1000 local optima with regard to the total weighted tardiness objective. The computations are repeated by minimizing the makespan as objective such that another pool P_{cmax} of local optima is generated.

Experiment 8. The goal of this experiment is to assess the location and distribution of local optima in the fitness landscape like in [95]. Focusing on a random solution which has served as initial solution of the steepest descent algorithm, two local optima have been generated. The first local optimum is determined while considering the minimization of the total weighted tardiness, and therefore, it is assigned to the pool P_{twt}. The second local optimum is obtained regarding the makespan objective. This solution is assigned to the pool P_{cmax}. The three pools provide an appropriate basis for the desired evaluation. It is worth mentioning that there are no identical solutions within every pool.

Every solution in a pool is represented by a binary vector [95]. The bits in this vector encode the precedence relations on the machines (i. e. job j precedes k on machine m). The encoding scheme enables using the distance metric of the Hamming distance [96]. Two schedules have a Hamming distance of d if and only if the number of different bits in their binary vectors is d. The average, normalized[13] Hamming distance AHD between the solutions in every pool is computed and shown in the first part of Tab. 6.1. The AHD differs between the three sets of instances. The 10 x 10 problem instances deliver the most diverse pool of random schedules

[13]The normalization of AHD is related to the length of the binary vector and provides an appropriate comparison with regard to the different instance sizes.

		P_{rand}	P_{twt}	P_{cmax}
	10x10	0.293	0.296	0.291
AHD	15x15	0.167	0.168	0.166
	20x20	0.154	0.154	0.153
	10x10	0.631	0.636	0.626
$\mathcal{E}$	15x15	0.376	0.378	0.373
	20x20	0.346	0.348	0.344

Tab. 6.1: Results of the diversity measurements.

P_{rand} with $AHD = 0.293$. For the minimum makespan JSP, it is known that the AHD does not differ substantially between P_{rand} and P_{cmax} [95]. This observation also holds true for the relationship between P_{rand} and P_{twt}. The AHD varies in a range of no more than 0.3% between the different pools. The measured AHD in the pools confirm the same diversity of the local optima for the JSPTWT and minimum makespan JSP. Moreover, the steepest descent algorithm cannot reduce the diversity from random solutions to local optima. Hence, local optima are spread over the entire fitness landscapes.

Another assessment of the generated local optima is available by measuring the entropy $\mathcal{E}$ in the pools. The entropy indicates the variety of the precedence relations which occurs in the solution set of a pool. Focusing on the precedence relation that job j precedes job k on machine m, the number of solutions ω_{mjk} is counted in a pool where this precedence relation is true. The entropy associated with this precedence relation is calculated as follows:

$$\mathcal{E}_{mjk} = \frac{-1}{\log\sqrt{2}} \cdot \frac{\omega_{mjk}}{1000} \cdot \log\frac{\omega_{mjk}}{1000}$$

$\mathcal{E}$ is the average entropy for all precedence relations, and thus, represents the average variety observed in the pools. The computed values of $\mathcal{E}$ are presented in the second part of Tab. 6.1. These values vary again between the different sets of instances. More important is that the variations between the different pools of solutions are very small. The entropy in the pools confirm that the steepest descent algorithm cannot reduce the diversity from random solutions to local optima. The algorithm is not capable to navigate the search into one direction.

The statistical investigations have clarified that the location and distribution of local optima is very similar in both fitness landscapes. Both surface structures show a multitude of valleys.

		P_{twt}	P_{cmax}
# Imp	10x10	9.00	5.32
	15x15	15.97	7.00
	20x20	25.64	8.64

Tab. 6.2: Average improving steps performed in the steepest descent algorithms.

Another information about the local structure of the fitness landscapes is provided by the average improving steps (# Imp) performed by the steepest descent algorithms in Experiment 8. This statistic indicates the average number of made descents for generating a local optimum. Tab. 6.2 presents the average improving steps related to the different set of instances. The observed higher number of descents and the given identical search space imply that the valleys of the fitness landscape in the JSPTWT are deeper. Starting from a randomly generated solution, the steepest descent algorithm needs more effort to identify the local optimum regarding the TWT objective.

To summarize, Experiment 8 has delivered an assessment about the location and distribution of local optima in the fitness landscape of the JSPTWT. The results have verified that the mentioned expectation about the similarities of the fitness landscape derived from the JSPTWT and the fitness landscape derived from the minimum makespan JSP is well-founded.

To conclude, a solution procedure for solving hard JSPTWT instances should cover two main features: first, a powerful local search procedure which explores the deep valleys in the landscape in a fast way. Second, an effective global strategy should detect the area of the search space containing the global optima. For this purpose, good local optima already found in the search process should influence the subsequent search in an advantageous manner, like it is performed by path relinking.

6.2 Algorithmic Concept for a new Solution Procedure

As the superior goal of this thesis, a new solution procedure developed for solving the JSPTWT is designed which bases on the metaheuristic GRASP, the "Greedy Randomized Adaptive Search Procedure". This section starts with the motivation why this metaheuristic seems to be suitable for solving difficult JSPTWT instances.

Afterwards, the components of the solution procedure are described in detail. The section concludes with the settings of the parameter values.

6.2.1 Motivation and Overview of the Algorithmic Concept

The decision to use the search concept of GRASP for solving the JSPTWT is motivated by at least three reasons. The first reason is that GRASP employs a local search procedure as subordinate heuristic. GRASP belongs to the class of single-solution metaheuristics and involves a methodology for the management of a local search algorithm. As already mentioned in Chapter 1, this feature is important and should form one main component in the solution procedure. Moreover, the recommendation derived from the fitness landscape, that a powerful local search should be embedded in the solution procedure, is complied.

The second reason is that GRASP produces a series of local optima which are independent of each other. After one local optimum is detected, the subsequent search process is started at an arbitrary position in the search space. In doing this, GRASP enables to guide the search to the next desired region without much effort. Other single-solution metaheuristics follows a single search trajectory (like tabu search or iterated local search), and thus, need more effort to explore far distant regions of the search space.

The third reason is that the metaheuristic concept of GRASP is flexible with regard to implementing further search algorithms and mechanisms. In doing this, the local search procedure can be designed to become very powerful in the exploration of the local structures in the current search region. Additional methodologies can help to guide the search to the most promising regions with high-quality solutions. This feature is important since a recommendation of the fitness landscape has been to develop an effective navigation strategy for the search process.

A new solution procedure has been developed which is based on the concept of GRASP. Its main structure is sketched in Fig. 6.4. After the initialization, i. e. reading the problem data of the instance and determining the termination criterion, the solution procedure starts with a learning phase. In this first phase, the procedure performs conventional GRASP iterations. In each of these GRASP iterations, a construction algorithm is started in order to generate a feasible schedule. Subsequently, the improvement algorithm takes this constructed schedule as initial schedule and applies a local search procedure. The improvement algorithm terminates if a solution

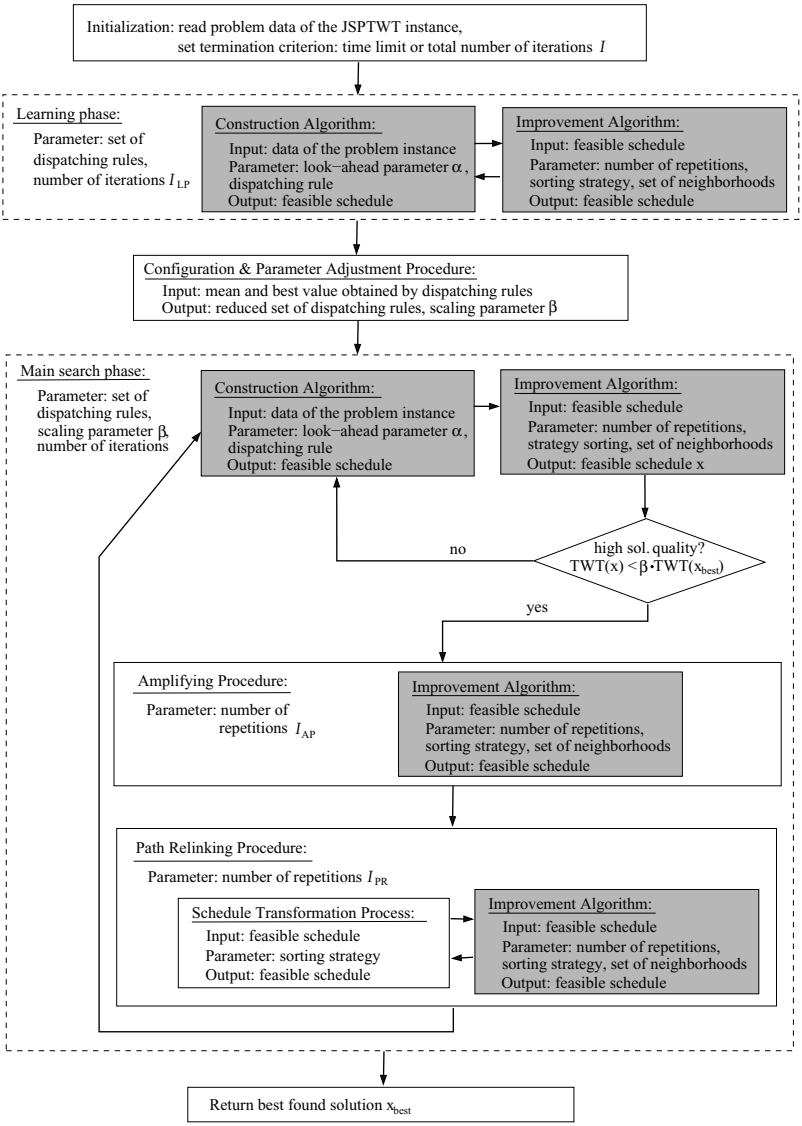


Fig. 6.4: Flowchart of the solution procedure EGRASP.

is created which is locally optimal with regard to the used neighborhood operators. At the same time, the current GRASP iteration is completed.

The learning phase runs a fixed number of GRASP iterations with the aim to collect information about the performance of various configurations. More precisely, the construction algorithm uses a dispatching rule to derive priorities among operations. In the learning phase, each of a given set of dispatching rules is used for a fixed number of GRASP iterations.

The subsequent procedure of the learning phase is called Configuration & Parameter Adjustment Procedure and is the first adaptive component. The Configuration & Parameter Adjustment Procedure reduces the set of available dispatching rules so that only the best performing rules are used in the following main search. Moreover, the parameter β is calculated which is used to separate high-quality solutions and low-quality solutions.

After the selection of dispatching rules and the adjustment of parameters, the configuration of the main search is completed and the corresponding main search phase is started. The basic parts in the main search form the construction algorithm and the improvement algorithm, composing a GRASP iteration. At the end of this GRASP iteration, the solution quality of the generated local optimum is checked. If the resulting total weighted tardiness value is larger than a bound (which is set as β times the best TWT value so far), the found local optimum yields a low quality. Therefore, the next GRASP iteration is started. Otherwise, if the resulting total weighted tardiness is lower than the bound, a solution of high quality has been generated. This outcome leads to the execution of two further adaptive components: the Amplifying Procedure and the Path Relinking Procedure.

The Amplifying Procedure takes the last constructed solution and starts the improvement algorithm with this solution. A number of repetitions of the improvement algorithm produces a series of local optima.

The Path Relinking Procedure performs a schedule transformation process which changes the operation sequences of the last found high-quality solution stepwise. In each step, a new and feasible schedule is created. The improvement algorithm uses this new schedule as initial schedule for the application of the local search procedure. Again, a series of local optima is generated.

Both adaptive components run a preset number of iterations. Afterwards, the solution procedure starts the construction algorithm again to generate a new sched-

ule. After the consumption of all available iterations (or a preset amount of time is expired), the solution procedure returns the best found solution and terminates.

The idea of dividing the search process into learning phase and main search phase is to identify the best configuration of the solution procedure. The application of the best performing dispatching rules shall navigate the search towards the best solutions in the search space. The implementation of the Amplifying Procedure and the Path Relinking Procedure aim at intensifying the search in promising regions of the search space.

Since the described algorithmic concept of the new heuristic exceeds the original concept of GRASP (corresponds to the gray-colored components in Fig. 6.4), the proposed solution procedure is referred to as an extended variant of GRASP, abbreviated EGRASP in the following. An executable file of this solution procedure is available from http://prodlog.wiwi.uni-halle.de/research_data/.

In the subsequent subsections, detailed information is given for the main components of the new heuristic.

6.2.2 Construction Algorithm

The aim of the construction algorithm is to generate a schedule that is feasible and already of good quality. The complete algorithmic process is shown as pseudocode in Algorithm 2.

The famous algorithm of Giffler and Thompson [49] serves as a basis for the design of the construction algorithm. The idea of the Giffler and Thompson algorithm is to build a schedule by consecutively inserting operations into a partial schedule. More precisely, the schedule construction process is divided into $n \cdot m$ steps with n and m as the number of jobs and machines respectively. In every step, exactly one of the unscheduled operations is chosen and inserted in the partial schedule. For this purpose, the set of schedulable operations is determined. An operation is called schedulable if the preceding operations according to its technological sequence are already scheduled. The set of schedulable operations is reduced by only considering operations which are processed on the critical machine. In each step, the machine is called critical machine which leads to the minimum completion time, if one of the schedulable operations is processed next on it. The resulting minimum completion times is also referred to as time bound TB. Moreover, an operation is also discarded from the set of schedulable operations if its earliest possible starting time exceeds

Algorithm 2 Construction Algorithm

1: Initialization: Empty schedule x
2: **while** Unscheduled operations exist **do**
3: Identify all schedulable operations
4: Reduce the set based on time bound TB^b
5: Order the operations in a Restricted Candidate List (RCL) with the help of a Dispatching Rule
6: Assign position-based probability values $1/r$
7: Select one operation randomly and insert it in schedule x
8: **end while**
9: Feasible, active schedule x

TB. As a result, the reduction yields the so-called critical set of schedulable operations. Thus, this critical set contains schedulable operations which have to be scheduled on the critical machine and their earliest possible starting time does not exceed the minimum completion time resulting from one of these operations.

The algorithm of Giffler and Thompson [49] yields a feasible schedule since the consideration of schedulable operations does not violate any technological sequence. Furthermore, the reduction to the critical set of schedulable operations definitely produces an active schedule. The restriction via TB ensures that the resulting idle times of the machines are very small in the constructed schedule so that no operation can start earlier without delaying the start of another operation.

In [125], Storer et al. have proposed the Hybrid Scheduling Algorithm which represents a generalization of the algorithm of Giffler and Thompson [49]. Their idea is to introduce a parameter for reducing the critical set of schedulable operations further.

In the critical set, each operation has an earliest possible starting time which is lying in a time interval spanned by the earliest possible starting time among these operations (left bound) and the earliest possible completion time among these operations (right bound). The left bound of the time interval is denoted by TB^0, the right bound is denoted by TB^1. The look-ahead parameter $\alpha \in [0, 1]$ reduces the right bound (in the following it is denoted by TB^α) so that the valid time interval decreases. As a result, some of the schedulable operations are discarded in the critical set if their earliest possible starting time is greater than TB^α.

The motivation of the look-ahead parameter α is to limit the allowance for a machine to stay idle. Using $\alpha = 0$ leads to the result that TB^α is identical to the

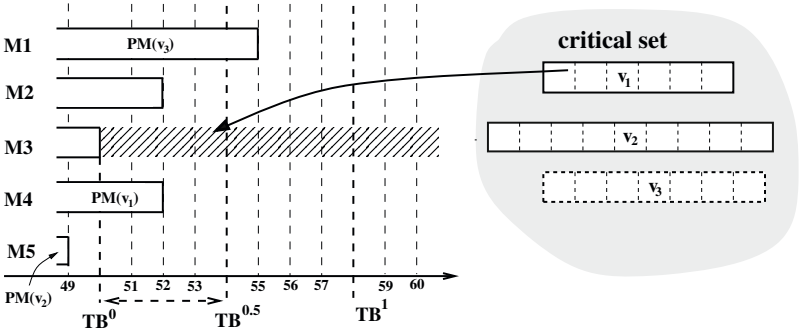


Fig. 6.5: Construction scheme for inserting the next unscheduled operation.

left bound of the time interval. Only TB^0 as earliest possible starting time is valid. With this setting, the construction algorithm only produces non-delay schedules. On the other hand, setting $\alpha = 1$ leads to the original Giffler and Thompson algorithm. Its use leads to the highest flexibility so that the construction algorithm can produce the totality of possible, active schedules.

The principle of the schedule generation process and the functionality of the look-ahead parameter α is illustrated in Fig. 6.5. In the depicted example, the schedule generation process has produced a partial schedule. In the considered step, the critical machine shall be M3 and three schedulable operations can be processed next on it. The schedulable operations in the critical set are denoted by v_1, v_2, v_3. For each of the three operations, its predecessor according to the technological sequence is already scheduled. The earliest possible starting time among the three operations, and thus, the value TB^0 arises from operation v_2 since the earliest possible starting time of v_2 is 50 (given by the completion time of the last scheduled operation on M3). The earliest possible completion time among the three operations is 58 which results from scheduling operation v_1 on M3. As a result, TB^1 is set to 58. Using $\alpha = 1$, all three operations can be inserted in this step of the construction algorithm. The value $\alpha = 0.5$ leads to a reduction of the valid time interval (see Fig. 6.5). Only operations can be inserted which have an earliest possible starting time smaller than $TB^{0.5} = 54$. In this example, only operations v_1 and v_2 are taken into account.

The schedule generation process continues with the ordering of the resulting critical set. The considered operations are placed in a list called the Restricted Can-

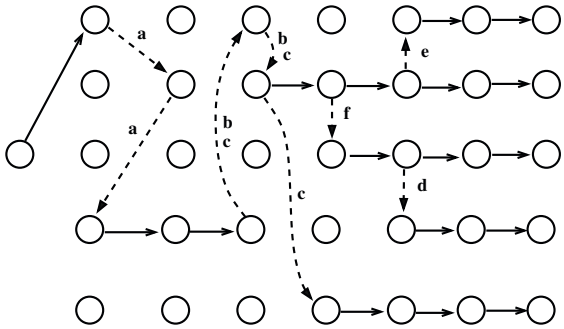
didate List (RCL). A dispatching rule which calculates a priority value for each operation is used for ordering the operations. In the sorted RCL, every operation gets assigned a selection probability based on its position in the list. Equivalent to the GRASP algorithm for solving the minimum makespan JSP [17], the probability values are calculated by a linear bias rule, assigning operation $v_{(r)}$ on position r the value $1/r$. The next operation to be inserted is chosen according to probabilistic sampling. Therefore, the schedule construction process is a stochastic process, delivering different schedules if it is applied multiple times.

The used dispatching rule affects the solution quality of the constructed schedule. The application of different rules yields different schedules. Therefore, the selected rule determines the region of the search space where the search process is started next. Obviously every dispatching rule proposed in the literature is applicable for ordering the RCL. The exclusive use of one rule does not have to be successful since there is no guarantee that it leads to discovering the best region of the search space. On the other hand, the procedure of EGRASP is flexible so that the dispatching rule can be changed before the next application of the construction algorithm.

6.2.3 Improvement Algorithm

The aim of the improvement algorithm is to increase the schedule quality by performing a local search procedure. The design of this local search procedure corresponds to a conventional first descent walk algorithm. This kind of algorithm generates a trajectory of neighboring schedules of continuously improved solutions with regard to the total weighted tardiness measure. The following features of the local search procedure are described in detail: the generation sequence of neighboring schedules via the order of critical blocks, the set of neighborhood operators used for generating new schedules, and the calculation procedure for determining the objective function value of new schedules.

The order of critical blocks affects the sequence of newly generated schedules. Since already the first improving schedule replaces the current solution and then the local search is restarted, the search process depends on this first found improvement. In the local search procedure, the set of critical blocks are ordered using the LOPA sorting strategy (LOPA is used as abbreviation for "longest path"). The basic idea of the LOPA strategy is to prefer these critical blocks which are lying on several longest paths [19]. For this reason, the set of longest paths leading through a block is used



block i	$g(i)$
a	15
b	15
f	7
c	5
d	4
e	1

Fig. 6.6: Example of critical blocks in $\mathcal{CT}$.

for the determination of a number g which represents the criticality of the block. More precisely, the longest paths' attached job weights are summed up yielding g, i. e. for a block i:

$$g(i) = \sum_{j \in J: i \subset \mathcal{LP}(0,F_j)} w_j$$

The set of critical blocks is listed in descending order according to the number g. The local search procedure employs a neighborhood operator starting with the first listed block. Neighboring schedules are generated by applying the specific perturbation scheme to this block. This neighbor generation process is repeated with the subsequent blocks in this list as long as no improving schedule is obtained. If the local search procedure has found an improving solution, the neighbor generation process terminates. On the other hand, if there is no improving solution, the local search procedure has found a local optimum with respect to the used neighborhood.

Fig. 6.6 shows an example of a critical tree which contains six blocks (related to an instance of size 5x5). Furthermore, it is assumed that the five jobs have weight $w_j = j$. The right side of the figure shows the list of critical blocks in the order of the LOPA strategy. Following this list, the search process starts the neighbor generation process with block a.

The motivation of the LOPA sorting strategy is that the first found improvement should affect as many jobs as possible. Therefore, the number $g(i)$ is defined representing the criticality of a block i. A high value of $g(i)$ means that the operation sequence of the block is an originator of many tardinesses. By using the LOPA

strategy, the designed improvement process of the local search procedure is carried out efficiently. As a result, improvements found at the beginning of the local search procedure are typically perturbations in the first part of the directed graph G' (close to source 0). Improvement found at the end of the local search procedure usually represent perturbations in the last part of G' (close to the finishing nodes F_j).

By using the LOPA strategy, a repetition of the improvement algorithm yields the same local optimum again. Therefore, the number of repetitions is zero if the LOPA sorting strategy is preset. Another available sorting strategy is the random order of the critical blocks. Every repetition of the improvement algorithm produces a different order of the blocks. As a result, the first found improving schedule found in the neighbor generation process usually differs from the first found improving schedule when using the LOPA strategy. By using the random order of the critical blocks, a repetition of the improvement algorithm produces a series of different local optima. Therefore, the number of repetitions is a parameter of the improvement algorithm which has to be set appropriately.

The algorithmic design is flexible to use any kind of neighborhood operator. Moreover, a set of neighborhood operators can be used together in order to improve the schedule quality. The investigations and results from Chapter 4 about different neighborhoods and their features enables to determine a good selection of neighborhood operators to be implemented.

The calculation of the objective function value of a new schedule is done using the Heads Updating procedure introduced in Chapter 5.

6.2.4 Adaptive Components

Incorporating learning and corresponding learning procedures in the search process are important parts of EGRASP, because they allow to adapt the performed search to the particular properties observed for a considered instance. Therefore, learning components are also referred to as adaptive components.

In EGRASP, several opportunities are used for incorporating adaptive components. The first of three components is the Configuration & Parameter Adjustment Procedure. The aim of this procedure is to analyze the results obtained in the learning phase of EGRASP and to determine the best configuration and parameter values for the remaining main search phase (see Fig. 6.4). The second adaptive component is the Amplifying Procedure. This procedure aims at intensifying the search in the

current region of the search space if a solution of high quality has been found. The third adaptive component is the Path Relinking Procedure. Path relinking aims at diversifying as well as intensifying the search process beneficially.

Configuration & Parameter Adjustment Procedure. The first purpose of the Configuration & Parameter Adjustment Procedure is to identify the best performing dispatching rules among the implemented rules for solving a particular instance. Remember that a dispatching rule is used in the construction algorithm to order the critical set of schedulable operations. Basically, four of these rules shall be used exclusively in the schedule generation process of the construction algorithm started in the main search phase. The selection bases on the obtained solution quality in the learning phase. An average objective function value of the total weighted tardiness measure is computed for every rule. Furthermore, the best obtained value of each rule is recorded in the learning phase. Based on these two statistics, the four best performing dispatching rules are identified. More precisely, the construction algorithm applied in the main search phase only uses the two best rules according to the average value and the two best rules according to the best obtained value. Note that the rules are really different from each other, i. e. if both statistics produce the same result the third, resp. the fourth best rule is taken into consideration.

The second purpose of the Configuration & Parameter Adjustment Procedure is to compute the scaling parameter β. The total weighted tardiness of the best found solution in the learning phase is involved in the computation for obtaining an adequate parameter value. As aforementioned, the aim of this parameter is to provide a separation of schedules due to the achieved solution quality. After identifying a solution of high quality, the search process continues with the Amplifying Procedure (see Fig. 6.4).

Amplifying Procedure. The identification of a high-quality solutions leads to the application of the Amplifying Procedure. The basic principle of this procedure is to repeat the application of the improvement algorithm with the last constructed start solution as initial solution, i. e. the solution which has lead to the new found high-quality solution. The improvement algorithm uses the random order as sorting strategy of the critical blocks. In summary, the Amplifying Procedure runs a fixed number of repetitions $\mathcal{I}_{AP}$ of the improvement algorithm producing a series of $\mathcal{I}_{AP}$ local optima.

The motivation of the Amplifying Procedure arises from the assumption that the identification of a high-quality solution has guided the search into a promising region of the search space. Performing the Amplifying Procedure intensifies the search into this region and potentially leads to an even better solution as the last high-quality solution.

Path Relinking Procedure. The concept of path relinking was developed by Laguna and Marti [81]. Basically, the idea is to perform a schedule transformation process which is applied to a candidate solution. For this purpose, the main attributes of the best solution found during the entire search process[14] are identified and are transferred into the candidate solution. Such main attributes can be e. g. specific precedence relations, operation sequences or fixed starting times of operations. The schedule transformation process applied to the candidate solution produces several intermediate solutions. By starting a local search procedure from each intermediate solution, the search process is intensified along the built trajectory of newly generated solutions.

Fig. 6.7 illustrates the described idea in which the schedule transformation process applied to a candidate solution leads to the intermediate solution x_1. After that, the process delivers the intermediate solution x_2 etc. Note that different approaches for transferring the attributes or the consideration of different types of attributes can produce different intermediate solutions. The dashed lines represent alternative ways on which path relinking can be performed.

In EGRASP, the Path Relinking Procedure is designed in the following way. The basic principle is to identify the operation sequences on the machines observed in the best found solution so far, to rebuild them in the considered candidate solution (schedule transformation process) and to apply the described improvement algorithm to intermediate solutions. As candidate solution serves the last generated local optimum of high quality.

In EGRASP, the schedule transformation process of path relinking is divided into m steps, with m as the number of machines. In each step, the schedule transformation is focused on the operation sequence of one selected machine, denoted by $m^\star$ in the following. The operation sequence on machine $m^\star$ of the candidate solution is changed systematically with the aim to obtain the operation sequence on $m^\star$ of the best found solution. Afterwards, the resulting intermediate solution serves as start

[14] Alternatively, path relinking works with a set consisting of the best solutions [3].

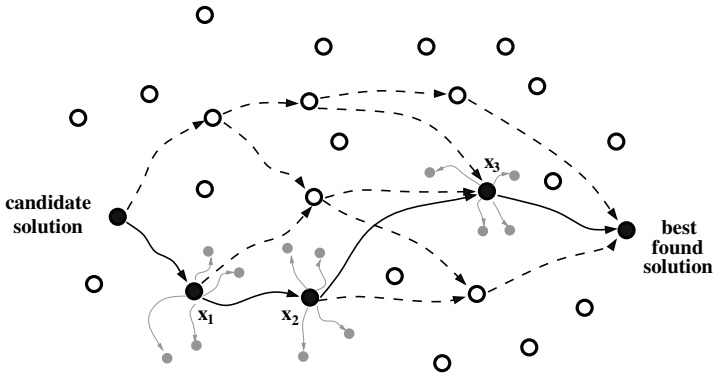


Fig. 6.7: General concept of path pelinking in the search space ($x_i \mathrel{\widehat{=}}$ intermediate solution, gray components $\mathrel{\widehat{=}}$ local search).

solution for the application of the improvement algorithm. The improvement algorithm is started with the random sorting strategy so that a number of repetitions produces a series of local optima. After the application of the improvement algorithm, the Path Relinking Procedure continues with the next step of the schedule transformation process, changing the operation sequence on the next machine.

The order of machines which are consecutively focused in the schedule transformation process is implicitly given by the LOPA sorting strategy. This means that the transformation process starts with the operation sequence of the machine that contains the first critical block according to the LOPA order. The order of machines follows the descending order given by the LOPA sorting strategy.

Focusing on the schedule transformation process in a particular step, the corresponding operation sequence in the candidate solution is changed in the following way. Basically, each position in the operation sequence is compared with the same position in the operation sequence of the best found solution, beginning with the first one. In doing this, two operations are compared with each other: operation v_{cs} on position p_{cs} in the candidate solution and operation v_{bfs} on the same position but in the best found solution. If v_{cs} and v_{bfs} are identical, the transformation process continues with the next position. But if the two operations are different to each other, a possible exchange of operations is investigated. For this purpose, the position of operation v_{bfs} in the candidate solution is identified, i. e. $p_{cs}(v_{bfs})$. The exchange of v_{cs} and v_{bfs} in the candidate solution sets v_{cs} on position $p_{cs}(v_{bfs})$

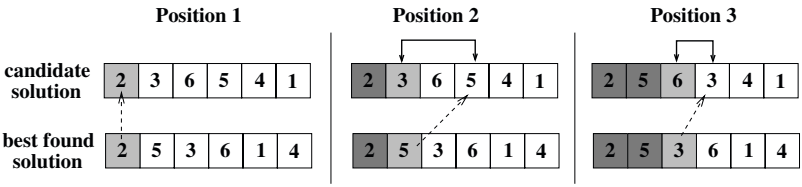


Fig. 6.8: Process of changing an operation sequence, related to the first three positions.

and v_{bfs} on the initially considered position $p_{cs}(v_{cs})$. The process continues with the next position and the corresponding operation.

In Fig. 6.8, the process is demonstrated with the help of a small example. The sequence of six operations is stored in an array. Since the first position in both sequences (current candidate solution and best found solution) contains the operation 2, there is no need to perform an exchange. In the second position, the operations in the candidate solution and in the best found solution are different. Therefore, the position of $v_{bfs} = 5$ has to be identified in the candidate solution. This operation is scheduled on the fourth position. The exchange of operations 3 and 5 on the positions 2 and 4 respectively shall be performed in this example. In the third position, the exchange of operations 3 and 6 is considered. The process continues with the investigation of swapping 3 and 6.

Basically, the exchange of two operations can yield an infeasible schedule. The new positions of the two operations, and thus, the new precedence relations among the operations, can lead to a cycle in the corresponding graph G' representing this schedule. For this reason, the exchange is only performed if the schedule feasibility is ensured. For doing this, two conditions have been developed which provide a check of the schedule feasibility. The conditions are based on the level values of the two operations, already used for computing a lower bound (see Chapter 5). The comparison of the level values enables to preclude parallel paths. Such a parallel path closes the cycle in the new schedule after performing the exchange. Therefore, the basic idea of the two following conditions is to preclude parallel paths between the swapped operations. In the following, it is assumed that two operations v_1 and v_2 shall be exchanged where v_1 precedes v_2 in the original schedule. The schedule obtained after swapping v_1 and v_2 is feasible if

$$l(v_1) \geq l(v_2) - 2 \tag{6.1}$$

or

$$l(v_1) = l(v_2) - 3 \wedge l(SM(v_1)) > l(PM(v_2)) - 1 \tag{6.2}$$

holds. Since $l(v_1) < l(v_2)$ holds in the original schedule, condition (6.1) checks if the deviation between the level values does not exceed a certain number. The condition (6.2) enhances the consideration by focusing on the succeeding operation of v_1 and the preceding operation of v_2 which are lying on such a potential parallel path.

Each of the two conditions is sufficient that the existence of parallel paths are precluded. Therefore, only one of the conditions (6.1) and (6.2) has to be fulfilled for performing the exchange. If both conditions are not fulfilled, the exchange of the two operations is rejected. The schedule transformation process continues with checking the next position in the operation sequence. As a result, the entire process does not have to reproduce the operation sequence observed in the best found solution.

6.2.5 Configuration and Parameter Values

In the algorithmic concept of the EGRASP, several components and parameters have to be specified in order to control the search process.

Selection of neighborhood operators to be implemented: Due to the need of an efficient and fast improvement algorithm, the local search procedure incorporates four neighborhood operators for generating new schedules: CET+2MT, CET, BCEI+2MT and SCEI (see Chapter 4). The main reason of this selection of operators is the outcome of Experiment 2. There, the above operators achieved the best absolute performance quality (i. e. the potential of the neighborhood in the local search with respect of the achieved effort) among a set of different operators. Furthermore, the four neighborhoods belong to the group of best performing or medium performing operators by means of the total performance quality (i. e. the full potential of neighborhoods in the complete local search algorithm). Combining two of these neighborhoods yields very good results again. They have shown an overall good trade-off between performance, feasibility and size (see Tab. 4.9).

The four neighborhoods are used alternately, i. e. after an improving solution is detected using one neighborhood, the next iteration is started with the next of the four neighborhood operators. The order of using the neighborhoods is derived

from the ranks achieved in Experiment 2: CET+2MT, CET, BCEI+2MT, SCEI, CET+2MT,... The improvement algorithm terminates if there is no further improvement possible with regard to any of the neighborhoods.

Note that using multiple neighborhoods goes beyond the original idea of GRASP. Actually, the applied strategy is adopted from the variable neighborhood search metaheuristic [99]. For this reason, one might classify EGRASP as a hybrid metaheuristic.

Selection of dispatching rules to be implemented: Dispatching rules are applied in the construction algorithm in order to sort the operations in the critical set. The use of only one rule does not have to result in a successful search process. The implementation of different rules leads to different schedules so that the search process is guided to different regions of the search space. However, using dispatching rules which are not capable to lead to a good solution quality for any problem instance should be omitted.

In order to separate good performing dispatching rules from bad performing rules, the following experiment aims at testing the solution quality of 24 dispatching rules. The 24 rules are listed in Tab. 6.3. In this set, due date related rules (e. g. Apparent Tardiness Cost rule [139]), processing time related rules (e. g. Shortest Processing Time rule), weighted rules (e. g. Weighted Shortest Processing Time rule [122]) and combined rules (e. g. Rule combining Processing Time and Waiting Time [66]) are included. Further information about the rules, e. g. the calculation scheme for determining the priority values, is provided in Appendix B. The experiment is carried out on a test bed consisting of 150 problem instances. The test bed contains the 40 la-instances and the ten orb-instances, each with three different due date factors $f = \{1.3, 1.5, 1.6\}$. Based on the ratio of jobs to machines $n:m$, the test bed consists of different classes of instances: $1:1$, $3:2$, $2:1$, $3:1$ and $4:1$.

Experiment 9: For each problem instance $i = 1, \ldots 150$, every dispatching rule $j = 1, \ldots 24$ is used for generating 1.000 schedules by applying the construction algorithm. Afterwards, the application of the improvement algorithm to each constructed solution produces a local optimum. As a result, 24.000 local optima have been computed for each problem instance i.

The assessment of the performance quality of each dispatching rule is based on two criteria: the best objective function value (bv) and the resulting average objective

Abbrev.	Name
ATC	Apparent Tardiness Cost rule
COVERT	Cost Over Time rule
DD+PT+WT	Rule combining Due Date, Processing Time and Waiting Time
ECT	Earliest Completion Time rule
EDD	Earliest Due Date rule
FCFS	First Come First Served rule
LPT	Longest Processing Time rule
MDD	Modified Due Date rule
ODD	Operational Due Date rule
PT+PW	Rule combining Processing Time and Waiting Time
PT+PW+ODD	Rule combining Processing Time, Waiting Time and Operational Due Date
PT+WINQ+SL	Rule combining Processing Time, Work In Next Queue and Slack rule
SL	Slack rule
S/OPN	Slack per number of operations rule
SPT	Shortest Processing Time rule
SPT+S/RPT	Rule combining SPT and Slack per Remaining Processing Time
WCR	Weighted Critical Ratio rule
WEDD	Weighted Earliest Due Date rule
WI	Weight rule
WINQ	Work In Next Queue rule
WMDD	Weighted Modified Due Date rule
WRA	Weighted Remaining Allowance rule
WSL+WSPT	Rule combining Weighted Slack rule and Weighted Shortest Processing Time
WSPT	Weighted Shortest Processing Time rule

Tab. 6.3: List of tested dispatching rules.

function value (av) according to the 1.000 computed local optima. For each of the two criteria, the rules are ranked for every instance $i = 1, \ldots 150$: the best performing rule gets rank 1; the rule which produces the worst result receives rank 24. Based on these data, the average ranks are computed for each of the two criteria. Tab. 6.4 shows the average ranks of the dispatching rules according to all 150 problem instances as well as for instances of the same instance class.

The results in Tab. 6.4 show that the dispatching rules are quite consistently ranked in the test bed of 150 problem instances. For the complete test bed, the best performing rule based on the criterion of the achieved average objective function value (av) is the Weighted Earliest Due Date rule (WEDD) with an average rank of 2.8. According to the best obtained value (bv) the Weighted Shortest Processing Time rule (WSPT) performs best. On the other hand, the Longest Processing Time rule (LPT) delivers the worst results indicated by the average ranks 23.8 and 23.4 for the two criteria.

For some of the rules, it can be observed that the performance quality varies among the different instance classes. The WEDD rule produces the best average

			instance classes									
	all		1:1		3:2		2:1		3:1		4:1	
Dispatching rule	av	bv	av	bv	av	bv	av	bv	av	bv	av	bv
ATC	3.4	4.6	2.0	6.6	2.3	4.5	4.0	4.1	4.1	4.0	3.8	3.9
COVERT	9.5	7.4	13.0	11.9	9.2	7.2	10.2	7.0	7.5	5.8	6.6	4.4
DD+PT+WT	13.9	11.9	15.0	12.9	13.1	12.1	14.1	9.5	12.4	11.8	15.7	14.8
ECT	11.0	11.7	8.1	7.8	11.3	10.6	10.4	10.3	12.5	14.7	13.3	16.1
EDD	11.3	12.8	14.3	13.0	12.4	13.1	10.6	13.6	10.4	11.6	8.7	13.1
FCFS	18.5	18.4	15.6	14.9	15.3	18.1	19.9	17.4	19.1	20.9	22.4	21.8
LPT	23.8	23.4	24.0	23.8	24.0	23.3	24.0	22.7	23.3	23.6	23.8	23.8
MDD	11.2	12.8	14.2	12.7	11.9	11.6	10.6	13.7	10.1	12.7	9.0	12.8
ODD	16.7	18.5	8.2	11.6	14.9	18.6	18.2	18.7	20.5	21.9	21.5	21.9
PT+PW	17.9	13.5	19.2	12.4	18.1	14.1	18.0	14.4	17.8	14.7	15.4	10.6
PT+PW+ODD	15.3	12.3	11.6	8.8	16.2	13.2	15.4	12.3	18.0	15.7	14.9	9.9
PT+WINQ+SL	18.3	16.4	17.5	14.5	18.7	15.3	18.4	16.8	18.0	16.4	19.5	19.4
SL	17.2	14.4	16.9	11.8	18.7	15.3	16.9	15.4	18.2	16.5	15.1	11.6
S/OPN	18.5	16.3	15.6	12.7	19.1	17.7	19.5	17.2	19.7	18.1	18.1	15.5
SPT	12.0	11.4	9.0	7.6	12.3	11.6	12.4	10.4	13.1	14.1	13.4	14.1
SPT+S/RPT	11.9	11.1	7.4	6.5	11.7	12.5	12.9	11.1	14.3	13.2	12.7	13.4
WCR	16.6	16.8	19.8	20.1	19.1	18.2	14.1	15.2	15.2	15.5	16.3	15.9
WEDD	2.8	5.3	5.8	9.8	3.7	6.7	2.7	5.7	1.0	2.0	1.0	2.3
WI	8.8	10.8	14.3	15.1	8.5	11.7	9.1	9.9	5.5	7.7	6.7	10.9
WINQ	21.0	18.7	22.3	19.3	21.0	20.1	21.2	17.1	19.7	17.9	21.5	21.4
WMDD	3.6	5.4	6.0	10.2	3.7	6.7	3.0	4.6	2.6	2.9	2.7	2.9
WRA	7.1	8.4	6.7	10.9	7.1	7.1	5.2	6.4	8.0	8.1	10.3	10.7
WSL+WSPT	6.4	7.2	9.9	9.8	5.4	6.7	5.4	7.5	5.9	5.9	4.7	5.4
WSPT	3.4	4.5	3.9	7.9	2.2	3.7	3.9	3.6	3.3	3.8	2.9	3.2

Tab. 6.4: Average ranks of the dispatching rules.

solution quality for problem instances with a job-machine-ratio of 3:1 or 4:1. For problem instance with job-machine-ratio 1:1, the average rank worsens significantly to the value 5.8. Another example is provided by the Operational Due Date rule (ODD). This rule performs well for some instances in the class 1:1 producing an average rank of 8.2. But the rule fails dealing with instances from the class 3:1 or 4:1.

Since EGRASP should perform well for a broad spectrum of different problem instances, rules are withdrawn if they fail significantly for all problem instances. For this purpose, Tab. 6.5 shows the transfer of the average ranks into a hierarchy of the rules. Position 1 gets the rule which delivers the best average rank for the considered criterion. The other positions are derived in a ascending order of the computed average ranks. Dispatching rules which are placed in the top ten of at least one instance class and one criterion are selected to be implemented in EGRASP. On the other hand, rules are withdrawn, which are not in any top ten position. This

	all		1:1		3:2		2:1		3:1		4:1	
Dispatching rule	av	bv	av	bv	av	bv	av	bv	av	bv	av	bv
ATC	2	2	1	2	2	2	4	2	4	4	4	4
COVERT	8	6	12	13	8	7	8	6	7	5	6	5
DD+PT+WT	14	12	16	17	14	12	14	8	11	10	17	16
ECT	9	11	7	4	9	8	9	10	12	15	12	19
EDD	11	14	14	18	13	14	10	14	10	9	8	13
FCFS	22	21	18	20	16	20	22	22	20	22	23	22
LPT	24	24	24	24	24	24	24	24	24	24	24	24
MDD	10	14	13	15	11	9	11	15	9	11	9	12
ODD	17	22	8	11	15	22	19	23	23	23	21	23
PT+PW	19	16	21	14	18	16	18	16	16	14	16	8
PT+PW+ODD	15	13	11	6	17	15	16	13	17	17	14	7
PT+WINQ+SL	20	19	20	19	19	18	20	19	17	18	20	20
SL	18	17	19	12	19	17	17	18	19	19	15	11
S/OPN	21	18	17	15	21	19	21	21	21	21	19	17
SPT	13	10	9	3	12	9	12	11	13	13	13	15
SPT+S/RPT	12	9	6	1	10	13	13	12	14	12	11	14
WCR	16	20	22	23	21	21	15	17	15	16	18	18
WEDD	1	3	3	7	3	3	1	4	1	1	1	1
WI	7	8	14	21	7	11	7	9	5	7	7	10
WINQ	23	23	23	22	23	23	23	20	21	20	21	21
WMDD	4	4	4	9	3	5	2	3	2	2	2	2
WRA	6	7	5	10	6	6	5	5	8	8	10	9
WSL+WSPT	5	5	10	7	5	3	6	7	6	6	5	6
WSPT	3	1	2	5	1	1	3	1	3	3	3	3

Tab. 6.5: Hierarchy of the dispatching rules.

decision results in 17 dispatching rules to be implemented in EGRASP:

ATC	COVERT	DD+PT+WT	ECT	EDD
MDD	ODD	PT+PW	PT+PW+ODD	SPT
SPT+S/RPT	WEDD	WI	WMDD	WRA
WSL+WSPT	WSPT			

Number of iterations in the learning phase $\mathcal{I}_{LP}$: The parameter indicates the number of computed local optima for each dispatching rule in order to estimate the quality of the respective regions in the search space. With a low value of $\mathcal{I}_{LP}$, the estimation is very rough and the selection of dispatching rules is done more or less randomly. If the value $\mathcal{I}_{LP}$ is very high, the estimation is more precise, but the learning phase consumes a lot of computational resources.

For this reason, the following experiment provides a test of five values $\mathcal{I}_{LP}$ with the aim to identify the lowest value among the five so that the selection of dispatching rules is stable. A stable selection means that an increase of $\mathcal{I}_{LP}$ almost delivers the same selection of rules.

In the experiment, the test bed consists of 52 problem instances of size 10 x 10. These instances are taken from the benchmark set of Singer and Pinedo [121]. In order to obtain a reliable assessment, only those instances are used which have never been solved with a total weighted tardiness of zero so far.

Experiment 10: The following five values are tested as number of iterations $\mathcal{I}_{LP}$: 500, 1.000, 1.500, 2.000 and 2.500. Each value is assessed with the help of EGRASP started once for every problem instance $i = 1, \ldots 52$. More precisely, EGRASP starts with running the learning phase. Its outcome is assessed by the Configuration & Parameter Adjustment Procedure. Afterwards, the run of EGRASP is terminated. As a result of each run, the Configuration & Parameter Adjustment Procedure has delivered the four best performing dispatching rules. Considering a problem instance $i = 1, \ldots 52$ and a value of $\mathcal{I}_{LP}$, the rule selection is given as $R_{1,i,\mathcal{I}_{LP}}, \ldots R_{4,i,\mathcal{I}_{LP}}$.

For the following assessment, the number $\delta_{k,i,\mathcal{I}_{LP}}$ is defined indicating the recurrence of the rule selection if $\mathcal{I}_{LP}$ is increased to the subsequent higher value. For the outcome of $\mathcal{I}_{LP} = 500$, this number is set as follows:

$$\delta_{k,i,500} = \begin{cases} 1, \text{ if } R_{k,i,500} = R_{k,i,1000} \\ 0, \text{ otherwise.} \end{cases} \quad \forall\ k = 1, \ldots 4,\ \forall\ i = 1, \ldots 52$$

Note that the values $\delta_{k,i,2500}$ cannot be computed since $\mathcal{I}_{LP} = 2.500$ is already the highest value.

The number $\bar{\delta}_{\mathcal{I}_{LP}}$ indicates the average recurrence of a rule selection for the values $\mathcal{I}_{LP} = 500, 1.000, 1.500$ and 2.000, aggregated for the complete test bed:

$$\bar{\delta}_{\mathcal{I}_{LP}} = \frac{1}{208} \sum_{k=1}^{4} \sum_{i=1}^{52} \delta_{k,i,\mathcal{I}_{LP}} \text{ [in \%]}$$

Furthermore, the concentration on the first and second best performing rule yields the number $\bar{\delta}^{bv}_{\mathcal{I}_{LP}}$ which measures the average recurrence of a rule selection based on their best obtained value (bv) in the learning phase (cf. Configuration & Parameter Adjustment Procedure in Section 6.2.4). Correspondingly, the number $\bar{\delta}^{av}_{\mathcal{I}_{LP}}$ indicates the average recurrence of a rule selection based on the average objective function value (av) observed in the learning phase, i. e. the third and fourth best performing rules.

$$\bar{\delta}^{bv}_{\mathcal{I}_{LP}} = \frac{1}{104} \sum_{k=1}^{2} \sum_{i=1}^{52} \delta_{k,i,\mathcal{I}_{LP}} \text{ [in \%]}, \qquad \bar{\delta}^{av}_{\mathcal{I}_{LP}} = \frac{1}{104} \sum_{k=3}^{4} \sum_{i=1}^{52} \delta_{k,i,\mathcal{I}_{LP}} \text{ [in \%]}$$

$\mathcal{I}_{LP}$	$\bar{\delta}_{\mathcal{I}_{LP}}$	$\bar{\delta}^{bv}_{\mathcal{I}_{LP}}$	$\bar{\delta}^{av}_{\mathcal{I}_{LP}}$
500	67.79%	52.88%	82.69%
1.000	79.81%	74.04%	85.58%
1.500	87.98%	83.65%	92.31%
2.000	81.73%	70.19%	93.27%

Tab. 6.6: Stabilitity of the selected dispatching rules.

All three introduced numbers are used to assess the stability of the rule selection given by the number of iterations $\mathcal{I}_{LP}$. If a value of a number is 100%, the increase of $\mathcal{I}_{LP}$ has produced the identical rule selection. Therefore, the numbers enable to identify an appropriate value $\mathcal{I}_{LP}$ so that a stable rule selection is obtained.

Tab. 6.6 shows the values of $\bar{\delta}_{\mathcal{I}_{LP}}$, $\bar{\delta}^{bv}_{\mathcal{I}_{LP}}$ and $\bar{\delta}^{av}_{\mathcal{I}_{LP}}$ computed for $\mathcal{I}_{LP} = 500$, 1.000, 1.500 and 2.000. Increasing $\mathcal{I}_{LP}$ stepwise from 500 to 1.500 produces an improvement of $\bar{\delta}_{\mathcal{I}_{LP}}$ from 68% to nearly 88%. Similar results are observed for the values of $\bar{\delta}^{bv}_{\mathcal{I}_{LP}}$ and $\bar{\delta}^{av}_{\mathcal{I}_{LP}}$. Performing 2.000 iterations provides almost the same rule selection like $\mathcal{I}_{LP} = 1.500$. A further increase of $\mathcal{I}_{LP}$ shows that the average recurrence of the rule selection $\bar{\delta}_{2000}$ decreases; only the rule selection based on average objective function value $\bar{\delta}^{av}_{2000}$ is constant. As a result, the value of $\mathcal{I}_{LP} = 1.500$ is fixed in EGRASP since it is the lowest value producing the highest stability in the rule selection.

Look-ahead parameter α and configuration of the main search phase: The usage of parameter α is to control the time bound TB^{α} in the construction algorithm. The aim is to reduce the critical set of schedulable operations so that only operations are taken into account which do not lead to a significant increase of machine idle times. As a result of the reduced critical set, the number of possible schedules generated by the construction algorithm decreases. Therefore, the parameter α implicitly restricts the region of the search space where the search process is started.

In EGRASP, two specific values of the parameter are used: $\alpha = 0.5$ and $\alpha = 1.0$. The value $\alpha = 0.5$ is chosen for an appropriate reduction of the critical set of schedulable operations. The value $\alpha = 1.0$ leads to the maximum diversification in the search process.

In the learning phase of EGRASP, $\alpha = 0.5$ is exclusively used in the construction algorithm. With this setting, it is expected to get reliable information about the performance quality of the implemented dispatching rules. In the main search phase, both values of α are used alternately. The idea of this alternation is to firstly keep the search process in the previous successful region of the search space by using

again $\alpha = 0.5$. Afterwards, by increasing α to 1.0, the search becomes more diverse since the respective region of the search space is expanded. The values of α are alternating so that the same amount of computational resources is consumed.

The above described alternation is linked with the following general setting of the main search phase. Remember that the main search phase starts after the Configuration & Parameter Adjustment Procedure. This procedure identifies the four best performing dispatching rules. For their further application within the main search phase, this phase is again divided into four phases, also referred to as sub-phases. In each sub-phase, one of the four selected dispatching rules is used exclusively in the construction algorithm. The first two sub-phases are executed with the best performing dispatching rules according to the best obtained value; in the third and fourth sub-phase, the rules based on the best average objective function value are chosen. The share between the four sub-phases corresponds to the ratio 2:1:2:1. Moreover, in every sub-phase the parameter α is used with the two values 0.5 and 1.0 in the ratio 1:1.

To illustrate this general setting of the main search phase, assume that the available computation time in the main search phase is 60 seconds. As a result, the first sub-phase runs 20 seconds. Here, α has the value 0.5 in the first ten seconds; in the second ten seconds, α is set to 1.0. The second sub-phase consumes a total of ten seconds at which each value of parameter α is used for five seconds. The following third sub-phase runs 20 seconds again etc.

Scaling parameter β: In the main search phase of EGRASP, high-quality solutions are identified so that adaptive components can be activated to intensify the search around the best found solutions of a problem instance. The evaluation whether or not a solution has a good quality is based on its objective function value (see Fig. 6.4). The best found solution of the learning phase provides an estimation for the total weighted tardiness value of the global optimum. Moreover, it is used for computing the scaling parameter β, which controls the range of high-quality solutions. The parameter is set as follows.

$$\beta = \begin{cases} 8 \cdot TWT(x_{best})^{-0.25} & \text{, if } TWT(x_{best}) \leq 1.975 \\ 1.2 & \text{, otherwise.} \end{cases}$$

The setting of parameter β is based on the following idea. For problem instances which exhibit total weighted tardiness values of more than 1.975, the scaling factor β is set to the value 1.2, indicating that a high-quality solution can be 20% worse than the best found solution so far. If the resulting total weighted tardiness is lower, the threshold of 20% does not work effectively. For example, if $TWT(x_{best}) = 5$ and $\beta = 1.2$, then adaptive components are only started for solutions with an objective less than 6. For this reason, the determination of the scaling factor β is a function of $TWT(x_{best})$. The lower the value of $TWT(x_{best})$, the higher the value of β. For $TWT(x_{best}) = 5$, the resulting scaling factor β is 5.35 which activates the Amplifying Procedure and the Path Relinking Procedure for solutions x with $TWT(x) < 27$.

In the calculation scheme of β, the monotonically decreasing function is set intuitively. The fixed value of 1.2 is obtained by a preliminary experiment. This experiment has been designed in a similar fashion like Experiment 10, i. e. testing several parameter values and assessing their outcome appropriately.

Number of iterations in Amplifying Procedure $\mathcal{I}_{AP}$: The Amplifying Procedure is an adaptive component for intensifying the search within a promising region. The number of iterations $\mathcal{I}_{AP}$ represents the number of local optima found and assessed in this area of the search space. In all computations, it is set to $n \cdot m$. This setting reflects the size of the search space depending on n and m. Consequently, the intensification process lasts longer for larger problem instances. Furthermore, experiments with different settings $\mathcal{I}_{AP}$ have shown that the value $\mathcal{I}_{AP} = n \cdot m$ yields a good trade-off between intensification phase and further search process.

Number of iterations in Path Relinking Procedure $\mathcal{I}_{PR}$: The Path Relinking Procedure is used for diversifying the search towards the best found solution so far, and intensifying the search along the built search trajectory. The number of iterations $\mathcal{I}_{PR}$ represents the number of computed local optima by starting from an intermediate solution. $\mathcal{I}_{PR}$ is set to n. Since the schedule transformation process is divided into m steps at which an intermediate solution is generated, the computational demand of the Path Relinking Procedure grows similar as in the Amplifying Procedure.

Termination Criterion: EGRASP terminates if a fixed number of iterations $\mathcal{I}$ is consumed or a predetermined time limit expires. The termination criterion depends on the considered experiment and is set by the user. Note that both termination

criteria cover the learning phase as well as the main search phase, i. e. if a fixed number of iterations is chosen and $\mathcal{I} < 17 \cdot \mathcal{I}_{LP}$ holds, EGRASP is already terminated in the learning phase.

Chapter 7

Computational Study

The development of the new solution procedure EGRASP arises from the combination of the basic GRASP and further search strategies which are recommended for solving job shop scheduling problems. In this chapter, the suitability of the new approach shall be validated.

This chapter provides a performance comparison between EGRASP and other methods proposed in literature. In the first section, the well-known benchmark set of Singer and Pinedo [121] is tackled which is also used by many other researchers (see e. g. [19, 76]). Moreover, the performance comparison is pursued by solving the instance set of Lawrence [85] which consists of problem instances of different size $n \mathrm{x} m$. In the second section, further computational tests are carried out on instance sets of other optimization problems. The aim is to evaluate the performance of EGRASP on scheduling problems closely related to the JSPTWT, e. g. the minimization of the number of tardy jobs for the job shop scheduling problem.

7.1 Benchmark Instances of the JSPTWT

7.1.1 Modification of JSP Instances

The test instances for the JSP contained in the OR-Library [13] typically consists of the technological machine sequence for the jobs and the processing time of every operation, while release times, due dates, and job weights are not specified. In order to transfer an instance of the standard JSP into an instance of the JSPTWT, Singer and Pinedo have proposed a way to supplement the missing data [121]. In this

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0_7,

approach, every job j receives the release time $r_j = 0$. With the help of the due date factor f, a due date d_j is computed for the job by:

$$d_j = \left\lfloor r_j + f \cdot \sum_{i=1}^{m} p_{ij} \right\rfloor .$$

Singer and Pinedo suggest using three different due date factors $f \in \{1.3, 1.5, 1.6\}$ so that differently strict types of due dates can be generated. $f = 1.3$ creates tight due dates, $f = 1.5$ moderate due dates, and $f = 1.6$ generates relaxed due dates. The job weights are set in the instances as follows: the first 20% of the jobs receive a weight of 4, the next 60% of jobs receive a weight of 2, and the final 20% of the jobs get a weight of 1. The job weight represents the importance of a job. The higher a job's weight, the important it is to complete the job on-time. This specific distribution of weights corresponds to the distribution of the varying importance of jobs observed in real-world experiences [111].

With this approach, the well-known JSP instances of Adams et al. [2] (abz05-abz09), Fisher and Thompson [46] (ft06, ft10, ft20), Lawrence [85] (la01-la40), and Applegate and Cook [7] (orb01-orb10), and Taillard [128] (tai01-tai80) are modified.

7.1.2 Standard Benchmark Set of Singer and Pinedo

Since Singer and Pinedo [121] first investigated the JSPTWT, the set of 66 instances that they have solved has become a standard benchmark suite. It is exclusively composed by 22 problem instances of size 10 x 10; namely abz05-abz06, ft10, la16-la24 and orb01-orb10. Note that the original size of the instances la21-la24 is 15 x 10. However, in the benchmark set of Singer and Pinedo, only the first ten jobs are considered. The three due dates factors f are applied to every original problem instance inducing the complete benchmark set of 66 instances.

The proposed benchmark is a widely used test suite for comparing new solution procedures for the JSPTWT. The currently best performing algorithms for this test suite are:

- EMD (2008): Genetic local search of Essafi, Mati and Dauzére-Pérès, 2008 [41].
- MDL (2011): Iterated local search of Mati, Dauzére-Pérès and Lahlou, 2011 [91].
- KB (2011): Shifting bottleneck-tabu search heuristic of Bülbül, 2011 [19].

- GGVV (2012): Hybrid evolutionary algorithm of González, González-Rodriguez, Vela and Varela, 2012 [57].

Tab. C.1-C.3 in Appendix C show the results obtained by these methods for each of the 66 instances. There are further solution procedures for which computational results of this benchmark set have been published, but the testing is either not complete (e. g. in [135]) or the allowed computation time are incompatible (e. g. [154]). Therefore, these approaches are not taken in the following analysis.

To make the computational results obtained for the various problem instances transparent, Bülbül [19] have proposed the performance measures TotalGap and # BKS. TotalGap measures the relative error of the obtained results with respect to the sum of the best known solutions:

$$\text{TotalGap} = \frac{\sum TWT(BFS_k) - \sum TWT(BKS_k)}{\sum TWT(BKS_k)} \text{ [in \%]}$$

Moreover, the number of best known solutions (# BKS) obtained by the considered solution procedure is provided. A value of # BKS = 66 indicates, for instance, a solution procedure has found all currently best known solutions.[15]

Comparison with best performing solution procedures. The first part of the computational study focuses on the comparison of EGRASP with the best performing solution procedures mentioned above. In order to obtain a fair comparison between previous results and results of EGRASP, the same amount of allowed computation time has to be given to every procedure. Following the approach presented in [91], the time limit of the computations is adjusted to the processing speed of the computer. The time limit given to EGRASP is derived from the computation time reported for EMD (2008). The genetic local search of EMD (2008) has exhibited the best performance results for a long time and subsequent research has taken it for benchmarking. It consumes a total of 180 seconds of run time for each problem instance. The 180 seconds are subdivided into 10 runs each consuming 18 seconds. The experiments of EMD (2008) are performed on a computer with 2.8 GHz [41]. Accordingly, the time limit of EGRASP running on a computer with 3.4 GHz is set to 148 seconds:

$$\frac{2.8 \text{ GHz}}{3.4 \text{ GHz}} \cdot 180 \text{ seconds} \approx 148 \text{ seconds}$$

[15] Originally, Bülbül have proposed a third performance measure called AverGap [19]. Since this measure cannot provide additional information, its usage is omitted in the study.

		(10/15)		(4/37)		(1/148)
Instance	BKS	mean	best	mean	best	best
abz05	1403	*	*	*	*	*
abz06	436	437.1	*	*	*	*
ft10	1363	1435.0	1435	*	*	*
la16	1169	*	*	*	*	*
la17	899	*	*	*	*	*
la18	929	*	*	*	*	*
la19	948	*	*	*	*	*
la20	805	833.7	*	*	*	*
la21	463	*	*	*	*	*
la22	1064	*	*	*	*	*
la23	835	*	*	*	*	*
la24	835	*	*	*	*	*
orb01	2568	2801.0	2801	2621	*	*
orb02	1408	1430.0	1429	1434.0	1434	*
orb03	2111	2299.0	2299	2173.8	2115	2115
orb04	1623	1692.6	1676	*	*	*
orb05	1593	1723.8	1667	1695.0	*	*
orb06	1790	1860.0	1860	1824.0	1798	*
orb07	590	614.0	614	610.0	608	*
orb08	2429	2603.0	2598	2444.0	*	*
orb09	1316	1422.0	1422	*	*	*
orb10	1679	1826.0	1826	1737.0	*	*
TotalGap		4.48	4.09	1.31	0.20	0.01
# BKS		9	11	14	18	21

Tab. 7.1: Computational results of EGRASP for the benchmark set of Singer and Pinedo ($f = 1.3$), after consuming 148 seconds computation time.

The following computations of the solution procedure EGRASP rely on three different settings distinguished by the number of runs and the computation time of a single run. In the first setting (10/15), the solution procedure is started 10 times with 15 seconds for each single run which is also used in previous computational studies (cf. e. g. [41, 91]). Two other settings (4/37) and (1/148) are tested as well, where the runs last longer, while the number of restarts decreases. In all settings, the same amount of computation time is consumed so that a reliable comparison is provided.

Tab. 7.1-7.3 show the objective function value of the best known solution (column "BKS"), the mean values of the computations for the settings (10/15) and (4/37), and the best objective function value obtained for each of the three settings. Compu-

		(10/15)		(4/37)		(1/148)
Instance	BKS	mean	best	mean	best	best
abz05	69	*	*	*	*	*
abz06	0	*	*	*	*	*
ft10	394	*	*	*	*	*
la16	166	*	*	*	*	*
la17	260	*	*	*	*	*
la18	34	*	*	*	*	*
la19	21	*	*	*	*	*
la20	0	*	*	*	*	*
la21	0	*	*	*	*	*
la22	196	*	*	*	*	*
la23	2	*	*	*	*	*
la24	82	88.0	88	83.5	*	*
orb01	1098	1221.0	1141	*	*	*
orb02	292	324.0	324	*	*	*
orb03	918	989.0	989	920.5	*	*
orb04	358	*	*	*	*	*
orb05	405	487.6	*	*	*	*
orb06	426	456.0	456	455.5	455	*
orb07	50	*	*	*	*	*
orb08	1023	1169.2	1158	1079.0	1054	1076
orb09	297	*	*	*	*	*
orb10	346	462.0	458	424.0	424	*
TotalGap		9.43	6.17	2.60	2.14	0.82
# BKS		14	15	17	19	21

Tab. 7.2: Computational results of EGRASP for the benchmark set of Singer and Pinedo ($f = 1.5$), after consuming 148 seconds computation time.

tational results reproducing the best known solution are asterisked. The assessment of the results in terms of TotalGap and # BKS is shown at the bottom of the tables.

In all the three tables, it becomes apparent that the performance of EGRASP strongly benefits from a longer computation time of a single run. With the exception of orb02 ($f = 1.3$), the obtained results in the setting (4/37) are at least as good as the results in the setting (10/15). Moreover, the results from the single run in setting (1/148) always dominate the results from the other settings (10/15) and (4/37). The observation, that the longer the single run, the better the performance quality of EGRASP is, is also verified by the decreasing TotalGap value as well as the increasing value of # BKS. Apparently, if the search process of a single run can last longer, EGRASP can find more high-quality solutions. As a result, the overall

		(10/15)		(4/37)		(1/148)
Instance	BKS	mean	best	mean	best	best
abz05	0	*	*	*	*	*
abz06	0	*	*	*	*	*
ft10	141	157.0	157	*	*	*
la16	0	*	*	*	*	*
la17	65	*	*	*	*	*
la18	0	*	*	*	*	*
la19	0	*	*	*	*	*
la20	0	*	*	*	*	*
la21	0	*	*	*	*	*
la22	0	*	*	*	*	*
la23	0	*	*	*	*	*
la24	0	*	*	*	*	*
orb01	566	713.4	696	651.0	651	*
orb02	44	52.0	52	*	*	*
orb03	422	533.0	533	533.0	533	*
orb04	66	88.0	88	*	*	*
orb05	163	183.4	181	167.3	*	*
orb06	28	36.6	31	31.0	31	*
orb07	0	*	*	*	*	*
orb08	621	691.0	691	674.5	651	646
orb09	66	91.2	*	67.5	*	*
orb10	76	132.8	100	94.0	84	84
TotalGap		21.50	17.80	12.23	10.50	1.46
# BKS		12	13	15	17	20

Tab. 7.3: Computational results of EGRASP for the benchmark set of Singer and Pinedo ($f = 1.6$), after consuming 148 seconds computation time.

best results are achieved under setting (1/148) which is capable to find 62 of 66 best known solutions.

The results to the abz-, ft- and la-instances shown in the first twelve rows of each Tab. 7.1-7.3 indicate that these instances are relatively easy to solve. Only for the setting (10/15), EGRASP produce some solutions worse than the best known solutions. In most cases, EGRASP delivers the best known solution in each run. The orb-instances are more difficult to solve independent of the used type of due dates. Only for four problem instances, EGRASP is capable to consistently compute the best known solution in each of the 10 runs of the setting (10/15).

In order to explain why EGRASP benefits from extending the computation time of single runs, the run time protocols are considered in detail. In these protocols,

Method	TotalGap	rank	# BKS	rank
EMD (2008)	0.27	3	60	3
MDL (2011)	1.16	4	59	4
KB (2011)	2.11	5	43	5
GGVV (2012)	0.00	1	66	1
EGRASP (1/148)	0.24	2	62	2

Tab. 7.4: Comparison of EGRASP with best performing solution procedures, solving the instances of Singer & Pinedo, based on the best solutions found by a procedure.

the objective function value of the best found solution is recorded as well as the component which has yielded this solution at first. Such a best solution can be found by a conventional GRASP iteration, the Amplifying Procedure or the Path Relinking Procedure. For doing this, recall that the main purpose of the Amplifying Procedure and the Path Relinking Procedure is to intensify the search in promising regions. The average number of best solutions found by the Amplifying Procedure increases from 4.1 in the setting (10/15) to 9.8 in the setting (4/37), and finally to 14 in the setting (1/148). The average number of best solutions found by the Path Relinking Procedure increases from 10.7 in the setting (10/15) to 20.3 in the setting (4/37). In the setting (1/148), 19 best found solutions are produced via path relinking. These results illustrate that the impact of the Amplifying Procedure and the Path Relinking Procedure grows with the increasing computation time given to a single run.

Tab. 7.4 presents a comparison of EGRASP and the best performing solution procedures based on the performance measures TotalGap and # BKS. The four best performing metaheuristics known from literature (see Appendix C) and EGRASP (1/148) are assessed and compared with each other. In order to obtain a reliable comparison, the TotalGap and # BKS values are calculated for the best solutions found by each of the procedures. Afterwards, a rank is derived for every performance measure and solution procedure. The results in Tab. 7.4 show that GGVV (2012) produces the best known solution for every of the 66 problem instances. For this reason, it receives rank 1. The second best solution procedure is EGRASP. Based on the two performance measures, the results are significantly better than those of EMD (2008) which takes rank 3. Rank 4 and 5 achieve the solution methods MDL (2011) and KB (2011), which produce TotalGap values of more than one percent.

Additional performance check of EGRASP (1/148). Since EGRASP contains stochastic search components, the variance of the results delivered by the setting (1/148) is tested by repeating the computations. In total, 25 runs of the setting (1/148) are assessed by means of the produced TotalGap and # BKS. In eleven runs, the computational results are even better than the first results presented in Tab. 7.1-7.3. On the other hand, in the same number of runs, EGRASP has delivered worse results. Consequently, EGRASP has produced the same TotalGap value in three of the 25 runs. In the 25 runs with setting (1/148), the TotalGap ranges within 0.08 to 0.77 with an average value of 0.27. In every run, # BKS is at least 60. The maximum value of # BKS is 64; the average value is 62.1. The analysis shows that the new solution procedure produces an excellent solution quality on a reliable basis. Hence, it deserves rank 2 in the competition, in particular with respect to the # BKS.

Comparison with recently published solution procedure. In 2013, Zhang et al. published a new hybrid metaheuristic for solving the JSPTWT [149] which combines an artificial bee colony algorithm with a local search procedure. The presented computational results provide another opportunity for a performance competition. In [149], the performance quality of the hybrid metaheuristic is assessed by solving the instances of the benchmark set of Singer and Pinedo [121]. It delivers better results than the genetic local search from Essafi et al. [41]. For this reason, this part of the computational study focuses on the comparison of EGRASP with ZSW (2013) and EMD (2008):

- ZSW (2013): Hybrid artificial bee colony algorithm of Zhang, Song and Wu, 2013 [149].
- EMD (2008): Genetic local search of Essafi, Mati and Dauzére-Pérès, 2008 [41].

In the computations of ZSW (2013), each solution procedure is started 10 times with every run lasting 30 seconds. As a result, a procedure consumes 300 seconds in total for a problem instance. In order to guarantee a fair comparison of the algorithms executed on different computer systems with 2.67 GHz and 3.4 GHz respectively, EGRASP is given a computation time limit of 236 seconds:

$$\frac{2.67 \text{ GHz}}{3.4 \text{ GHz}} \cdot 300 \text{ seconds} \approx 236 \text{ seconds}$$

		ZSW (2013)	EMD (2008)	EGRASP (10/24)	EGRASP (1/236)
Instance	BKS	mean	mean	mean	best
abz05	1403	1405	*	*	*
abz06	436	*	*	*	*
ft10	1363	1367	1372	*	*
la16	1169	1170	1175	*	*
la17	899	*	900	*	*
la18	929	930	932	*	*
la19	948	*	949	*	*
la20	805	810	*	*	*
la21	463	*	464	*	*
la22	1064	1071	1085	*	*
la23	835	846	859	*	*
la24	835	*	*	*	*
orb01	2568	2573	2614	2681	*
orb02	1408	1412	1431	1434	*
orb03	2111	2115	2164	2219	*
orb04	1623	1625	1640	*	*
orb05	1593	1599	1622	1738	1667
orb06	1790	1797	1792	1852	*
orb07	590	604	592	609	*
orb08	2429	2433	2486	2482	*
orb09	1316	1334	*	*	*
orb10	1679	1682	1708	*	*
TotalGap		0.35	1.15	1.86	0.26
# BKS		5	5	15	21

Tab. 7.5: Computational results of ZSW (2013), EMD (2008) and EGRASP for the benchmark set of Singer and Pinedo ($f = 1.3$), after consuming 236 seconds computation time.

For the following computational comparison, EGRASP is started with the two settings (10/24) and (1/236). In the first setting, 10 runs of EGRASP are executed each with a time limit of 24 seconds. In the setting (1/236), EGRASP is started once with 236 seconds of allowed computation time.

Tab. 7.5-7.7 present the results of ZSW (2013) and EMD (2008) reported in [149]. Furthermore, the mean values of the outcome of EGRASP under the setting (10/24) are shown in fifth column of the tables. Finally, the results of EGRASP with setting (1/236) are presented in the last column of the tables. Since the mean value corresponds to the best value in the setting (1/236), the last column refers to the computed best values. However, in all computations, the same amount of computational resources are consumed so that a reliable comparison is available again.

		ZSW (2013)	EMD (2008)	EGRASP (10/24)	EGRASP (1/236)
Instance	BKS	mean	mean	mean	best
abz05	69	*	*	*	*
abz06	0	*	*	*	*
ft10	394	*	*	*	*
la16	166	*	*	*	*
la17	260	*	*	*	*
la18	34	*	*	*	*
la19	21	*	*	*	*
la20	0	*	*	*	*
la21	0	*	*	*	*
la22	196	*	*	*	*
la23	2	*	*	*	*
la24	82	*	83	*	*
orb01	1098	1107	1120	*	*
orb02	292	*	*	*	*
orb03	918	925	932	959	*
orb04	358	359	381	*	*
orb05	405	409	*	481	*
orb06	426	*	438	456	*
orb07	50	55	52	*	*
orb08	1023	1030	1042	1159	1035
orb09	297	*	307	*	*
orb10	346	349	376	424	*
TotalGap		0.56	2.07	5.61	0.19
# BKS		15	13	17	21

Tab. 7.6: Computational results of ZSW (2013), EMD (2008) and EGRASP for the benchmark set of Singer and Pinedo ($f = 1.5$), after consuming 236 seconds computation time.

The computational results are assessed by the performance measures TotalGap and # BKS. The calculated values are shown at the bottom of the tables.

The results in Tab. 7.5-7.7 show that the average performance quality of EGRASP in the setting (10/24) is different to the performance quality of ZSW (2013) and EMD (2008). On the one hand, EGRASP produces the highest number of best known solutions. For all 66 problem instances, EGRASP achieves # BKS = 45. ZSW (2013) and EMD (2008) are only capable to find 34 and 31 best known solutions. On the other hand, the results obtained by EGRASP in the setting (10/24) exhibits a high variance which results in the highest TotalGap value in all three tables. Especially the performance quality observed by solving the orb-instances is rarely as good as the performance quality of ZSW (2013) and EMD (2008).

		ZSW (2013)	EMD (2008)	EGRASP (10/24)	EGRASP (1/236)
Instance	BKS	mean	mean	mean	best
abz05	0	*	*	*	*
abz06	0	*	*	*	*
ft10	141	149	160	154	*
la16	0	*	*	*	*
la17	65	*	*	*	*
la18	0	*	*	*	*
la19	0	*	*	*	*
la20	0	*	*	*	*
la21	0	*	*	*	*
la22	0	*	*	*	*
la23	0	*	*	*	*
la24	0	*	*	*	*
orb01	566	584	612	657	*
orb02	44	*	*	52	*
orb03	422	438	467	527	*
orb04	66	73	75	*	*
orb05	163	171	181	177	*
orb06	28	*	*	31	31
orb07	0	*	*	*	*
orb08	621	630	655	683	646
orb09	66	74	74	84	*
orb10	76	86	109	100	*
TotalGap		3.72	9.39	14.97	1.24
# BKS		14	14	13	20

Tab. 7.7: Computational results of ZSW (2013), EMD (2008) and EGRASP for the benchmark set of Singer and Pinedo ($f = 1.6$), after consuming 236 seconds computation time.

As noticed above, the performance quality obtained by EGRASP significantly increases if the computation time spent in a single run increases. This observation is also confirmed by the considered competition. The setting (1/236) leads to significantly better results than the setting (10/24). Moreover, the obtained computational results are clearly better than those of ZSW (2013) and EMD (2008). In all three tables, the performance measures TotalGap and # BKS receive their best values in the computations of EGRASP (1/236). Therefore, EGRASP provides the best solution quality among the three considered solution procedures.

Solving the problem instance orb08 ($f = 1.6$). The computational experiments have shown that the instance orb08 with due dates based on $f = 1.6$ is among the hardest problem instances for the new solution procedure. EGRASP has

never produced the best known solution with objective TWT = 621. Even if the allowed computation time has been scaled up to more than 10 minutes, the situation hardly changes. The overall best solution delivered by EGRASP has TWT = 646.

A more detailed analysis for this phenomenon is provided in Appendix D. Further investigations show that there are more than 20 different solutions which possess the best known objective function value of 621. These solutions are very close to each other in the search space composing a big valley structure. The center of this big valley and the overall best solution found by EGRASP have a Hamming distance of 57, indicating that 12.7% of the precedence relations are different. To bridge this distance, the local search procedure has to apply the CET neighborhood operator for at least 57 times. With regard to the average improving steps performed by the CET operator in a steepest descent algorithm (see Tab. 4.6), the improvement algorithm in EGRASP is not able to detect this search region.

Furthermore, the initial solutions generated by the construction algorithm are located in far away regions of the search space. The four dispatching rules which are identified by the Configuration & Parameter Adjustment Procedure as best performing rules are WSL+WSPT, PT+PW+ODD, EDD and WMDD. They yield to deterministic schedules[16] having a Hamming distance to this big valley of at least 100. Even though the construction algorithm samples initial solutions around the deterministic schedule, this distance remains too high. In order to eliminate this problem, a different dispatching rule could be used to create solutions in a closer region. However, the design of the solution procedure would discriminate against other problem instances. Further details and information concerning the EGRASP result for the problem instance orb08 ($f = 1.6$) are presented in Appendix D.

Summarizing, EGRASP shows to be a powerful alternative for solving the JSPTWT. With regard to the considered benchmark set, the produced computational results are of high quality. It turns out that EGRASP consistently benefits from a longer run time. In contrast to other methods, EGRASP can transfer computation time into progressing solutions for longer periods which, in turn, suspends the difficulty of premature convergence. As a result, it is not necessary to restart computations very often, since the procedure is able to systematically explore the search space at one stroke.

[16]For the construction of a deterministic schedule, the next operation to be inserted is always the one with highest priority.

7.1.3 Lawrence's Instances

Essafi et al. [41], Bülbül [19] and González et al. [57] have tested their solution procedures on the problem instances la01-la40, introduced by Lawrence [85]. Based on the three different types of due dates, the benchmark set consists of a total of 120 instances. In this set, the instance size varies from 50 to 300 operations. Beside "square" problem instances, also "rectangular" instances with a job-machine-ratio of 2:1, 3:1, 3:2, and 4:1 are included in the instance set. Together, eight different classes of instances are given in the set where the size $n \mathrm{x} m$ is constant in each class.

The benchmark set provides a computational comparison using various JSPTWT instances. Therefore, the following part of the computational study focuses on this benchmark set comparing EGRASP with the three solution procedures mentioned above.

- EMD (2008): Genetic local search of Essafi, Mati and Dauzére-Pérès, 2008 [41].
- KB (2011): Shifting bottleneck-tabu search heuristic of Bülbül, 2011 [19].
- GGVV (2012): Hybrid evolutionary algorithm of González, González-Rodriguez, Vela and Varela, 2012 [57].

In contrast to the previous computational tests, the termination criterion of EGRASP for solving these problem instances is based on a fixed number of iterations $\mathcal{I}$, i. e. the total number of generated local optima. EMD (2008) set up their genetic local search algorithm by fixing the number of generations, the size of the population etc. (see [41]). In average, their algorithm generates 520.000 local optima per instance and run, i. e. 624 million local optima are evaluated for the entire benchmark set. KB (2011) does not provide any specific information about the used computational effort in their computations [19]. GGVV (2012) report the average computation time used for solving a problem instance [57]. However, with regard to the challenging results found by EMD (2008), the termination criterion of EGRASP is adjusted to the limitation of generated local optima.

The number of iterations used by EGRASP for solving a problem instance is constant. But in contrast to EMD (2008), this number varies with regard to the problem size. The total number of 624 million iterations is split proportionally among the instances. More precisely, for an instance of class k with size $n_k \mathrm{x} m_k$, the number of iterations is as follows:

Instances	# ops.	$\mathcal{I}_k$	Instances	# ops.	$\mathcal{I}_k$
la01-la05	50	1.73×10^6	la21-la25	150	5.20×10^6
la06-la10	75	2.60×10^6	la26-la30	200	6.93×10^6
la11-la15	100	3.47×10^6	la31-la35	300	10.40×10^6
la16-la20	100	3.47×10^6	la36-la40	225	7.80×10^6

Tab. 7.8: Termination limits for the computations to Lawrence's instances.

$$\mathcal{I}_k = \frac{n_k \cdot m_k}{\sum_l |BS_l| \cdot n_l \cdot m_l} \cdot 624 \times 10^6$$

Note that $|BS_k|$ denotes the number of instances constituting class k. As a result, EGRASP terminates after 1.73 million iterations are carried out if, for instance, it is applied to solve the problem instance la01 from the class $10\,\text{x}\,5$, independent of the used due date factor f. Tab. 7.8 shows the computed number of iterations obtained for every instance class.

In order to exploit the performance quality of EGRASP in the best possible way, EGRASP is started once per problem instance, consuming the total number of iterations in a single run.

Tab. 7.9-7.11 show the results of EMD (2008), KB (2011), GGVV (2012) as well as the best solutions produced by EGRASP. Results reproducing the best known solution are asterisked. For EMD (2008), mean values and best values of the computations are reported. For the solution procedure of KB (2011), the best obtained value is only available. Moreover, the computational study of KB (2011) does not include the problem instances la31-la40. Finally, GGVV (2012) reports the mean values of their computational results. Therefore, the computational comparison of the four solution procedures may not be entirely accurate.

EMD (2008) is the first algorithm which has delivered 108 of the 120 best known solutions. KB (2011) has computed the best solution for la24 ($f = 1.6$) and GGVV (2012) has found the best known solution for the problem instance la36 ($f = 1.5$).

In Tab. 7.9-7.11, the bold values indicate problem instances for which EGRASP is able to generate a new best solution. This is the case for 10 of the 120 problem instances. For 52 of the 120 problem instances, the best known solution is reproduced by EGRASP. For the remaining 58 problem instances, the gained solution is worse than the best known solution. The TotalGap achieved in the computations is 3.31%.

The performance quality of EGRASP changes between the different instance classes. Problem instances of small size are relatively easy to be solved as indi-

				EMD (2008)		KB (2011)	GGVV (2012)	EGRASP
Inst.	n	m	BKS	mean	best	best	mean	best
la01	10	5	2299	*	*	*	*	*
la02	10	5	1762	*	*	*	*	*
la03	10	5	1951	*	*	*	*	*
la04	10	5	1917	*	*	*	*	*
la05	10	5	1878	*	*	*	*	*
la06	15	5	5810	5827	*	6008	5964	5815
la07	15	5	5765	5801	*	5961	5808	*
la08	15	5	5475	5482	*	5560	5533	*
la09	15	5	5608	5648	*	6116	*	*
la10	15	5	6618	6621	*	6734	*	*
la11	20	5	**11978**	12341	12039	12792	12200	*
la12	20	5	10542	10683	*	11238	10683	10633
la13	20	5	11557	11889	*	12533	11608	11652
la14	20	5	13107	13225	*	13681	13175	13384
la15	20	5	12330	12428	*	12964	12364	12545
la16	10	10	1169	*	*	*	*	*
la17	10	10	899	*	*	*	*	*
la18	10	10	929	*	*	*	*	*
la19	10	10	948	*	*	955	949	*
la20	10	10	805	*	*	*	838	*
la21	15	10	3560	3771	*	3841	3857	*
la22	15	10	4227	4471	*	4453	4317	4412
la23	15	10	3777	3955	*	4103	3957	*
la24	15	10	**3539**	3831	3553	3770	3697	*
la25	15	10	3313	3569	*	3724	3410	*
la26	20	10	**8946**	9748	9093	10562	9416	*
la27	20	10	**9090**	9860	9127	9827	10316	*
la28	20	10	**9091**	9757	9263	10198	9578	*
la29	20	10	8744	9397	*	9792	9354	9254
la30	20	10	8626	8986	*	9297	9060	8971
la31	30	10	27671	28999	*	—	30936	29463
la32	30	10	30301	32004	*	—	34228	32673
la33	30	10	25902	27216	*	—	31139	27265
la34	30	10	27901	29131	*	—	31139	29000
la35	30	10	29652	30979	*	—	31750	31583
la36	15	15	2957	3187	*	—	3132	3046
la37	15	15	2290	2880	*	—	2580	2605
la38	15	15	**2122**	2287	2159	—	2276	*
la39	15	15	1676	1861	*	—	1820	1702
la40	15	15	2078	2305	*	—	2207	2182
TotalGap				4.36	0.15	6.89	7.37	3.39
# BKS				10	34	9	10	23

Tab. 7.9: Computational results of EMD (2008), KB (2011), GGVV (2012) and EGRASP for Lawrence's instances ($f = 1.3$).

				EMD (2008)		KB (2011)	GGVV (2012)	EGRASP
Inst.	n	m	BKS	mean	best	best	mean	best
la01	10	5	1610	*	*	1616	*	*
la02	10	5	1028	*	*	*	*	*
la03	10	5	1280	*	*	*	*	*
la04	10	5	1277	*	*	*	*	*
la05	10	5	1205	*	*	*	*	*
la06	15	5	4592	4658	*	4821	4685	4597
la07	15	5	4470	4548	*	4624	*	*
la08	15	5	4034	4094	*	4423	4079	*
la09	15	5	4419	4421	*	4618	4424	*
la10	15	5	5148	*	*	5304	5162	*
la11	20	5	10170	10332	*	10682	10187	10304
la12	20	5	8982	9084	*	9494	9105	9143
la13	20	5	9626	9846	*	10158	9678	9786
la14	20	5	11286	11382	*	12024	11412	11323
la15	20	5	10343	10455	*	10464	10377	10539
la16	10	10	166	*	*	*	*	*
la17	10	10	260	*	*	*	*	*
la18	10	10	34	*	*	*	*	*
la19	10	10	21	*	*	23	*	*
la20	10	10	0	*	*	*	*	*
la21	15	10	1570	1697	*	1740	1760	*
la22	15	10	2015	2273	*	2099	2264	2130
la23	15	10	1510	1683	*	1731	*	*
la24	15	10	1570	1618	*	1849	1621	1577
la25	15	10	1430	1497	*	1499	1439	*
la26	20	10	5369	6106	*	6328	6324	5520
la27	20	10	**5757**	6142	5793	6252	6303	*
la28	20	10	5744	6254	*	6096	6096	6029
la29	20	10	5959	6392	*	6265	6027	6193
la30	20	10	5259	5496	*	5273	5539	5478
la31	30	10	22545	23417	*	—	24720	23591
la32	30	10	24467	25766	*	—	27364	25809
la33	30	10	20493	21767	*	—	22067	21457
la34	30	10	22572	23341	*	—	23974	22729
la35	30	10	24325	24654	*	—	25841	25375
la36	15	15	589	866	590	—	*	640
la37	15	15	413	589	*	—	415	430
la38	15	15	401	438	*	—	416	436
la39	15	15	0	9	*	—	7	8
la40	15	15	51	79	*	—	84	115
TotalGap				3.85	0.02	5.60	5.53	2.78
# BKS				11	38	8	13	18

Tab. 7.10: Computational results of EMD (2008), KB (2011), GGVV (2012) and EGRASP for Lawrence's instances ($f = 1.5$).

				EMD (2008)		KB (2011)	GGVV (2012)	EGRASP
Inst.	n	m	BKS	mean	best	best	mean	best
la01	10	5	1230	*	*	*	*	*
la02	10	5	695	*	*	*	*	*
la03	10	5	1024	*	*	*	*	*
la04	10	5	1029	*	*	1068	*	*
la05	10	5	877	*	*	*	*	*
la06	15	5	4098	4130	*	4180	4122	4116
la07	15	5	3843	3988	*	*	*	*
la08	15	5	3400	*	*	3584	3421	*
la09	15	5	3811	3835	*	4040	3815	*
la10	15	5	4503	4533	*	4728	4513	*
la11	20	5	9232	9399	*	9679	9301	9310
la12	20	5	**8222**	8302	8255	8663	8258	*
la13	20	5	8767	8916	*	9244	8845	8789
la14	20	5	10496	10594	*	11292	10564	10533
la15	20	5	9248	9392	*	9675	9365	9341
la16	10	10	0	*	*	*	*	*
la17	10	10	65	*	*	*	*	*
la18	10	10	0	*	*	*	*	*
la19	10	10	0	*	*	*	*	*
la20	10	10	0	*	*	*	*	*
la21	15	10	868	949	*	890	915	983
la22	15	10	1242	1450	*	1364	1404	1357
la23	15	10	**930**	977	936	1010	946	*
la24	15	10	693	773	752	*	749	762
la25	15	10	874	922	*	929	893	876
la26	20	10	4159	5125	*	5236	4890	4561
la27	20	10	**4203**	4590	4307	4441	4396	*
la28	20	10	4248	4594	*	4403	4545	4576
la29	20	10	4392	4706	*	5049	4569	4632
la30	20	10	3631	4131	*	3821	4211	4272
la31	30	10	20006	21141	*	—	21827	20776
la32	30	10	21722	22918	*	—	24038	22719
la33	30	10	17771	18801	*	—	19417	19105
la34	30	10	19693	20548	*	—	21589	20355
la35	30	10	20601	21991	*	—	22988	22847
la36	15	15	116	192	*	—	151	118
la37	15	15	0	*	*	—	*	*
la38	15	15	0	*	*	—	*	*
la39	15	15	0	*	*	—	*	*
la40	15	15	0	*	*	—	*	*
TotalGap				4.87	0.10	6.20	6.54	4.18
# BKS				15	35	11	15	21

Tab. 7.11: Computational results of EMD (2008), KB (2011), GGVV (2012) and EGRASP for Lawrence's instances ($f = 1.6$).

cated by the asterisks for the 15 instances la01-la05. For larger instances of size 30 x 10, EGRASP only generates solutions of moderate quality compared with the best solutions obtained by EMD (2008). The TotalGap of EGRASP related to the instance class 30 x 10 is 5.37%.

Comparing the outcome of EGRASP with the results of KB (2011), one observe that EGRASP dominates the solution procedure of KB (2011). For the instances la01-la30, EGRASP generates better solutions for 56 of 90 problem instances. For 27 problem instances, the achieved solution quality is the same for the two approaches. Therefore, EGRASP clearly outperforms the solution procedure of KB (2011). The comparison with the results of GGVV (2012) shows that EGRASP dominates this solution procedure as well. The reported average performance quality of GGVV (2012) is worse than the performance quality delivered by EGRASP as easily shown by the TotalGap value and # BKS. Remember that GGVV (2012) provides the best solution procedure for solving the benchmark set of Singer and Pinedo. The best results generated by EMD (2008) show that EGRASP is not dominating this solution procedure. In many cases, EMD (2008) has found a better solution.

EGRASP constantly delivers worse results in the instance class of size 30 x 10 as EMD (2008). In other instance classes, at least one new best solution is found or already known best solutions are at least reproduced. The reason for this weakness in the class 30 x 10 is quite similar to the reason of the failed search process for the instance orb08 ($f = 1.6$). EGRASP cannot identify the right region in the search space where it could start its search process successfully. EGRASP only searches in regions with moderate solution quality.

Nevertheless, 10 new best solutions are found in the benchmark set. In comparison with previous best known solutions, the achieved improvements of the objective function value are up to 2.5%. Therefore, it is confirmed that EGRASP is a powerful solution procedure for solving the JSPTWT.

7.2 Other Objective Functions

Even though EGRASP is designed for solving the JSPTWT, it is very interesting to see how it performs when other objective functions are pursued. In this section, EGRASP is applied for solving the job shop scheduling problem again. But, the problem structure is modified by replacing the objective function.

The first part of this section focuses on solving the JSP with the objective of minimizing the total flow time. In the second part, the problem of solving JSP with minimizing the number of tardy jobs in a schedule is considered. Both objective functions are closely related to the total weighted tardiness measure. Each of the three objectives depends on the completion times of the n jobs, and therefore, belongs to the class of regular measures. Recall that a regular measure represents a non-decreasing function of the completion times (see page 13 or [47]).

7.2.1 JSP with minimizing the Total Flow Time

This subsection treats the job shop scheduling problem with the objective of minimizing the total flow time of the jobs (JSPTFT). The flow time f_j of a job j denotes its duration in the production system, i. e. $f_j = c_j - r_j$. In the literature, this objective function is often referred to as the minimization of the sum of completion times [110] because release times r_j are given with the value zero. The minimization of the total flow time "gives an indication of the total holding or inventory costs incurred by the schedule"[17]. In particular, holding costs will increase if the job's completion time increases.

As mentioned in Chapter 2, the single machine scheduling problem with total weighted tardiness objective is an $\mathcal{NP}$-complete problem. On the other hand, by replacing the objective function with the minimization of the sum of (weighted) completion time, the single machine problem is solvable in polynomial time applying the (Weighted) Shortest Processing Time rule in the schedule generation process [110]. The additional introduction of job's release times $r_j \geq 0$ leads to the minimization of the flow time as objective and the considered single machine scheduling problem becomes $\mathcal{NP}$-hard [86]. As a result, the JSPTFT also belongs to the class of $\mathcal{NP}$-hard problems.

The number of publications exclusively considering the JSPTFT is very limited. The best-first search algorithm of Sierra and Varela [118] belongs to the class of exact optimization methods. This algorithm systematically explores the search space of the JSPTFT. In [119], the search algorithm is further improved with the help of new dominance rules. A combination of tabu search and genetic algorithm is used by González et al. [56] for the development of a hybrid metaheuristic. Note that all mentioned solution procedures have been published by the same research group.

[17] cf. M. L. Pinedo: Scheduling: Theory, Algorithms, and Systems [110], page 19.

Other research results are only presented in broad computational studies of solution procedures dealing with various objective functions (see e. g. [42, 66, 94, 115]).

The hybrid metaheuristic of González et al. [56] is the state-of-the-art solution procedure for solving the JSPTFT. The computational comparison with this solution procedure is interesting as well, due to the similarities to the hybrid metaheuristic of González et al. [57] developed for solving the JSPTWT (previously denoted by GGVV (2012) in Section 7.1). Recall that GGVV (2012) outperforms all other existing solution procedures on the benchmark set of Singer and Pinedo (see Subsection 7.1.2). The two metaheuristics - the first for solving the JSPTFT [56] and the second for solving the JSPTWT [57] - are presented by the same research group. Moreover, many components of the two procedures are identical. For example, in both metaheuristics, the implemented genetic algorithm exploits the Job Order Crossover operator proposed by Bierwirth [15]. Only a few components are designed in different ways. For example, the neighborhood operator used in [56] is designed like the single critical end insert (SCEI) neighborhood [36]. In [57], the operator bases on the principles of the critical end transpose (CET) neighborhood [101]. For this reason, the following part of the computational study focuses on the comparison with GVSV (2010) for solving the JSPTFT:

- GVSV (2010): Tabu search and genetic algorithm of González, Vela, Sierra and Varela, 2010 [56].

In their computational study, GVSV (2010) uses a benchmark set consisting of 56 problem instances. The benchmark set includes instances of size 10 x 5, 8 x 8, 9 x 9, 15 x 5, 20 x 5, and 10 x 10 so that six instance classes are distinguished. In the following computations, EGRASP is only applied to the last four instance classes since solving these larger instances lead to an appropriate comparison of the two procedures. The instance class of size 9 x 9 consists of 18 instances: abz05_9, abz06_9, ft10_9, la16_9-la20_9 and orb01_9-orb10_9. Originally, each of these instances incorporates ten jobs to be processed on ten machines. But, in the considered benchmark set, the instances are reduced by removing the last job and the last machine. Finally, the benchmark set contains five instances of size 15 x 5: la06-la10; five instances of size 20 x 5: la11-la15; and five instances of size 10 x 10: la16-la20. In total, the benchmark set consists of 33 problem instances.

In the study of GVSV (2010), the solution procedure is started 20 times for every problem instance. The average computation time is reported for each of the 33

	GVSV (2010)	EGRASP			GVSV (2010)	EGRASP	
Inst.	$\bar{t}$	t_{inst}	T_{inst}	Inst.	$\bar{t}$	t_{inst}	T_{inst}
9x9	82.2 s	64 s	1.280 s	20x5	789.2 s	617 s	12.340 s
15x5	318.0 s	249 s	4.980 s	10x10	104.2 s	82 s	1.640 s

Tab. 7.12: Time limits for the computations of the JSPTFT instances, depending on the size nxm.

problem instances. For each of the four instance classes, the mean computation time $\bar{t}$ is derived from the reported data (see Tab. 7.12). Using the same approach as the one in Section 7.1.2, the computation time limits of EGRASP t_{inst} and T_{inst} are calculated by adjusting it to the processing speed of the used computer[18]. The first time limit t_{inst} represents the single run time of EGRASP if it is started with the setting $(20/t_{inst})$, i. e. 20 runs of EGRASP are carried out each lasting t_{inst} seconds. The second time limit T_{inst} is used if EGRASP is started with the setting $(1/T_{inst})$. The values of t_{inst} and T_{inst} are also shown in Tab. 7.12.

Consequences of solving JSPTFT instances with EGRASP: For the following computations, EGRASP does not need any modification when solving the JSPTFT instances. The reason for the immediate application is that all release times are zero in the considered problem instances. By setting the due date of each job to zero, EGRASP assesses the solution quality as the resulting total flow time.

As a consequence of solving the JSPTFT, the disjunctive graph representation implies a growing effort of generating all neighboring schedules. More precisely, since due dates are set to zero, all n longest paths lead through the node set of the graph G', i. e. none of the longest paths contain the arc $(0, F_j)$. Remember that the longest path of job j consists of the arc $(0, F_j)$ if job j is completed on-time. But, this goal is never reachable if due dates are zero. Consequently, the number of critical arcs and blocks which are derived from the graph representation is comparatively very large in every step of the neighborhood search. In contrast to the JSPTWT, the single descent steps of the neighborhood search become very costly and the performance quality in a fixed time period deteriorates. This implication is also confirmed by computational experiments of GVSV (2010) (see [56]). Nevertheless, neither EGRASP nor GVSV (2010) reduces the set of neighboring schedules in advance so that the computational comparison bases on a fair competition.

[18]The computational experiments of GVSV (2010) are performed on a computer with 2.66 GHz [56].

				GVSV (2010)		EGRASP ($20/t_{inst}$)		EGRASP ($1/T_{inst}$)
Inst.	n	m	BKS†	mean	best	mean	best	best
abz05_9	9	9	8586	*	*	*	*	*
abz06_9	9	9	6524	*	*	*	*	*
ft10_9	9	9	5982	*	*	*	*	*
la16_9	9	9	5724	*	*	*	*	*
la17_9	9	9	5390	*	*	5396.6	*	*
la18_9	9	9	5770	*	*	*	*	*
la19_9	9	9	5891	*	*	*	*	*
la20_9	9	9	5915	*	*	*	*	*
orb01_9	9	9	6367	6371.6	*	*	*	*
orb02_9	9	9	5867	5867.6	*	*	*	*
orb03_9	9	9	6310	*	*	6321.1	6321	*
orb04_9	9	9	6661	6676.3	*	*	*	*
orb05_9	9	9	5605	*	*	*	*	*
orb06_9	9	9	6106	*	*	*	*	*
orb07_9	9	9	2668	*	*	*	*	*
orb08_9	9	9	5656	5670.5	5668	5779.7	5753	5668
orb09_9	9	9	6013	*	*	*	*	*
orb10_9	9	9	6328	*	*	*	*	*
la06	15	5	8625	8628.0	*	8675.6	8673	8635
la07	15	5	8069	8070.7	*	8116.5	8116	*
la08	15	5	7946	7962.4	*	8107.8	8051	8012
la09	15	5	9034	9072.6	*	9205.0	9136	9054
la10	15	5	8798	8799.6	*	8914.8	8833	8844
la11	20	5	13880	13985.5	*	14403.7	14377	14209
la12	20	5	11710	11753.1	*	12212.9	12186	12073
la13	20	5	13281	13367.6	*	13810.9	13786	13643
la14	20	5	14514	14573.4	*	14922.0	14922	14876
la15	20	5	14111	14187.4	*	14583.7	14563	14532
la16	10	10	7376	*	*	7393.0	7393	*
la17	10	10	6537	*	*	6546.4	*	*
la18	10	10	6970	*	*	6984.0	6984	*
la19	10	10	7217	7223.3	*	*	*	*
la20	10	10	7345	7402.4	*	*	*	*
TotalGap				0.21	0.00	1.25	1.11	0.79
# BKS				17	32	17	19	23

Tab. 7.13: Computational results of GVSV (2010) and EGRASP for the benchmark set of González et al. [56] (†the best known solutions of the 9x9 instances are optimal solutions).

Tab. 7.13 show the best known solutions from literature, the results of GVSV (2010) reported in [56] and the obtained results delivered by EGRASP. For the

instances of size 9x9, the best known solutions are optimal solutions [56, 117]. For the solution procedure of GVSV (2010), the table summarizes the mean values and best values of their computations. Furthermore, the outcome of the computations of EGRASP with the settings $(20/t_{inst})$ and $(1/T_{inst})$ is shown in the last columns. An asterisk indicates that the obtained value is equal to the objective function value of the best known solution. At the bottom of the table, the two performance measures TotalGap and # BKS provide an assessment of the delivered results.

The computational results in Tab. 7.13 show a similar performance quality of the two solution procedures for the instances of size 9x9. Here, the best obtained results delivered by EGRASP with the setting $(1/T_{inst})$ are identical to the best results produced by GVSV (2010). The average performance quality of EGRASP in the setting $(20/t_{inst})$ indicated by its mean values shows that EGRASP performs almost as good as the solution procedure of GVSV (2010). For the problem instances of the classes 15x5 and 20x5, the experiments come up with better results of the solution procedure of GVSV (2010). The generated schedules are quite consistently better than those achieved by EGRASP. For the problem instances la16-la20 of size 10x10, the performance results are very similar again. Both solution procedures deliver all best known solutions. The average performance quality observed for the instances la19-la20 is slightly better using EGRASP.

The best results of EGRASP are produced in the setting $(1/T_{inst})$ allowing to consume the complete computation time in a single run. Due to the delivered solution quality in the instance classes 15x5 and 20x5, the total performance quality of EGRASP is slightly weaker indicated by the TotalGap value and # BKS.

The computational results show that EGRASP is also applicable and useful for solving the JSPTFT. Due to the similarities of the two objective functions - total weighted tardiness minimization and total flow time minimization - EGRASP does not need to be modified for the considered benchmark set. The obtained results confirm that EGRASP provides an excellent performance quality for problem instances of size 9x9 and 10x10.

7.2.2 JSP with minimizing the Number of Tardy Jobs

This subsection focuses on solving the job shop scheduling problem with the objective of minimizing the (weighted) number of tardy jobs (in the following referred to as JSPTWU). This objective function is also relevant in the consideration of meeting

job specific due dates. In logistic systems, the objective function value corresponds to the α-service-level, which is a measure of the non-stock-out incidence. Pinedo states that "the weighted number of tardy jobs is not only a measure of academic interest, it is often an objective in practice as it is a measure that can be recorded very easily."[19]

The total weighted number of tardy jobs is defined as $TWU = \sum_{j=1}^{n} w_j \cdot u_j$, where the variable $u_j \in \{0, 1\}$ is the tardiness indicator of job j, with $u_j = 1$ if job j is tardy, i. e. $c_j > d_j$, and $u_j = 0$ otherwise. In the literature, the objective function is also treated as the meet-due-date which is calculated by $1 - \sum u_j/n$. The meet-due-date rate objective is obviously equivalent to the minimization of the number of tardy jobs $\sum u_j$, where jobs weights do not contribute to the objective function value. As a result, the optimization problem becomes a maximization problem. A meet-due-date rate of 1 indicates that all jobs are on-time.

The single machine scheduling problem with minimizing the number of tardy jobs $\sum u_j$ as objective can be solved in polynomial time using Moore's algorithm [100]. By integrating job weights into the objective function, the single machine scheduling problem with TWU objective already becomes $\mathcal{NP}$-hard [110]. Consequently, the JSPTWU belongs to the class of $\mathcal{NP}$-hard problems. Moreover, the JSP with the unweighted variant of the objective function, i. e. with meet-due-date rate objective, is also classified as $\mathcal{NP}$-hard problem [86].

There are only few publications tackling the job shop scheduling with minimizing the (weighted) number of tardy jobs as objective. The objective function is mostly considered in broad computational studies that are used for demonstrating the general efficiency of a solution procedure like dispatching rules or genetic algorithms (see e. g. [42, 66, 94, 115]). The papers of Chiang and Fu [31, 32] are the only known publications which exclusively deal with the JSPTWU. More precisely, the considered objective function is the minimization of tardy jobs, i. e. the consideration of job weights is omitted. In [31], the authors propose a memetic algorithm for solving the problem. The memetic algorithm combines a genetic algorithm with local search, also known as genetic local search. In [32], Chiang and Fu consider the JSP with an extension to sequence-dependent setup times and present a genetic algorithm incorporating simulation-based scheduling mechanisms.

The following part of the computational study focuses on the comparison of

[19] cf. M. L. Pinedo: Scheduling: Theory, Algorithms, and Systems [110], page 19.

EGRASP with the memetic algorithm of Chiang and Fu [31]. In order to provide a reliable assessment, the computational study is enhanced by the genetic algorithm of Mattfeld and Bierwirth [94]. This solution procedure is designed for solving job shop scheduling problems with any tardiness related objective function. The computational tests in [94] show that it produces a superior schedule quality among a group of genetic algorithms. Therefore, it is included as further benchmark for solving the JSPTWU. In summary, the outcome of EGRASP is compared with the results obtained by CF (2008) and MB (2004):

- CF (2008): Rule-centric memetic algorithm of Chiang and Fu, 2008 [31].
- MB (2004): Efficient genetic algorithm of Mattfeld and Bierwirth, 2004 [94]

The tests focus on the benchmark set of CF (2008) proposed in [31]. This benchmark set consists of two instance classes each containing ten problem instances. The first class includes the problem instances ap1-ap10 each of size $20\,\mathrm{x}\,20$, whereas the instances ap11-ap20 of the second class have size $60\,\mathrm{x}\,20$. Each of the 20 instances is randomly generated. Due dates of the jobs are generated so that the meet-due-date rate of a schedule is in the range $[0.75, 0.80]$ if this schedule is constructed with the help of a dispatching rule [31].

In order to use EGRASP for solving the JSPTWU, the following modifications have to be done. The assessment of the computed local optima is based on the weighted number of tardy jobs. This means that the best found solution in the search process is the one with lowest TWU value so far. If there is another local optimum with the same TWU value, this new found schedule is accepted as best found solution if the resulting total weighted tardiness value in this new schedule is better. The main components in EGRASP, i. e. the construction algorithm, the improvement algorithm, and the adaptive components are unchanged so that e. g. the implemented neighborhoods are still the same, and the assessment of neighboring schedules is still based on the total weighted tardiness objective.

For the computations performed by CF (2008), there is no information about the used computer system. The solution procedure of CF (2008) is started five times for every problem instance. Every run terminates after 180 seconds. In total, 900 seconds computation time are spent for solving a problem instance. The computations of EGRASP and MB (2004) are carried out in a comparable way. More precisely, EGRASP is applied with the settings (5/180) and (1/900) for each problem instance. The solution procedure of MB (2004) is restarted as long as the computation time

			CF (2008)		MB (2004)		EGRASP (5/180)		EGRASP (1/900)
Inst.	n	m	mean	best	mean	best	mean	best	best
ap1	20	20	n.r.	n.r.	0.701	0.750	0.850	0.850	0.900
ap2	20	20	n.r.	n.r.	0.829	0.900	0.950	0.950	0.950
ap3	20	20	n.r.	n.r.	0.657	0.750	0.850	0.850	0.900
ap4	20	20	n.r.	n.r.	0.763	0.800	0.850	0.850	0.900
ap5	20	20	n.r.	n.r.	0.788	0.850	0.900	0.900	0.950
ap6	20	20	n.r.	n.r.	0.779	0.800	0.900	0.900	0.900
ap7	20	20	n.r.	n.r.	0.755	0.800	0.900	0.900	0.900
ap8	20	20	n.r.	n.r.	0.773	0.850	0.950	0.950	0.950
ap9	20	20	n.r.	n.r.	0.800	0.850	0.900	0.900	0.950
ap10	20	20	n.r.	n.r.	0.847	0.900	0.960	1.000	1.000
on average			0.895	0.915	0.769	0.825	0.901	0.905	0.930
ap11	60	20	n.r.	n.r.	0.544	0.600	0.800	0.800	0.833
ap12	60	20	n.r.	n.r.	0.504	0.533	0.800	0.800	0.800
ap13	60	20	n.r.	n.r.	0.676	0.717	0.867	0.867	0.867
ap14	60	20	n.r.	n.r.	0.594	0.633	0.783	0.783	0.783
ap15	60	20	n.r.	n.r.	0.576	0.600	0.817	0.817	0.817
ap16	60	20	n.r.	n.r.	0.558	0.617	0.733	0.733	0.783
ap17	60	20	n.r.	n.r.	0.525	0.567	0.750	0.750	0.767
ap18	60	20	n.r.	n.r.	0.597	0.633	0.817	0.817	0.817
ap19	60	20	n.r.	n.r.	0.589	0.617	0.800	0.800	0.800
ap20	60	20	n.r.	n.r.	0.554	0.583	0.733	0.733	0.750
on average			0.863	0.880	0.572	0.610	0.790	0.790	0.802

Tab. 7.14: Computational results of CF (2008), MB (2004) and EGRASP for the benchmark set of Chiang and Fu, after consuming 900 seconds computation time, n.r. = not reported.

limit of 900 seconds is not exceeded.

For the entire benchmark set, CF (2008) only report aggregated results in [31]. Tab. 7.14 shows the average meet-due-date rate for the two instance classes. Furthermore, the table presents the obtained results of MB (2004) and EGRASP. In the columns "mean", all runs are used for the calculation of the average meet-due-date rate resulting from solving a problem instance. In the columns "best", only the best obtained result is presented for a problem instance. The computations are additionally assessed by the average meet-due-date rate derived for the two instance classes 20x20 and 60x20.

The computational comparison of EGRASP with the results of CF (2008) and MB (2004) discloses that EGRASP produces the best results for instances of size 20x20. The average performance quality observed in the setting (5/180) as well as

the gained best solutions delivered by the EGRASP in the setting (1/900) dominate the results of the two other approaches. In the second class of instances, the solution procedure of CF (2008) delivers the best results. The outcome of EGRASP shows that only schedules of moderate solution quality are generated. The gained results of MB (2004) exhibit a poor solution quality since the due dates in the instances are generated in such a way that the application of a dispatching rule can yield a better meet-due-date rate.

The computational tests show that EGRASP is useful for solving JSPTWU instances. EGRASP provides an excellent performance quality for problem instances of size 20 x 20. For larger instances, EGRASP is only dominated by the solution procedure of CF (2008).

In order to provide further computational results for the JSPTWU, EGRASP is applied to the problem instances of the benchmark set of Singer and Pinedo [121]. The objective function is replaced by the TWU objective. The gained results are reported in Appendix E. These results shall provide the opportunity to assess the performance quality of solution procedures in future research.

7.3 Summary

The presented computational study provides a broad performance comparison of EGRASP with existing methods. In total, 239 problem instances have been solved by EGRASP for assessing its performance for solving the JSP with TWT measure as well as with closely related objective function.

In all computational tests, it becomes apparent that EGRASP benefits from a longer run time in the computations. This observation shows that EGRASP is capable to use computation time for improving solutions and solution quality further. Moreover, it confirms that EGRASP systematically explores the search space. As a result, the best performance quality of EGRASP is always obtained if the entire computational resources are consumed in a single run. Consequently, it is not necessary to restart the computations a second or third time.

The computational results show that EGRASP is highly competitive in the group of best performing solution procedure. Even though EGRASP is only ranked as the second best solution procedure in some computational tests, its performance quality is constantly superior. In particular, EGRASP generates excellent solutions for

square JSPTWT instances. Moreover, many new best solutions have been found by EGRASP.

Its applicability for solving the JSP with TFT measure or TWU measure has been demonstrated in the second section. EGRASP also produces solutions of high quality if JSPTFT instances or JSPTWU instances have to be solved. For this reason, EGRASP represents an appropriate method for solving these problems which comes up without additional costs.

Chapter 8

Conclusion

This thesis dealt with the job shop scheduling problem which is among the most intensively studied combinatorial optimization problems. Research on the job shop scheduling has mainly focused on the minimization of the makespan as the objective, since the corresponding schedules lead to a maximum productivity and output for companies. In recent years, the observance of due dates has become more and more important. Therefore, recent research concentrates on the job shop scheduling problem with the objective of minimizing the total weighted tardiness (JSPTWT).

The literature related to the JSPTWT already provides several studies and solution procedures. However, many concepts for solution procedures are only based on the adaptation of solution techniques for the minimum makespan JSP or the hybridization of different metaheuristics. Often, it is unknown why these solution procedures are successful for solving the JSPTWT or what components are the major originators for the gained solution quality. For this reason, the aim of this thesis is to provide more insight into the problem structure of JSPTWT; to analyze several techniques and concepts for solving the JSPTWT; and in the end, to present a new and competitive solution procedure.

The first scientific contribution of this thesis is a model of the JSPTWT as disjunctive graph. The presented representation form differs substantially from other approaches in order to obtain a suitable and proper graphical representation. The disjunctive graph representation contains additional node and arc sets such that this representation enables an immediate assessment of the objective function value. Based on this disjunctive graph representation, the critical tree has been introduced which is a structural element to identify important precedence relations which causes

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

the solution quality. The modeling framework forms the basis for the investigation of solution techniques developed for the JSPTWT.

The analysis and study of eleven neighborhood operators form the second scientific contribution. A neighborhood takes the most important role within a local search procedure, since it essentially affects the gained performance quality. Therefore, the characteristics and the performance quality of the eleven neighborhoods are subject of the thesis. The results demonstrate that the application of one neighborhood operator or a combination of them cannot produce a satisfying solution quality. However, the assessment enables to separate well performing neighborhoods from poor performing ones. In this way, an adequate selection of neighborhood operators is available for implementing them in a local search procedure, and in a local search based metaheuristic as well.

An essential part of local search procedures is the assessment of newly generated neighboring solutions. For the JSPTWT, the Heads Updating procedure has been developed, constituting this thesis's third scientific contribution. Traditional approaches for the assessment of newly generated solutions rely on an estimation of the resulting total weighted tardiness based on the modifications made by the neighborhood operator. In contrast to these concepts, the Heads Updating procedure exactly assesses the solution quality by considering all modifications as well as their effects. The procedure is generally applicable for all kinds of neighborhoods and its application always leads to an accurate neighborhood search process. The presented performance test has shown that it outperforms other traditional approaches.

The fourth scientific contribution forms an overview of metaheuristic concepts and the derivation of further promising solution techniques. The similarities of the minimum makespan JSP and JSPTWT are illustrated by a brief survey of the fitness landscape concept. This concept provides insights to the search behavior of solution procedures. In order to improve the search process, the surface structure of the fitness landscape can be investigated. Implications, which are obtained by the analysis of the fitness landscape derived from the minimum makespan JSP, can also be used for the development of solution techniques for solving the JSPTWT as well. In this way, solution techniques and mechanisms are recommended which guide the search process in an appropriate manner. For solving the JSPTWT, using path relinking is potentially advantageous.

The primary goal of this thesis, and thus, the main scientific contribution is to present a new solution procedure which catches all previous results. The new solution procedure is based on the metaheuristic GRASP. Because of several modifications and integrating other search techniques, this procedure is called EGRASP. The computational experiments have shown that EGRASP is a good alternative for solving the JSPTWT. The obtained performance quality, as well as the identification of new best solutions, confirms that EGRASP delivers a superior schedule quality. Further computational tests have shown that EGRASP can also produce schedules of high quality for similar optimization problems.

This thesis contains a multitude of new results for modeling and solving the JSPTWT. Even though EGRASP produces quite good results, it cannot deliver the best known solution for all considered JSPTWT instances. From this perspective, additional investigations into the search space and the associated fitness landscape of the JSPTWT form the potential for future research. The detailed analysis of the search process on the fitness landscape can result into new solution techniques which are able to guide the search successfully for all JSPTWT instances.

Further research potential is given by the consideration of additional features in the problem structure, e. g. the limitation of buffers in front of machines. Even though the standardized problem structure provides a basis for the suitable comparison of solution procedures, the problems occurring in companies might be more constrained. Therefore, the specific investigation of such job shop scheduling problems may result in further sophisticated solution techniques.

Bibliography

[1] E. H. L. Aarts and J. K. Lenstra. Introduction. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 1–18. John Wiley & Sons Ltd, 1997.

[2] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[3] R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29(4):393–430, 2003.

[4] S. B. Akers and J. Friedman. A non-numerical approach to production scheduling problems. *Operations Research*, 3(4):429–442, 1955.

[5] E. J. Anderson and J. C. Nyirenda. Two new rules to minimize tardiness in a job shop. *The International Journal of Production Research*, 28(12):2277–2292, 1990.

[6] Anon. Internationale Produktionsstudie 2003. Stellhebel für den Markterfolg - Branchenanalyse Maschinenbau. Published by Droege & Comp. GmbH and Fraunhofer Institut für Produktionstechnologie IPT Aachen, 2003.

[7] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

[8] V. A. Armentano and C. R. Scrich. Tabu search for minimizing total tardiness in a job shop. *International Journal of Production Economics*, 63(2):131–140, 2000.

[9] M. Asano and H. Ohta. A heuristic for job shop scheduling to minimize total weighted tardiness. *Computers & Industrial Engineering*, 42(2):137–147, 2002.

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

[10] K. R. Baker and J. W. M. Bertrand. A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management*, 3(1):37–42, 1982.

[11] E. Balas and A. Vazacopoulos. Guided local search and the shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.

[12] P. Baptiste, M. Flamini, and F. Sourd. Langrangian bound for just-in-time job-shop scheduling. *Computers & Operations Research*, 35(3):906–915, 2008.

[13] J. E. Beasley. OR-Library. http://people.brunel.ac.uk/ mastjjb/jeb/info.html, 2014.

[14] R. E. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[15] C. Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR-Spectrum*, 17(2-3):87–92, 1995.

[16] C. Bierwirth, D. C. Mattfeld, and H. Kopfer. On permutation representations for scheduling problems. In *Parallel Problem Solving from Nature — PPSN IV*, pages 310–318. Springer, 1996.

[17] S. Binato, W. J. Hery, D. Loewenstern, and M. G. C. Resende. A greedy randomized adaptive search procedure for job shop scheduling. In C. C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 58–79. Kluwer Academic Publishers, 2002.

[18] J. Blazewicz, E. Pesch, and M. Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2):317–331, 2000.

[19] K. Bülbül. A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research*, 38(6):967–983, 2011.

[20] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: a survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[21] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[22] R. Braune, G. Zäpfel, and M. Affenzeller. Enhancing local search algorithms for solving job shops with min-sum objectives by approximate move evaluation. *Journal of Scheduling*, 16(5):495–518, 2013.

[23] P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer, 2002.

[24] P. Brucker. The job-shop problem: Old and new challenges. In *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA)*, pages 15–22, 2007.

[25] P. Brucker and S. Knust. *Complex Scheduling*. Springer, Berlin, 2006.

[26] J. A. Buzacott, H. Corsten, R. Gössinger, and H. M. Schneider. *Production Planning and Control: Basics and Concepts*. Oldenbourg Verlag, 2012.

[27] X. Cai and C. J. Goh. A fast heuristic for the train scheduling problem. *Computers & Operations Research*, 21(5):499–510, 1994.

[28] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[29] D. C. Carroll. *Heuristic sequencing of single and multiple component jobs*. PhD thesis, Massachusetts Institute of Technology, 1965.

[30] H. Chen and P. B. Luh. An alternative framework to langrangian relaxation approach for job shop scheduling. *European Journal on Operational Research*, 149(3):499–512, 2003.

[31] T.-C. Chiang and L.-C. Fu. A rule-centric memetic algorithm to minimize the number of tardy jobs in the job shop. *International Journal of Production Research*, 46(24):6913–6931, 2008.

[32] T.-C. Chiang and L.-C. Fu. Using a family of critical ratio-based approaches to minimize the number of tardy jobs in the job shop with sequence dependent setup times. *European Journal of Operational Research*, 196(1):78–92, 2009.

[33] R. W. Conway. Priority dispatching and job lateness in a job shop. *Journal of Industrial Engineering*, 16(4):228, 1965.

[34] H. Corsten and R. Gössinger. *Einführung in das Supply Chain Management*. Oldenbourg Verlag, 2nd edition, 2008.

[35] K. M. J. de Bontridder. Minimizing total weighted tardiness in a generalized job shop. *Journal of Scheduling*, 8(6):479–496, 2005.

[36] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(3):231–252, 1993.

[37] W. Domschke and A. Drexl. *Einführung in Operations Research*. Springer, 8th edition, 2011.

[38] M. Dorigo. *Optimization, learning and natural algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[39] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[40] R. Driessel and L. Mönch. Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. *Computers & Industrial Engineering*, 61(2):336–345, 2011.

[41] I. Essafi, Y. Mati, and S. Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35(8):2599–2616, 2008.

[42] H.-L. Fang, D. Corne, and P. Ross. A genetic algorithm for job-shop problems with various schedule quality criteria. In T. C. Fogarty, editor, *Evolutionary Computing*, pages 39–49. Springer, 1996.

[43] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.

[44] P. Festa and M. G. C. Resende. An annotated bibliography of GRASP - Part I: Algorithms. *International Transactions in Operational Research*, 16(1):1–24, 2009.

[45] P. Festa and M. G. C. Resende. An annotated bibliography of GRASP - Part II: Applications. *International Transactions in Operational Research*, 16(2):131–172, 2009.

[46] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall (New Jersey), 1963.

[47] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, Chichester (England), 1982.

[48] M. Gendreau and J.-Y. Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005.

[49] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1959.

[50] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[51] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[52] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[53] F. Glover and G. A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

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

[55] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[56] M. A. González, C. R. Vela, M. R. Sierra, and R. Varela. Tabu search and genetic algorithm for scheduling with total flow time minimization. In *Proceedings of the workshop on constraint satisfaction techniques for planning and scheduling problems (COPLAS)*, pages 33–41, 2010.

[57] M. Á. González, I. González-Rodríguez, C. R. Vela, and R. Varela. An efficient hybrid evolutionary algorithm for scheduling with setup times and weighted tardiness minimization. *Soft Computing*, 16(12):2097–2113, 2012.

[58] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnoy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[59] P. Hansen, N. Mladenović, and J. A. M. Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[60] R. Haupt. A survey of priority rule-based scheduling. *OR-Spektrum*, 11(1):3–16, 1989.

[61] D. J. Hoitomt, P. B. Luh, and K. R. Pattipati. A practical approach to job-shop scheduling problems. *IEEE Transactions on Robotics and Automation*, 9(1):1–13, 1993.

[62] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* University of Michigan Press, 1975.

[63] O. Holthaus and C. Rajendran. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105, 1997.

[64] K.-L. Huang and C.-J. Liao. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, 35(4):1030–1046, 2008.

[65] J. R. Jackson. An extension of johnson's results on job lot scheduling. *Naval Research Logistics Quarterly*, 3(3):201–203, 1956.

[66] M. S. Jayamohan and C. Rajendran. New dispatching rules for job shop scheduling: a step forward. *International Journal of Production Research*, 38(3):563–586, 2000.

[67] J. J. Kanet and J. C. Hayya. Priority dispatching with operation due dates in a job shop. *Journal of Operations Management*, 2(3):167–175, 1982.

[68] J. J. Kanet and X. Li. A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling*, 7(4):261–276, 2004.

[69] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[70] S. Kauffman. *The origins of order: Self organization and selection in evolution.* Oxford University Press, 1993.

[71] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.

[72] B. Khosravi, J. A. Bennell, and C. N. Potts. Train Scheduling and Rescheduling in the UK with a Modified Shifting Bottleneck Procedure. In *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, pages 120–131, 2012.

[73] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[74] S. Knust. Complexity Results for scheduling problems. http://www.informatik.uni-osnabrueck.de/knust/class/, 2009.

[75] S. Kreipl. *Werkstattsteuerung bei alternativen Arbeitsplänen.* PhD thesis, University of Passau, 1998.

[76] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.

[77] S. Kreipl and M. L. Pinedo. Planning and scheduling in supply chains: an overview of issues in practice. *Production and Operations Management*, 13(1):77–92, 2004.

[78] J. Kuhpfahl and C. Bierwirth. A new neighbourhood operator for the job shop scheduling problem with total weighted tardiness objective. In *Proceedings of International Conference on Applied Mathematical Optimization and Modelling*, pages 204–209, 2012.

[79] E. Kutanoglu and I. Sabuncuoglu. An analysis of heuristics in a dynamic job shop with weighted tardiness objectives. *International Journal of Production Research*, 37(1):165–187, 1999.

[80] B. Lageweg, J. K. Lenstra, and A. H. G. Rinnoy Kan. Job shop scheduling by implicit enumeration. *Management Science*, 24(4):441–450, 1977.

[81] M. Laguna and R. Marti. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

[82] G. Lancia, F. Rinaldi, and P. Serafini. A time-indexed LP-based approach for min-sum job-shop problems. *Annals of Operations Research*, 186(1):175–198, 2011.

[83] E. L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

[84] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[85] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Published by Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania., 1984.

[86] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[87] C.-J. Liao and H.-C. Juan. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research*, 34(7):1899–1909, 2007.

[88] S.-C. Lin, E. D. Goodman, and W. F. Punch III. A genetic algorithm approach to dynamic job shop scheduling problem. In *Proceedings of 7th International Conference on Genetic Algorithms (ICGA)*, pages 481–488, 1997.

[89] H. R. Lourenco, O. Martin, and T. Stützle. A beginner's introduction to iterated local search. In *Proceedings of MIC'2001 Metaheuristics International Conference*, pages 1–6, 2001.

[90] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2003.

[91] Y. Mati, S. Dauzère-Pérès, and C. Lahlou. A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research*, 212(1):33–42, 2011.

[92] H. Matsuo, C. J. Suh, and R. S. Sullivan. A controlled search simulated annealing method for the general jobshop scheduling problem. *Working paper 03-04-88. University Texas*, 1988.

[93] D. C. Mattfeld. *Evolutionary Search and the Job Shop: investigations on genetic algorithms for production*. PhD thesis, University of Bremen, 1996.

[94] D. C. Mattfeld and C. Bierwirth. An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research*, 155(3):616–630, 2004.

[95] D. C. Mattfeld, C. Bierwirth, and H. Kopfer. A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, 86(1):441–453, 1999.

[96] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New ideas in optimization*, pages 245–260. Citeseer, 1999.

[97] Z. Michalewicz and D. B. Fogel. *How to solve it: modern heuristics*. Springer, 2004.

[98] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[99] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[100] J. M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

[101] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[102] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.

[103] E. Nowicki and C. Smutnicki. Some new ideas in TS for job shop scheduling. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution*, pages 165–190. Springer, 2005.

[104] E. Oliveira and B. M. Smith. A job-shop scheduling model for the single-track railway scheduling problem. Technical report, University of Leeds.

[105] I. H. Osman, H. Belouadah, K. Fleszar, and M. Saffar. Hybrid of the weighted minimum slack and shortest processing time dispatching rules for the total weighted tardiness single machine scheduling problem with availability constraints. In *Proceedings of the 4rd Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA)*, pages 202–215, 2009.

[106] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[107] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineralo, 1998.

[108] P. M. Pardalos, O. V. Shylo, and A. Vazacopoulos. Solving job shop scheduling problems utilizing the properties of backbone and "big valley". *Computational Optimization and Applications*, 47(1):61–76, 2010.

[109] M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, New York, 2nd edition, 2009.

[110] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Upper Saddle River (NJ), 4th edition, 2012.

[111] M. L. Pinedo and M. Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46(1):1–17, 1999.

[112] L. S. Pitsoulis and M. G. C. Resende. A greedy randomized adaptive search procedure. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.

[113] E. Pitzer and M. Affenzeller. A comprehensive survey on fitness landscape analysis. In J. Fodor, R. Klempous, and C. Paz Suárez Araujo, editors, *Recent Advances in Intelligent Engineering Systems*, pages 161–191. Springer, 2012.

[114] M.-C. Portmann. Genetic algorithms and scheduling: a state of the art and some propositions. In *Proceedings of the Workshop on Production Planning and Control*, pages i–xxiv, 1996.

[115] C. Rajendran and O. Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1):156–170, 1999.

[116] F. Rothlauf. *Design of modern heuristics: principles and application*. Springer, 2011.

[117] M. R. Sierra. *Métodos de Poda por Dominancia en Búsqueda Heurística. Aplicaciones a Problemas de Scheduling*. PhD thesis, Universidad de Oviedo, 2009.

[118] M. R. Sierra and R. Varela. A new admissible heuristic for the job shop scheduling problem with total flow time. In *Proceedings of Workshop on Constraint Satisfaction Techniques for Planning and Scheduling. International conference on Automated Planning and Scheduling. ICAPS*, pages 1–8, 2008.

[119] M. R. Sierra and R. Varela. Pruning by dominance in best-first search for the job shop scheduling problem with total flow time. *Journal of Intelligent Manufacturing*, 21(1):111–119, 2010.

[120] M. Singer. Decomposition methods for large job shops. *Computers & Operations Research*, 28(11):193–207, 2001.

[121] M. Singer and M. L. Pinedo. Computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 29:109–119, 1998.

[122] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[123] P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20(1):1–45, 1996.

[124] K. Steinhöfel, A. Albrecht, and C.-K. Wong. Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research*, 118(3):524–548, 1999.

[125] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

[126] M. J. Streeter and S. F. Smith. How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research*, 26(1):247–287, 2006.

[127] T. G. Stützle. *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. PhD thesis, Technical University Darmstadt, 1999.

[128] E. D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[129] E. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[130] E.-G. Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[131] R. Tavakkoli-Moghaddam, M. Khalili, and B. Naderi. A hybridization of simulated annealing and electromagnetic-like mechanism for job shop problems with machine availability and sequence-dependent setup times to minimize total weighted tardiness. *Soft Computing*, 13(10):995–1006, 2009.

[132] V. G. Timkovsky. A polynomial-time algorithm for the two-machine unit-time release-date job-shop schedule-length problem. *Discrete Applied Mathematics*, 77(2):185–200, 1997.

[133] V. G. Timkovsky. Is a unit-time job shop not easier than identical parallel machines? *Discrete Applied Mathematics*, 85(2):149–162, 1998.

[134] R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996.

[135] P. Van Hentenryck and L. Michel. Scheduling abstractions for local search. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 319–334. Springer, 2004.

[136] P. J. M. van Laarhoven. *Theoretical and Computational Aspects of Simulated Annealing*. PhD thesis, Erasmus University Rotterdam, 1988.

[137] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[138] M. Van Steen. *Graph Theory and Complex Networks: An Introduction*. On Demand Publishing, 2010.

[139] A. P. J. Vepsalainen and T. E. Morton. Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047, 1987.

[140] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, 1999.

[141] J.-P. Watson. *Empirical modeling and analysis of local search algorithms for the job-shop scheduling problem*. PhD thesis, Colorado State University, 2003.

[142] J.-P. Watson. An introduction to fitness landscape analysis and cost models for local search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 599–623. Springer, 2010.

[143] J.-P. Watson, J. C. Beck, A. E. Howe, and L. D. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143(2):189–217, 2003.

[144] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics*, 63(5):325–336, 1990.

[145] S. D. Wu, E.-S. Byeon, and R. H. Storer. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999.

[146] G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*. Springer, 2010.

[147] C. Y. Zhang, P. G. Li, Z. L. Guan, and Y. Q. Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007.

[148] R. Zhang and J. Kuhpfahl. Corrigendum to "A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective" [Computers & Operations Research 38 (2011) 854-867]. *Computers & Operations Research*, 40(11):2816, 2013.

[149] R. Zhang, S. Song, and C. Wu. A hybrid artificial bee colony algorithm for the job shop scheduling problem. *International Journal of Production Economics*, 141(1):167–178, 2013.

[150] R. Zhang and C. Wu. A hybrid approach to large-scale job shop scheduling. *Applied Intelligence*, 32(1):47–59, 2010.

[151] R. Zhang and C. Wu. A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, 38(5):854–867, 2011.

[152] R. Zhang and C. Wu. A PMBGA to optimize the selection of rules for job shop scheduling based on the Giffler-Thompson algorithm. *Journal of Applied Mathematics*, 2012:1–17, 2012.

[153] R. Zhang and C. Wu. A neighbourhood property for the job shop scheduling problem with application to hybrid particle swarm optimization. *IMA Journal of Management Mathematics*, 24(1):111–134, 2013.

[154] H. Zhou, W. Cheung, and L. C. Leung. Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research*, 194(3):637–649, 2009.

Appendix A

Applying Neighborhood Operators: the example ft06 ($f = 1.3$)

In order to demonstrate the different perturbation schemes of the eleven proposed neighborhoods, the following overview provides one example to each of them. Starting from the FCFS schedule of instance ft06 with $f = 1.3$ as initial solution (see Section 2.6 with Fig. 2.6 - Fig. 2.8 for the graph representation G', the Gantt-Chart and the corresponding critical tree $\mathcal{CT}$), every operator is applied once for generating a corresponding neighboring solution. The FCFS schedule has a total weighted tardiness of 172. Tab. A.1 shows the set of the critical arcs and blocks derived from this schedule.

Block no.	Mch.	# $\mathcal{LP}$	Operation Order
1	3	3	$(1/1) \rightarrow (1/3) \rightarrow (1/5) \rightarrow (2/2)$
2	5	3	$(3/2) \rightarrow (3/5) \rightarrow (6/3) \rightarrow (6/1)$
3	5	2	$(3/2) \rightarrow (3/5) \rightarrow (6/3) \rightarrow (6/1) \rightarrow (5/4)$
4	5	1	$(3/2) \rightarrow (3/5) \rightarrow (6/3) \rightarrow (6/1) \rightarrow (5/4) \rightarrow (5/6)$
5	2	1	$(1/2) \rightarrow (1/4) \rightarrow (1/6) \rightarrow (3/1)$
6	6	1	$(5/1) \rightarrow (4/2)$
7	4	1	$(6/2) \rightarrow (6/5)$

Tab. A.1: Overview of the critical blocks for the FCFS solution of ft06 ($f = 1.3$).

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

CET Perturbation scheme: reversal of one critical arc at the beginning or at the end of a critical block. Example: block no. 1, reversing $(1/5) \rightarrow (2/2)$ $\Rightarrow$ TWT value of new schedule: 137.

CET+2MT Perturbation scheme: reversal of one critical arc $v \rightarrow w$ at the beginning or at the end of a critical block and additionally, at most, two further reversals related to $SM(v)$ and $PM(w)$. Example: block no. 4, reversing $(3/2) \rightarrow (3/5)$, additional reversals of $(4/2) \rightarrow (4/5)$ (machine 6) and $(3/1) \rightarrow (2/5)$ (machine 2) are performed $\Rightarrow$ TWT value of new schedule: 183.

CE3P Perturbation scheme: permutation of three adjacent operations, either based on the first two or the last two operations of a critical block. Example: permuting the last three operations of block no. 2 to the arc sequence $(6/1) \rightarrow (3/5) \rightarrow (6/3)$ $\Rightarrow$ TWT value of new schedule: 150.

SCEI Perturbation scheme: excluding an operation and inserting it at the most distanced position in the block so that the new schedule is feasible. Example: block no. 4, excluding (6/3) and inserting it behind (5/6) $\Rightarrow$ TWT value of new schedule: 155.

CSR Perturbation scheme: reversal of a connected sequence of operations in a critical block. Example: block no. 2, reversing the arc sequence $(3/5) \rightarrow (6/3) \rightarrow (6/1)$ $\Rightarrow$ TWT value of new schedule: 155.

ECET Perturbation scheme: reversal of one or both critical arc at the start and at the end of a critical block. Example: block no. 1, reversing $(1/1) \rightarrow (1/3)$ and $(1/5) \rightarrow (2/2)$ $\Rightarrow$ TWT value of new schedule: 137.

ICT Perturbation scheme: reversal of every second arc in a critical block, starting with the first one. Example: block no. 1 $\Rightarrow$ TWT value of new schedule: 137.

CE4P Perturbation scheme: permutation of four adjacent operations, either based on the first two or the last two operations of a critical block, additional operations are the direct preceding as well as subsequent operation on the machine. Example: block no. 4, last two operations with direct predecessor and successor, permuted to the new arc sequence $(6/1) \rightarrow (5/6) \rightarrow (5/4) \rightarrow (6/3)$ $\Rightarrow$ TWT value of new schedule: 155.

DOCEI Perturbation scheme: excluding two adjacent operations and inserting them at the beginning, and respectively, at the end of the block. Example: block no. 3, excluding (3/5) and inserting it at the end, excluding (6/3) and inserting it at the beginning ⇒ TWT value of new schedule: 262.

DICEI Perturbation scheme: excluding the first and the last operation of a block, and inserting them subsequently in reversed order within the block. Example: block no. 1, inserting (1/1) in front of (1/5) and (2/2) behind (1/3) ⇒ TWT value of new schedule: 153.

BCEI+2MT Perturbation scheme: excluding an operation v and inserting it at the most distanced (but feasible) position at the end of the block (alternatively: reversing the first critical arc $v \rightarrow w$); additionally, at most, two further reversals related to $SM(v)$ and $PM(SJ(v))$. Example: block no. 5, excluding (1/4) and inserting it behind (3/1); additionally, reversal of (2/4) → (4/3) (machine no. 1) is performed ⇒ TWT value of new schedule: 172.

Appendix B

Overview of considered Dispatching Rules

The following list presents an overview of the set of dispatching rules which have been investigated for their potential implementation in the new EGRASP (see Section 6.2.5). For each rule, the calculation of the priority value $Z(o_{ij})$ for the schedulable operation o_{ij} at time t as well as the operation $o^\star$ which has got the highest priority value within the critical set of the schedulable operations $\mathcal{SO}$ are indicated.

1. ATC - Apparent Tardiness Cost rule [139]
   measured by: $Z(o_{ij}) = (w_j/p_j) \cdot \exp\{-\max\{0, d_j - p_{ij} - t\}/(k \cdot \bar{p})\}$
   with k as look ahead parameter and value 3, $\bar{p}$ as the average processing time
   highest priority for $o^\star = \text{argmax}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

2. COVERT - Cost Over Time rule [29]
   measured by: $$Z(o_{ij}) = \begin{cases} 1/p_{ij}, & \text{if } t \geq u_j \\ (t - n_i)/(p_{ij} \cdot (u_i - n_i)), & \text{if } n_j \leq t < u_j \\ 0, & \text{if } t < n_j \end{cases}$$
   with $n_j = d_j - k \cdot p_{ij}$ and $u_j = d_j - p_{ij}$.
   highest priority for $o^\star = \text{argmax}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

3. DD+PT+WT - Rule combining Due Date, Processing Time and Waiting Time
   measured by: $Z(o_{ij}) = d_j + p_{ij} + (s_{ij} - c_{(i-1)j})$
   highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

4. ECT - Earliest Completion Time rule
   measured by: $Z(o_{ij}) = c_{ij}$
   with c_{ij} as the completion time of o_{ij} if it is scheduled next
   highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

5. EDD - Earliest Due Date rule [65]
   measured by: $Z(o_{ij}) = d_j$
   highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

6. FCFS - First Come First Served rule
   measured by: $Z(o_{ij}) = c_{(i-1)j}$
   with $c_{(i-1)j}$ as the completion time of the predecessor $PM(o_{ij})$
   highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

7. LPT - Longest Processing Time rule
   measured by: $Z(o_{ij}) = p_{ij}$
   highest priority for $o^\star = \text{argmax}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

8. MDD - Modified Due Date rule [10]
   measured by: $Z(o_{ij}) = \max\{t + p_{ij}, d_j\}$
   highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

9. ODD - Operational Due Date rule [67]
   measured by: $Z(o_{ij}) = r_j + c \cdot \sum_{k=1}^{i-1} p_{ij}$
   with c as allowance factor with value 1
   highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

10. PT+PW - Rule combining Processing Time and Waiting Time [66]
    measured by: $Z(o_{ij}) = p_{ij} + (s_{ij} - c_{(i-1)j})$
    with $c_{(i-1)j}$ as the completion time of the predecessor $PM(o_{ij})$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

11. PT+PW+ODD - Rule combining Processing Time, Waiting Time and Operational Due Date [66]
    measured by: $Z(o_{ij}) = p_{ij} + (s_{ij} - c_{(i-1)j}) + (r_j + c \cdot \sum_{k=1}^{i-1} p_{ij})$
    with $c_{(i-1)j}$ as the completion time of the predecessor $PM(o_{ij})$ and c as allowance factor with value 1
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

12. PT+WINQ+SL - Rule combining Processing Time, Work In Next Queue and Slack rule [63]
    measured by: $Z(o_{ij}) = p_{ij} + W_{(i+1)j} + \min\{d_j - t - (\sum_{k=i}^{m} p_{ij}), 0\}$
    with $W_{(i+1)j}$ as the current total work content on the next machine
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

13. SL - Slack rule
    measured by: $Z(o_{ij}) = d_j - t - (\sum_{k=i}^{m} p_{ij})$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

14. SPT - Shortest Processing Time rule
    measured by: $Z(o_{ij}) = p_{ij}$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

15. SPT+S/RPT - Rule combining SPT and Slack per Remaining Processing Time [5]
    measured by: $Z(o_{ij}) = \max\{p_{ij}, p_{ij} \cdot ((d_j - t)/\sum_{k=i}^{m} p_{kl})\}$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

16. S/OPN - Slack per number of operations [33]
    measured by: $Z(o_{ij}) = (d_j - t - (\sum_{k=i}^{m} p_{ij}))/(m - i)$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

17. WCR - Weighted Critical Ratio rule [68]
    measured by: $Z(o_{ij}) = (d_j - t)/(w_j \cdot p_{ij})$
    highest priority for $o^\star = \text{argmax}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

18. WEDD - Weighted Earliest Due Date rule [68]
    measured by: $Z(o_{ij}) = d_j/w_j$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

19. WI - Weight rule
    measured by: $Z(o_{ij}) = w_j$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

20. WINQ - Work In Next Queue rule [60]
    measured by: $Z(o_{ij}) = W_{(i+1)j} = \sum_{k=1}^{m} \sum_{l=1}^{n} o'_{kl}$
    with $\mu_{kl} = \mu_{(i+1)j},\ o'_{kl} \in \mathcal{SO}$ and $PM(o'_{kl}) \notin \mathcal{SO}$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

21. WMDD - Weighted Modified Due Date rule [68]
    measured by: $Z(o_{ij}) = (1/w_j) \cdot \max\{p_{ij}, d_j - t\}$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

22. WRA - Weighted Remaining Allowance rule [68]
    measured by: $Z(o_{ij}) = (d_j - t)/w_j$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

23. WSL+WSPT - Rule combining Weighted Slack rule and Weighted Shortest Processing Time [105]
    measured by: $Z(o_{ij}) = \max\{(d_j - t - (\sum_{k=i}^{m} p_{ij}))/\bar{w} + p_{ij}/w_j, p_{ij}/w_j\}$
    with $\bar{w}$ as the average weight attached to the jobs
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

24. WSPT - Weighted Shortest Processing Time rule [122]
    measured by: $Z(o_{ij}) = p_{ij}/w_j$
    highest priority for $o^\star = \text{argmin}\{Z(o_{ij}) \mid \forall o_{ij} \in \mathcal{SO}\}$

Appendix C

Computational Results from the Literature

The following papers present solution procedures for solving the JSPTWT. All of them are assessed by using the benchmark set of Singer and Pinedo [121]:

- SK (2000): Large Step Random Walk of Kreipl [76].
  5 runs, each 200s long, on a PC with 233 Mhz. Mean and best value of the 5 runs are reported.

- EMD (2008): Genetic Local Search of Essafi, Mati and Dauzére-Pérès [41].
  10 runs, each 18s long, on a PC with 2.8 Ghz. Mean and best value of the 10 runs are reported.

- MDL (2011): Iterated Local Search of Mati, Dauzére-Pérès and Lahlou [91].
  10 runs, each 18s long, on a PC with 2.6 Ghz. Mean and best value of the 10 runs are reported.

- KB (2011): Shifting Bottleneck-Tabu Search Heuristic of Bülbül [19].
  1 run, on average 280s long, on a PC with 2.4 Ghz. Best values are reported.

- ZW (2012): Probabilistic Model-building Genetic Alg. of Zhang and Wu [152].
  20 runs, each 60s long, no information about PC power. Mean and number of best obtained values in 20 runs are reported.

- GGVV (2012): Hybrid Evolutionary Algorithm of González, González-Rodriguez, Vela and Varela [57].

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

10 runs, each 18s long, on a PC with 2.66 Ghz. Mean and best value of the 10 runs are reported.

With the exception of ZW (2012), all results are based on a fair comparison regarding the usage of computational resources and computation time respectively. Furthermore, the assessment of the results of SK (2000) relied on best known solutions (see [76]) which have been excelled at a later date. Therefore, a comparison with SK (2000) cannot be provided.

	TWT	SK (2000)		EMD (2008)		MDL (2011)		KB (2011)	ZW (2012)		GGVV (2012)	
Instance	BKS	mean	best	mean	best	mean	best	best	mean	best	mean	best
abz05	1403	1451	—	*	*	1414	*	1462	1415	*	1412	*
abz06	436	*	*	*	*	*	*	*	*	*	*	*
ft10	1363	1368	—	1372	*	*	*	*	1375	*	1383	*
la16	1169	1170	—	1175	*	*	*	*	1170	*	*	*
la17	899	*	—	*	*	*	*	*	900	*	*	*
la18	929	*	*	933	*	*	*	*	943	*	*	*
la19	948	951	*	949	*	934	*	955	951	*	998	*
la20	805	809	—	*	*	*	*	*	822	*	834	*
la21	463	*	—	*	*	*	*	*	467	*	*	*
la22	1064	1086	n.f.	1087	*	1077	*	1084	1096	*	1079	*
la23	835	875	n.f.	865	*	865	*	877	838	*	870	*
la24	835	*	*	*	*	*	*	*	844	*	*	*
orb01	2568	2616	*	2651	2570	2639	*	2630	2613	*	2578	*
orb02	1408	1434	n.f.	1444	*	1426	*	*	1441	*	1426	*
orb03	2111	2204	n.f.	2170	*	2158	*	2115	2133	*	2160	*
orb04	1623	1674	*	1643	*	1690	*	1652	1637	*	1632	*
orb05	1593	1667	n.f.	1659	*	1775	1738	*	1617	*	1615	*
orb06	1790	1802	—	*	*	1793	*	*	1836	*	1854	*
orb07	590	618	n.f.	592	*	*	*	616	592	*	*	*
orb08	2429	2554	n.f.	2522	2439	2523	2461	2453	2483	—	2477	*
orb09	1316	1334	*	*	*	*	*	*	1317	*	*	*
orb10	1679	1775	n.f.	1718	*	1774	*	1801	1684	*	1731	*
TotalGap [in %]		—	—	1.67	0.04	2.25	0.63	1.40	1.25	—	1.52	0.00
# BKS (22)			—		20		20	12		21		22

Tab. C.1: Comparison of the computational results for the benchmark set of Singer and Pinedo ($f = 1.3$).

	TWT	SK (2000)		EMD (2008)		MDL (2011)		KB (2011)	ZW (2012)		GGVV (2012)	
Instance	BKS	mean	best	mean	best	mean	best	best	mean	best	mean	best
abz05	69	70	*	*	*	*	*	*	*	*	*	*
abz06	0	*	*	*	*	*	*	*	*	*	*	*
ft10	394	414	*	*	*	*	*	*	394	*	*	*
la16	166	*	*	*	*	*	*	*	*	*	*	*
la17	260	*	*	*	*	*	*	*	*	*	*	*
la18	34	*	*	*	*	*	*	*	35	*	*	*
la19	21	*	*	*	*	*	*	23	*	*	*	*
la20	0	*	*	*	*	*	*	1	*	*	0	*
la21	0	*	*	*	*	*	*	1	*	*	*	*
la22	196	*	*	*	*	*	*	*	*	*	*	*
la23	2	*	*	*	*	*	*	*	*	*	*	*
la24	82	90	*	86	*	86	*	*	82	*	88	*
orb01	1098	1143	n.f.	1159	*	1247	1196	1202	1108	*	*	*
orb02	292	*	*	*	*	*	*	*	*	*	*	*
orb03	918	965	*	943	*	961	952	928	923	*	939	*
orb04	358	*	*	394	*	435	*	*	360	*	*	*
orb05	405	455	*	*	*	415	*	*	405	*	428	*
orb06	426	*	*	440	*	437	*	*	427	*	430	*
orb07	50	119	n.f.	55	*	*	*	*	50	*	*	*
orb08	1023	1138	n.f.	1059	*	1036	*	*	1032	*	1033	*
orb09	297	*	*	311	*	299	*	*	309	*	302	*
orb10	346	408	*	400	*	436	424	424	377	*	430	*
TotalGap [in %]		—	—	3.87	0.00	6.20	3.26	3.03	1.10	0.00	2.38	0.00
# BKS (22)			—		22		19	17		22		22

Tab. C.2: Comparison of the computational results for the benchmark set of Singer and Pinedo ($f = 1.5$).

	TWT	SK (2000)		EMD (2008)		MDL (2011)		KB (2011)	ZW(2012)		GGVV (2012)	
Instance	BKS	mean	best	mean	best	mean	best	best	mean	best	mean	best
abz05	0	*	*	*	*	*	*	*	*	*	*	*
abz06	0	*	*	*	*	*	*	*	*	*	*	*
ft10	141	144	*	162	*	152	*	155	143	*	145	*
la16	0	*	*	*	*	*	*	*	*	*	*	*
la17	65	*	*	*	*	*	*	*	*	*	*	*
la18	0	*	*	*	*	*	*	*	*	*	*	*
la19	0	*	*	*	*	*	*	*	*	*	*	*
la20	0	*	*	*	*	*	*	*	*	*	*	*
la21	0	*	*	*	*	*	*	*	*	*	*	*
la22	0	*	*	*	*	*	*	*	*	*	*	*
la23	0	*	*	*	*	*	*	*	*	*	*	*
la24	0	*	*	*	*	*	*	*	*	*	*	*
orb01	566	624	n.f.	688	604	653	604	619	589	*	592	*
orb02	44	*	*	*	*	*	*	52	45	*	*	*
orb03	422	441	*	514	*	463	*	461	425	*	434	*
orb04	66	*	*	78	*	68	*	*	66	*	*	*
orb05	163	174	*	181	181	176	*	181	166	*	176	*
orb06	28	31	—	*	*	*	*	31	29	*	*	*
orb07	0	*	*	*	*	*	*	*	*	*	*	*
orb08	621	658	*	669	646	643	*	672	633	*	639	*
orb09	66	*	*	83	*	80	*	*	67	*	*	*
orb10	76	97	n.f.	142	84	117	78	78	83	*	82	*
TotalGap [in %]		—	—	17.54	3.94	10.23	1.77	8.33	2.35	0.00	3.99	0.00
# BKS (22)			—		18		20	14		22		22

Tab. C.3: Comparison of the computational results for the benchmark set of Singer and Pinedo ($f = 1.6$).

Appendix D

Analysis of the EGRASP result for the problem instance orb08 ($f = 1.6$)

All computational tests with EGRASP have delivered the result that the instance orb08 represents the hardest problem instance in the benchmark set of Singer and Pinedo [107]. Even though EGRASP receive more computation time, it could never produce the best known solution for the problem instance orb08 ($f = 1.6$). Therefore, this appendix provides an analysis for finding the reasons for this phenomenon.

The best known solution published in literature has an objective function value of TWT = 621 which corresponds to the globally optimal solution[20]. However, there are more than 20 different global optima which differ to each other by only few precedence relations. This observation clarifies that the set of global optima is located in a small region of the search space, also referred to as big valley. The average Hamming distance among the global optima is 3.3. The central global optimum denotes the solution with minimum distance to all other global optima. Tab. D.1 shows the job orders on the machines which are given in this central global optimum. Note that the precedence relations in a gray-colored box could be different in other global optima.

The best solution found by EGRASP has a TWT value of 646. The job orders of this best found solution are presented in Tab. D.2. The number of different precedence relations between central global optimum and best found solution is 57. The closest global optimum from the set of best known solutions has a Hamming

[20]The best solution delivered by CPLEX 12.2 has an objective function value of 621.

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

Mach.	Job order
1	$9 \to 6 \to 2 \to 1 \to 7 \to 4 \to 5 \to 8 \to 10 \to 3$
2	$6 \to 8 \to 5 \to 7 \to 4 \to 2 \to 1 \to 9 \to 3 \to 10$
3	$4 \to 8 \to 9 \to 7 \to 2 \to 6 \to 1 \to 3 \to 10 \to 5$
4	$9 \to 6 \to 2 \to 7 \to 1 \to 4 \to 8 \to 5 \to 3 \to 10$
5	$6 \to 7 \to 9 \to 8 \to 2 \to 4 \to 5 \to 1 \to 3 \to 10$
6	$6 \to 2 \to 9 \to 8 \to 1 \to 4 \to 3 \to 5 \to 7 \to 10$
7	$6 \to 9 \to 2 \to 7 \to 4 \to 1 \to 5 \to 3 \to 8 \to 10$
8	$2 \to 9 \to 4 \to 7 \to 8 \to 1 \to 6 \to 3 \to 5 \to 10$
9	$6 \to 9 \to 2 \to 8 \to 1 \to 4 \to 5 \to 3 \to 10 \to 7$
10	$6 \to 9 \to 2 \to 7 \to 8 \to 4 \to 1 \to 5 \to 3 \to 10$

Tab. D.1: Job orders in the central global optimum.

distance of 53. The comparison of the two critical trees (between central global optimum and best found solution) shows that already 27.2% of the critical arcs in the central global optimum are also critical in the best found solution, and 54.5% of the critical arcs are present as precedence relations.

However, the measured distance illustrates that at least 53 precedence relations has to be reversed in order to transfer the best found solution into one global optimum. In terms of the neighborhood search process, the local search procedure has to perform e. g. the CET neighborhood 53 times. Since the improvement phase relies on the first descent walk scheme, the neighborhood operator has to produce subsequently improving schedules within 53 local search iterations. However, regarding the average number of improving steps in Tab. 4.6, this improvement process is a priori very unlikely.

One essential strategy of the algorithmic concept of EGRASP is the identification

Mach.	Job order
1	$9 \to 6 \to 2 \to 1 \to 4 \to 8 \to 7 \to 10 \to 5 \to 3$
2	$8 \to 6 \to 4 \to 7 \to 2 \to 1 \to 5 \to 9 \to 3 \to 10$
3	$8 \to 4 \to 9 \to 7 \to 2 \to 6 \to 1 \to 10 \to 3 \to 5$
4	$9 \to 6 \to 2 \to 8 \to 4 \to 1 \to 7 \to 10 \to 5 \to 3$
5	$6 \to 8 \to 9 \to 7 \to 4 \to 2 \to 1 \to 5 \to 10 \to 3$
6	$6 \to 8 \to 1 \to 9 \to 2 \to 4 \to 10 \to 3 \to 5 \to 7$
7	$6 \to 9 \to 4 \to 1 \to 2 \to 7 \to 8 \to 5 \to 3 \to 10$
8	$4 \to 9 \to 2 \to 8 \to 1 \to 6 \to 7 \to 10 \to 3 \to 5$
9	$6 \to 8 \to 9 \to 1 \to 2 \to 4 \to 10 \to 5 \to 3 \to 7$
10	$6 \to 9 \to 8 \to 2 \to 4 \to 1 \to 7 \to 5 \to 10 \to 3$

Tab. D.2: Job orders in the best found solution of EGRASP.

of the most promising regions in the search space. The aim of the Configuration & Parameter Adjustment Procedure is to determine these regions after the first scanning of the search space in the learning phase of EGRASP. However, the mentioned distance between global optima and best found solution implies that the Configuration & Parameter Adjustment Procedure guides the search into other regions than the big valley. This conclusion is also confirmed by the distance of the four selected dispatching rules to the central global optimum. The four selected dispatching rules in the EGRASP process are WSL+WSPT, PT+PW+ODD, EDD and WMDD. Based on one dispatching rule, its deterministic solution denotes the solution in which the next schedulable operation within the construction process is the one with highest priority. The distance between deterministic solution and the central global optimum is 154 for WSL+WSPT, 131 for PT+PW+ODD, 105 for EDD and 146 for WMDD. Unfortunately, the Configuration & Parameter Adjustment Procedure rejects the ODD rule whose deterministic solution has the least distance with 79. These distance values evidence that the search process starts in regions which are far away from the favorable big valley. Even though the solutions generated in the construction phase are stochastic, there are located in an area around its deterministic solution. The average Hamming distance in such a pool of start schedules is 114.8. Due to the measured distance, the search process is never started close enough to the big valley.

The function of the Amplifying Procedure is to intensify the search further on. As a result of the mislead process of the Configuration & Parameter Adjustment Procedure, the Amplifying Procedure cannot produce a better solution.

The aim of the Path Relinking Procedure is to search the region between best found solution and the last found high-quality solution. Even though, EGRASP detects the considered best found solution early in the search process, the Path Relinking Procedure does not obtain the global optimum, too. The built search trajectory by path relinking does not pass through the big valley.

In order to eliminate this failure, EGRASP could incorporate another dispatching rule which creates start solutions nearby the respective big valley. Another opportunity is the modification of the Configuration & Parameter Adjustment Procedure so that the ODD rule is among the chosen dispatching rules. Furthermore, the development and implementation of another navigation strategy could help to guide the search successfully. However, all proposed modifications would be custom-made for

the problem instance orb08 ($f = 1.6$). A negative effect for solving other problem instances is not ruled out.

	o_{1j}		o_{2j}		o_{3j}		o_{4j}		o_{5j}		o_{6j}		o_{7j}		o_{8j}		o_{9j}		o_{10j}				
Job j	μ_{1j}	p_{1j}	μ_{2j}	p_{2j}	μ_{3j}	p_{3j}	μ_{4j}	p_{4j}	μ_{5j}	p_{5j}	μ_{6j}	p_{6j}	μ_{7j}	p_{7j}	μ_{8j}	p_{8j}	μ_{9j}	p_{9j}	μ_{10j}	p_{10j}	w_j	r_j	d_j
1	1	55	2	74	3	45	4	23	5	76	6	19	7	18	8	61	9	44	10	11	4	0	681
2	0	63	1	43	2	51	3	18	4	42	7	11	6	29	5	52	9	29	8	88	4	0	681
3	2	88	1	31	0	47	4	10	3	62	5	60	6	58	7	29	8	52	9	92	2	0	846
4	2	16	1	71	0	55	4	55	3	9	7	49	6	83	5	54	9	7	8	57	2	0	729
5	1	7	0	41	4	92	3	94	2	46	6	79	5	34	9	38	8	8	7	18	2	0	731
6	1	25	0	5	4	89	3	94	2	14	6	94	5	20	9	23	8	44	7	39	2	0	715
7	1	24	2	21	4	47	0	40	3	94	6	71	7	89	9	75	5	97	8	15	2	0	916
8	1	5	2	7	4	74	0	28	3	72	5	61	7	9	8	53	9	32	6	97	2	0	700
9	0	34	2	52	3	37	4	6	1	94	6	6	7	56	9	41	5	5	8	16	1	0	555
10	0	77	2	74	3	82	4	10	1	29	5	15	7	51	8	65	9	37	6	21	1	0	737

Tab. D.3: Data to the instance orb08 ($f = 1.6$).

Appendix E

Computational Results for the JSPTWU

The following tables present performance results solving the benchmark set of Singer and Pinedo [107] with the objective of minimizing the weighted number of tardy jobs. For this purpose, EGRASP is started with each of the three settings (10/15), (4/37) and (1/148). The motivation of these tests is to provide further benchmark results for the JSPTWU.

J. Kuhpfahl, *Job Shop Scheduling with Consideration of Due Dates,*
Produktion und Logistik, DOI 10.1007/978-3-658-10292-0,

	(10/15)		(4/37)		(1/148)
Instance	mean	best	mean	best	best
abz05	6.0	6	5.0	5	5
abz06	4.0	4	4.0	4	4
ft10	10.0	10	10.0	10	10
la16	7.0	7	6.0	6	6
la17	6.0	6	6.0	6	6
la18	6.0	6	6.0	6	5
la19	6.0	6	6.0	6	6
la20	4.4	4	4.0	4	4
la21	4.0	4	4.0	4	3
la22	6.0	6	5.0	5	5
la23	7.0	7	6.0	6	6
la24	6.0	6	6.0	6	6
orb01	10.0	10	10.0	10	10
orb02	7.0	7	7.0	7	7
orb03	12.0	12	10.0	10	10
orb04	8.0	8	8.0	8	8
orb05	9.0	9	9.0	9	9
orb06	10.0	10	10.0	10	8
orb07	8.0	8	7.0	7	7
orb08	10.0	10	10.0	10	10
orb09	7.0	7	6.0	6	6
orb10	8.0	8	8.0	8	8

Tab. E.1: Results of EGRASP for the benchmark set of Singer and Pinedo with TWU objective ($f = 1.3$), after consuming 148 seconds computation time.

	(10/15)		(4/37)		(1/148)
Instance	mean	best	mean	best	best
abz05	1.0	1	1.0	1	1
abz06	0.0	0	0.0	0	0
ft10	2.0	2	2.0	2	2
la16	2.0	2	2.0	2	2
la17	1.0	1	1.0	1	1
la18	2.0	2	1.8	1	1
la19	1.0	1	1.0	1	1
la20	0.0	0	0.0	0	0
la21	0.3	0	0.0	0	0
la22	2.0	2	2.0	2	2
la23	1.0	1	1.0	1	1
la24	1.0	1	1.0	1	1
orb01	6.0	6	6.0	6	6
orb02	3.0	3	3.0	3	2
orb03	6.0	6	6.0	6	6
orb04	3.2	3	3.0	3	3
orb05	4.0	4	4.0	4	4
orb06	6.0	6	4.0	4	4
orb07	3.0	3	2.0	2	2
orb08	7.0	7	6.8	6	6
orb09	2.0	2	2.0	2	2
orb10	4.0	4	3.0	3	3

Tab. E.2: Results of EGRASP for the benchmark set of Singer and Pinedo with TWU objective ($f = 1.5$), after consuming 148 seconds computation time.

	(10/15)		(4/37)		(1/148)
Instance	mean	best	mean	best	best
abz05	0.0	0	0.0	0	0
abz06	0.0	0	0.0	0	0
ft10	2.0	2	2.0	2	2
la16	0.0	0	0.0	0	0
la17	1.0	1	1.0	1	1
la18	0.0	0	0.0	0	0
la19	0.0	0	0.0	0	0
la20	0.0	0	0.0	0	0
la21	0.0	0	0.0	0	0
la22	0.0	0	0.0	0	0
la23	0.0	0	0.0	0	0
la24	0.0	0	0.0	0	0
orb01	4.0	4	4.0	4	4
orb02	2.0	2	2.0	2	2
orb03	4.0	4	4.0	4	4
orb04	2.0	2	1.8	1	1
orb05	3.0	3	3.0	3	2
orb06	2.0	2	1.0	1	1
orb07	0.0	0	0.0	0	0
orb08	6.0	6	5.0	5	5
orb09	1.0	1	1.0	1	1
orb10	1.0	1	1.0	1	1

Tab. E.3: Results of EGRASP for the benchmark set of Singer and Pinedo with TWU objective ($f = 1.6$), after consuming 148 seconds computation time.