Bergische Universität Wuppertal
Fakultät für Wirtschaftswissenschaft
Schumpeter School of Business and Economics

Optimizing Production and Outbound Distribution Decisions with Fixed Departure Times: Static and Dynamic Scheduling Approaches

Dissertation zur Erlangung des akademischen Grades
Doktor der Wirtschaftswissenschaft
vorgelegt von
David Bachtenkirch, M.Sc.
Wuppertal, März 2021

Betreuer: Prof. Dr. Stefan Bock / Prof. Dr. Dirk Briskorn

The PhD thesis can be quoted as follows:

urn:nbn:de:hbz:468-20220207-104523-5
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20220207-104523-5]

DOI: 10.25926/ms7h-jw42
[https://doi.org/10.25926/ms7h-jw42]

Acknowledgments

This dissertation resulted during my role as a scientific assistant from February 2015 to March 2021 at the Department of Business Computing and Operations Research (Lehrstuhl für Wirtschaftsinformatik und Operations Research [WINFOR])) at the University of Wuppertal in Germany.

First and foremost, I am incredibly grateful to my supervisor, Prof. Dr. Stefan Bock, for supervising my dissertation, always having time to discuss research ideas, and giving me direction in writing this dissertation. Furthermore, I would like to thank Stefan for being a great boss that created a very relaxed and productive work environment, which made me enjoy going to the office every day. I immensely enjoyed all the on- and off-topic conversations we had in the past six years.

Furthermore, I would like to thank Prof. Dr. Dirk Briskorn for being my second supervisor and reviewing this dissertation.

I would also like to thank all of my colleagues at WINFOR, Volker Arendt, Rudolf Bauer, Paul Göpfert, and Anna Katharina Janiszczak, for creating a great work environment in the past six years. Moreover, I would like to thank Rudolf Bauer for supervising my master's seminar papers and final thesis and recommending me for the open position at WINFOR. Mostly, I would like to thank Paul Göpfert for the regular discussions on research and work-related topics, sharing programming ideas, and all the conversations around teaching at the faculty and life in general.

Above all, I would like to wholeheartedly thank my mother for her love and support throughout my whole life. You always believed in me and encouraged me to find success.

Last but not least, I would also like to thank my beloved wife Nadia and our three children for their love and for giving me happiness in the past years and for many years to come.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Notation	Description	First page
3PL	third-party logistics	2
ATS	assemble-to-stock	1
B&B	branch-and-bound	7
B&C	branch-and-cut	14
B&P	branch-and-price	14
BFRS	breadth-first search	63
BFS	best-first search	64
BTO	build-to-order	1
BTO-SC	build-to-order supply chain	1
CG	column generation	14
CL	candidate list	131
CP	cutting plane	14
CWE	constrained weighted earliness	37
D-SFDDHT	scheduling problem with flexible customer-dependent delivery dates and dynamic job arrivals that pursues the minimization of holding and transportation costs	47
DFS	depth-first search	63
DP	dynamic programming	15
EDD	earliest due date	41
ERD	earliest release date	99
FCFS	first-come-first-served	154
FIFO	first-in-first-out	63
FPTAS	fully polynomial-time approximation scheme	19

Notation	Description	First page
GA	genetic algorithm	17
GRASP	greedy randomized adaptive search procedure	7
ILP	integer linear programming problem	14
IPODS	integrated production and outbound distribution scheduling	2
IPODS-FD	integrated production and outbound distribution scheduling with fixed delivery departure dates	3
JIT	just-in-time	35
LAP	linear assignment problem	110
LIFO	last-in-first-out	63
LP	linear programming problem	13
LS	local search	16
MH	metaheuristic	16
MILP	mixed-integer linear programming problem	7
MTO	make-to-order	1
MTS	make-to-stock	1
OR	operations research	3
PTAS	polynomial-time approximation scheme	19
RCL	restricted candidate list	131
RTC	real-time control	3
SA	simulated annealing	17
SCM	supply chain management	4
SCP	(unweighted) minimal set covering problem	117
SFDDHT	scheduling problem with flexible customer-dependent delivery dates that pursues the minimization of holding and transportation costs	47
SFDDHT-B&B	B&B algorithm for the SFDDHT	61
SFDDHT-GRASP	GRASP algorithm for the SFDDHT	130
SPT	shortest processing time	42
SRPT	shortest remaining processing time	42

Notation	Description	First page
TS	tabu search	16
TSPTW	traveling salesman problem with time windows	133
TU	time units	76
WE	weighted earliness	35
WSPT	weighted shortest processing time	120

List of Symbols

Notation	Description	First page
$\breve{a}_j$	Arrival time of job $j \in \check{N}$ in the D-SFDDHT	145
A^{scp}	Matrix that defines the subsets in the SCP	117
$a^{scp}_{i,j}$	Entry of the m-th row and n-th column in matrix A^{scp} in the SCP	117
$B(\zeta)$	Final branching decisions for schedule ζ	108
$B^0(\zeta)$	Initial branching decisions for schedule ζ	105
c^H_j	Holding cost of job j in a IPODS / the SFDDHT	52
$\check{c}^H_j$	Holding cost of job $j \in \check{N}$ in the D-SFDDHT	147
$\check{C}_j$	Production completion time of job $j \in \check{N}$ in the D-SFDDHT	147
$c^{lap}_{i,j}$	Cost of assigning element $i \in M$ to target $j \in M$ in the LAP	110
$C^{min}(\mathcal{S}, j, i)$	Minimal completion time of job j scheduled at position i in schedule $\mathcal{S}$	135
$\bar{C}^{max}$	Maximal completion time of any unscheduled job $j \in \bar{J}(\zeta)$	79
C_j	Completion time of job j in a scheduling problem / the SFDDHT	52
$C^{b,min}_{N^*_j,p}$	Makespan of the back schedule with $n^N - p$ jobs from set N^*_j	102
C^e_j	Earliest completion time of a job $j \in N$ in the SFDDHT	96
$C^{f,min}_{N^*_j,p}$	Makespan of the front production schedule with $p - 1$ jobs from set N^*_j	102
C^l_j	Latest completion time of a job $j \in N$ in the SFDDHT	96
$C^{max}_{j,p}$	Upper bound for the maximal completion time of job $j \in N$ at position $p \in N$ in the SFDDHT	102

Notation	**Description**	**First page**
$\bar{C}_j^{max}(\zeta)$	Maximal completion time of unscheduled job $j \in \bar{J}(\zeta)$	103
$\bar{C}^{min}(\bar{J}^*(\zeta))$	Lower bound of the makespan of the remaining jobs without job j	104
$C_{j,p}^{min}$	Lower bound for the minimal completion time of job $j \in N$ at position $pinN$ in the SFDDHT	102
$\bar{C}_j^{max}(\zeta)$	Minimal completion time of unscheduled job $j \in \bar{J}(\zeta)$	104
c_g^T	Transportation cost for a transport to customer $g \in G$ in the SFDDHT	52
$\breve{c}_g^T$	Fixed transportation cost for a transport to customer $g \in \breve{G}$ in the D-SFDDHT	147
$C(U_i)$	Set of maximally chosen completion times to schedule $n(U_i)$ jobs that complete in interval $U_i \in U(P)$ (LB I)	113
$C_{i,p}^U$	Maximally chosen completion time to schedule a job of set $N'(U_i)$ at the p-th position before time u_i^l (LB I)	113
D	Dominance relation between two subproblems $\mathcal{X}_1$ and $\mathcal{X}_2$ where $\mathcal{X}_1 D \mathcal{X}_2$ allows $\mathcal{X}_2$ to be pruned.	66
$\bar{d}_j$	Production deadline of job j in a scheduling problem and simultaneously latest delivery departure time in the SFDDHT	51
$\bar{d}^{max}$	Maximal deadline $\bar{d}_j$ in the SFDDHT	102
D_j	Delivery departure time of job j in an IPODS / the SFDDHT	52
$D_{j,\tau}(\zeta)$	Initial set of delivery departure times for a candidate job $j \in \bar{J}^b(\zeta)$ that completes at $C_j = \tau$ given schedule ζ	106
$D_{j,\tau}^1(\zeta)$	First subset of delivery departure times for a candidate job $j \in \bar{J}^b(\zeta)$ that completes at $C_j = \tau$ given schedule ζ	107
$D_{j,\tau}^2(\zeta)$	Second subset of delivery departure times for a candidate job $j \in \bar{J}^b(\zeta)$ that completes at $C_j = \tau$ given schedule ζ	107
$D_{j,\tau}^3(\zeta)$	Third subset of delivery departure times for a candidate job $j \in \bar{J}^b(\zeta)$ that completes at $C_j = \tau$ given schedule ζ	107
$D_{j,\tau}^*(\zeta)$	Final set of delivery departure times for a candidate job $j \in \bar{J}^b(\zeta)$ that completes at $C_j = \tau$ given schedule ζ	108
Δ	Set of delivery batches of schedule ζ	73

Notation	Description	First page
$\Delta^{\bullet}(g)$	Set of nonempty delivery batches of customer g defined by schedule ζ	91
$\Delta(g)$	Set of delivery batches for customer g of schedule ζ	90
$\Delta^{c}(g)$	Set of closed delivery batches for customer g in a schedule ζ	90
$\Delta_{g,l}$	The l-th delivery batch for customer g of schedule ζ	90
$\Delta^{o}(g)$	Set of open delivery batches for customer g in a schedule ζ	90
Δ_{l}	The l-th delivery batch of schedule ζ	73
$\delta_{l}(j)$	The j-th job of the l-th delivery batch of schedule ζ	73
$D^{e}(\mathcal{S}, j, i)$	Earliest delivery departure time of job j scheduled at position i in schedule $\mathcal{S}$	135
$\check{D}_{j}$	Delivery departure time of job $j \in \check{N}$ in the D-SFDDHT	146
$\check{d}_{j}$	Production and delivery deadline of job $j \in \check{N}$ in the D-SFDDHT	151
$D^{l}(\mathcal{S}, j, i)$	Latest delivery departure time of job j scheduled at position i in schedule $\mathcal{S}$	135
$\prec_{\mathrm{ERD}}$	Precedence relation defined by the ERD ordering	99
$\mathcal{E}$	Set of arcs of a B&B search tree $\mathcal{T}$	61
$f(x)$	The objective function of an optimization problem $\mathcal{P}$ that maps a feasible solution x to a value	61
$f(s)$	Objective function for an optimization problem	131
$f^{*}(\mathcal{X})$	The optimal objective function value of an optimization problem $\mathcal{P}$	61
G	Set of customers in the SFDDHT	51
Γ	Set of production blocks of schedule ζ	74
Γ^{1}	The first nonempty production block of schedule ζ	88
$\gamma_{l}(j)$	The j-th job of the l-th production block defined by schedule ζ	74
Γ_{l}	The l-th production block of schedule ζ	74
$g^{N}(j)$	Customer of job $j \in N$ in the SFDDHT	51
$\check{G}$	Set of customers in the D-SFDDHT	145
$g^{T}(l)$	Customer of transport $l \in T$ in the SFDDHT	51
$h(\mathcal{X}_{i})$	Heuristic function that indicates the next subproblem of a B&B algorithm	63

Notation	Description	First page
$c^H(\zeta)$	Total holding cost of a (partial) schedule ζ of the SFDDHT	67
I^T_g	Set of transports to customer $g \in G$ in the SFDDHT	51
I^T	Set of transports in the SFDDHT	51
$J(\zeta)$	Set of scheduled jobs in a (partial) schedule ζ	67
$J_g(\zeta)$	Set of scheduled jobs of customer $g \in G$ in a (partial) schedule ζ of the SFDDHT	67
$\bar{J}^c_g$	Set of (remaining) critical jobs of customer g that must be assigned to a unused transport, given schedule ζ	116
$\bar{J}^{II}_g$	Set of remaining agreeable jobs for customer g (LB II)	121
J_k	Set of scheduled jobs by schedule $\mathcal{S}$ at the end of the k-th GRASP iteration	133
J_k	Set of remaining jobs at the end of the k-th GRASP iteration of SFDDHT-GRASP	133
$\bar{J}(\zeta)$	Set of remaining (unscheduled) jobs in a (partial) schedule ζ	67
$\bar{J}(g, l)$	Set of remaining jobs of customer g that can still be assigned to the l-th delivery departure time of customer g by a schedule derived from schedule ζ	89
J^n	Set of non-terminal jobs defined by schedule ζ	74
$\bar{J}^*_j$	Set of remaining jobs without job j	103
$\bar{J}^b(\zeta)$	Set of candidate branching jobs for schedule ζ	106
$\bar{J}^s_j$	Set of remaining feasible successor jobs for job $j \in \bar{J}(\zeta)$ (LB III)	124
j^*_k	Job $j \in N$ with the k-th largest processing time window length considered in the k-th GRASP iteration of SFDDHT-GRASP	133
J^t	Set of terminal jobs defined by schedule ζ	74
$\bar{J}^{III,t}_g$	Set of jobs of customer g classified as terminal (LB III)	125
j^t_j	Earliest terminal successor of a non-terminal job defined by schedule ζ	74
$j^U_{i,j}$	The job of set $N'(U_i) \subseteq N(U_i)$ with $U_i \in U(P)$ with the j-th smallest processing time (LB I)	113
$k(\zeta)$	First assigned position in schedule ζ	67
$\mathcal{L}$	Active list of unexplored subproblems of a B&B algorithm	62

Notation	Description	First page
LB	Lower bound on the objective function value of a sub-problem	65
$LB_i(\zeta)$	Lower bound with index $i \in \{1, 2, 3\}$, given schedule ζ	109
M^{pos}	Position matrix	138
M^{prec}	Precedence matrix	97
M^{II}_g	Set of machines for jobs of customer g (LB II)	121
M^{lap}	Set of elements/targets in the LAP	110
m^{lap}	Number of elements/targets in the LAP	110
$\mathcal{M}^r$	Set of reinsertion moves of the SFDDHT-GRASP	138
M^{scp}	Set of rows of matrix A^{scp} in the SCP	117
m^{scp}	Number of rows of matrix A^{scp} in the SCP	117
N	Set of jobs in the SFDDHT	51
$\mathbb{N}_0$	Set of natural numbers with 0	156
n^{Δ}_l	Number of jobs in the l-th delivery batch defined by schedule ζ	73
$n^{\Delta}_{g,l}(\bar{J})$	Maximal delivery batch size for the l-th transport to customer g given the remaining jobs $\bar{J}$	114
$n^{D,max}_g(\zeta)$	Upper bound for the number of additional transports for customer g in a completed schedule, computed from schedule ζ	115
n^{ϵ}	Maximal number of elites stored in the elite pool by the SFDDHT-GRASP	140
N^f	Set of semi-fixed orders of an SFDDHT instance generated as a snapshot	151
N_g	Set of jobs of customer $g \in G$ in the SFDDHT	51
n^G	Number of customers in the SFDDHT	51
n^{Γ}_l	Number of jobs in the l-th production block defined by schedule ζ	74
$\check{N}_g$	Set of customers in the D-SFDDHT	147
$n^{\check{G}}$	Number of customers in the D-SFDDHT	145
n^{iters}	Iteration limit for the SFDDHT-GRASP	141
$n^J(\zeta)$	Number of scheduled jobs in a (partial) schedule ζ	67
$n^{\bar{J}}(\zeta)$	Number of remaining (unscheduled) jobs in a (partial) schedule ζ of the SFDDHT	67
n^N	Number of jobs in the SFDDHT	51
n^N_g	Number of jobs of customer $g \in G$ in a SFDDHT instance	51
$n^{\check{N}}$	Number of orders in the D-SFDDHT	145

Notation	Description	First page
n^{RCL}	Maximal size of the restricted candidate list for the SFDDHT-GRASP	141
$\breve{N}$	Set of orders in the D-SFDDHT	145
N^{scp}	Set of columns of matrix A^{scp} in the SCP	117
n^{scp}	Number of columns of matrix A^{scp} in the SCP	117
$\bar{N}^{scp}$	Set of columns that define a cover (comprise all elements) in the SCP	117
n^{T}	Number of transports in the SFDDHT	51
n^{T}_{g}	Number of transports to customer $g \in G$ in the SFDDHT	52
n^{thd}	Number of threads used by the SFDDHT-GRASP	140
$n^{T(P)}$	Number of unique delivery departure times in processing interval P (LB I)	112
$\breve{N}_{\tau}$	Set of revealed orders at time point τ in the D-SFDDHT	146
$\mathcal{N}$	Set of nodes of a B&B search tree $\mathcal{T}$	61
$\breve{N}^{u,p}_{\tau}$	Set of unfulfilled processed orders at time-point τ in the D-SFDDHT	146
$\breve{N}^{u}_{\tau}$	Set of unfulfilled revealed orders at time-point τ in the D-SFDDHT	146
$\breve{N}^{u,-}_{\tau}$	Set of unfulfilled unprocessed orders at time-point τ in the D-SFDDHT	146
$N(U_i)$	Set of completable jobs in interval $U_i \in U(P)$ (LB I)	112
$n(U_i)$	Maximal number of jointly completable jobs in interval $U_i \in U(P)$ (LB I)	112
$n^{U(P)}$	Number of intervals in set $U(P)$ (LB I)	112
N^{*}_{j}	Set of jobs N without job j	98
$\mathcal{P}$	An optimization problem	61
$\pi(i)$	Job at the i-th position of a processing sequence $\Pi(\zeta)$ in a (partial) schedule ζ	67
$\pi^{-1}(j)$	Scheduled position of job $j \in J(\zeta)$ in the processing sequence $\Pi(\zeta)$ in a (partial) schedule ζ	67
$\Pi^{b}(j, p)$	Back production schedule without job j at position p that determines the positions $p, \dots, n^{N}$	98
$\Pi^{f}(j, p)$	Front production schedule with job j at position p that determines positions $1, \dots, p$	98
P	Interval to process the remaining jobs $\bar{J}$ given schedule ζ (LB I)	111

Notation	Description	First page
p^e	Earliest time-point of processing interval P to process the remaining jobs $\bar{J}(\zeta)$ (LB I)	111
p^l	Latest time-point of processing interval P to process the remaining jobs $\bar{J}(\zeta)$(LB I)	111
Π^s_l	Alternative production sequence that swaps the jobs at positions k and l given sequence Π	83
$\check{p}_j$	Processing time of job $j \in \check{N}$ in the D-SFDDHT	151
p_j	Processing time of job j in a scheduling problem / the SFDDHT	51
p^w_j	Processing time window length of job $j \in N$ in the SFDDHT	133
$\mathbb{R}$	Set of real numbers	61
r^b_j	Equivalent release date of job j in the back production schedule $\Pi^b(j,p)$ with $j,p \in N$	102
r_j	Production release date of job j in a scheduling problem / the SFDDHT	51
$P(\zeta)$	Transportation assignments of jobs in a (partial) schedule ζ	67
$\rho(i)$	Transportation assigmnent of the job at the i-th position of a processing sequence $\Pi(\zeta)$ in a (partial) schedule ζ of the SFDDHT	67
r^{II}_i	Release time of machine i in a parallel machine problem (LB II)	121
$\mathbb{R}^+$	Set of positive real numbers	11
s	Solution for an optimization problem	131
S^1	Start time of the job at position k in schedule ζ	68
S^ϵ	Elite pool that stores elite solution during application of the SFDDHT-GRASP	140
$\mathcal{S}$	Schedule/Solution for the SFDDHT constucted by SFDDHT-GRASP	133
$\hat{s}$	Best-found solution for an optimization problem	131
S_j	Start time of job j in a scheduling problem / the SFDDHT	68
$\check{S}_j$	Production start time of job $j \in \check{N}$ in the D-SFDDHT	146
$\mathcal{S}_k$	Schedule constucted by SFDDHT-GRASP at the end of the k-th iteration	133
$\mathcal{S}^r_\tau$	Relevant schedule proposed for the anticipation horizon $[\tau, \tau + t^a]$	148

Notation	Description	First page
S_j^e	Earliest start time of a job $j \in N$ in the SFDDHT	96
S_j^l	Latest start time of a job $j \in N$ in the SFDDHT	96
t^a	Anticipation horizon length	148
$T_g^a(\zeta)$	Set of assignable delivery departure times for customer g, given schedule ζ	115
$c^T(\zeta)$	Total transportation cost of a (partial) schedule ζ of the SFDDHT	68
T_g	Set of delivery departure times for customer $g \in G$ in the SFDDHT	51
$t_{g,l}$	Delivery departure time of the transport $l \in I_g^T$ of customer $g \in G$ in the SFDDHT	52
$\check{T}_g$	Set of delivery departure times for customer $g \in \check{G}$ in the D-SFDDHT	145
T_j^N	Set of feasible delivery departure times for job j in the SFDDHT	107
t_l	Delivery departure time of the transport $l \in I^T$ in the SFDDHT	51
t^{lim}	Time limit for the SFDDHT-GRASP	141
t_g^{min}	The earliest used delivery departure time of customer $g \in G$ defined by schedule ζ	89
t_l^P	The l-th delivery departure time in processing interval P (LB I)	112
t^T	Planning horizon end in the D-SFDDHT	145
$\mathcal{T}$	A B&B search tree	61
$T_{g,j}^{u,a}(\zeta)$	Set of unused assignable delivery departure times for customer g by job $j \in \bar{J}_g$ given schedule ζ	116
$T_g^{u,a}(\zeta)$	Set of unused assignable delivery departure times for customer g, given schedule ζ	115
u_i	Interval of set $U(P)$	112
u_i^e	Earliest time-point of interval U_i of set $U(P)$ (LB I)	112
u_i^l	Latest time-point of interval U_i of set $U(P)$ (LB I)	112
$U(P)$	Set of disjoint intervals in processing interval P (LB I)	112
$V^f(\mathcal{S})$	Greedy feasibility function of SFDDHT-GRASP	138
$\mathcal{X}$	Set of feasible solutions of an optimization problem $\mathcal{P}$	61
$\mathcal{X}^c$	Set of feasible canonical schedules for a SFDDHT instance	69

Notation	**Description**	**First page**
$\mathcal{X}^{c}_{i}$	The subproblem of the i-th SFDDHT-BNB search tree node	70
$\hat{x}$	Incumbent solution of an optimization problem during the application of a B&B algorithm	62
$\check{X}_{j}$	Binary decision variable that indicates whether the order request $j \in \check{N}$ is accepted in the D-SFDDHT	146
$X^{lap}_{i,j}$	Binary assignment variable. If $X^{lap}_{i,j}$ is set to 1, the i-th target is assigned to the j-th target in the LAP	110
$\mathcal{X}^{o}$	Set of feasible solutions / schedules for a SFDDHT instance	66
X^{scp}_{j}	Binary variable that indicates whether column n is part of the cover $\bar{N}^{scp} \subseteq N^{scp}$ in the SCP	117
$\mathcal{X}^{*}$	Set of optimal solutions of an optimization problem $\mathcal{P}$	61
x^{*}	An optimal solution of an optimization problem $\mathcal{P}$	61
$z^{e}_{i}(\zeta, \bar{J})$	Cost estimate for scheduling the remaining jobs $\bar{J}$ with index $i \in \{1, 2, 3\}$, given schedule ζ	109
ζ	A (partial) schedule for an SFDDHT instance	67
ζ^{c}_{i}	The schedule of the i-th SFDDHT-BNB search tree node	71
$Z^{M}(\Pi)$	Set of alternative permutations by application of the move dominance given sequence Π	85
$z^{III,n}_{j}$	Estimated cost of classifying job $j \in \bar{J}(\zeta)$ as nonterminal (LB III)	124
$Z^{S}(\Pi)$	Set of alternative permutations by application of the swap dominance given sequence Π	83
$z^{III,*}_{j}$	Estimated minimal cost of scheduling job $j \in \bar{J}(\zeta)$ as either terminal or nonterminal (LB III)	124
$z^{III,t}_{j}$	Estimated cost of classifying job $j \in \bar{J}(\zeta)$ as terminal (LB III)	124
$z(\zeta)$	Objetive function value (total cost) of a (partial) schedule ζ of the SFDDHT	68

Chapter 1

Introduction

Different customer tastes and preferences, as well as technological advancements, characterize today's global markets. In many industries, the requirement to offer various products challenges production companies that want to compete successfully. The build-to-order supply chain (BTO-SC) or make-to-order (MTO) system involves manufacturing processes tailored to fulfill customer demands in markets with highly customizable products. High-tech companies such as Dell, BMW, Compaq, and Gateway have implemented BTO-SCs to address the uncertain variability in demand that arises from offering a vast range of products and product configurations (Gunasekaran and Ngai, 2009). For example, the Dell computer company allows end-customers to configure computer systems online and order their desired computer with a promised delivery in days. Dell's German online shop[1] offers delivery in less than six working days for most of its products. Gunasekaran and Ngai (2009) define a BTO-SC as

> ... the system that produces goods and services based on individual customer requirements in a timely and cost competitive manner by leveraging global outsourcing, the application of information technology and through the standardization of components and delayed product differentiation strategies.

The difference between MTO and build-to-order (BTO) systems is that an MTO system includes the manufacturing of components and parts alongside assembly, while a BTO system concentrates on the assembly of already manufactured items. Both manufacturing processes differ from traditional production operations by usually carrying no (long-term) inventory of finished goods and building products to order only (Wagner et al., 2003). These characteristics differentiate the considered manufacturing processes from the classic make-to-stock (MTS) or assemble-to-stock (ATS) paradigms wherein products are manufactured based on predicted future demands.

Apart from structuring the manufacturing processes and setting up the supplier network supply chain to realize an MTO system effectively, distribution decisions play a significant role in fulfilling customer orders in a timely manner and at low costs. As the manufacturer produces or assembles final products only after an incoming order arrives,

[1] Dell GmbH. Shop für Home PCs und Zubehör. Dell. 13th November 2020: https://www.dell.com/de-de/shop.

the production end-date and the ordering customer's distribution must be synchronized to ensure short delivery times and to fulfill customer needs.

Gunasekaran and Ngai (2009) emphasize that using third-party logistics (3PL) is essential to accomplish this task in BTO and MTO systems. A recent study (Langley and Infosys, 2020) highlights the general importance of 3PL. In a questionnaire, 73% of the responding companies outsourced domestic transportation, and 65% outsourced international transportation. The issue of optimizing the use of 3PL services to minimize order lead times and the total distribution costs consequently arises.

There is a widespread agreement on the importance of coordinating production and distribution decisions in the supply chain. However, the scientific research on this topic is often directed towards strategic and tactical planning levels, as demonstrated in literature reviews by Sarmiento and Nagi (1999), Erengüç et al. (1999), Goetschalckx et al. (2002), Bilgen and Ozkarahan (2004), and Chen (2004). In contrast, integrated production planning and distribution models on the operational planning level have received much less attention (Chen, 2010). This observation holds especially true for coordinating production and delivery decisions in detailed scheduling processes that involve 3PL providers. Detailed scheduling models coordinate the production and distribution on an order-by-order level to optimize metrics such as revenues, inventory and distribution costs, and customer service levels.

The difficulty of coordinating production scheduling and distribution decisions arises from various restrictions. For instance, the production and delivery of an order are constrained to specific time windows. Order production generally starts in the final production or assembly stage, depending on resource availability, which is influenced by earlier stages. Additionally, fulfilling an order is expected up to a final delivery date due to, for example, contract penalties or the perishability of products. Chen and Vairaktarakis (2005) studied the latter issue in the context of coordinating food preparation and delivery by food caterers. Other products and industries with hard time windows are, for example, time-sensitive chemical compounds (Armstrong et al., 2008; Devapriya et al., 2006) or the production and distribution of ready-mix concrete paste that hardens after a short period (García and Lozano, 2004, 2005). Furthermore, time window restrictions appear in the steel industry. As steel coils are generally bulky, and storage space at the customer site is limited, delivery is expected to occur in a pre-agreed time window (Li et al., 2017). Time window restrictions are a complicated issue for manufacturers that rely on 3PL for outbound distribution activities. The manufacturer must adhere to a pre-agreed delivery schedule that dictates when products can be collected from the manufacturing site by 3PL vehicles and shipped to customer locations. Missing a departure time consequently means storing the produced items until the next available shipment or discarding expired items.

This dissertation contributes to the research on efficiently scheduling production processes in an environment with outsourced distribution to a 3PL company with a fixed delivery schedule. In general, Chen (2010) refers to scheduling problems that coordinate production and outbound distribution as integrated production and outbound distribution scheduling (IPODS) problems. The present dissertation specifically refers to IPODS with distribution by 3PL as integrated production and outbound distribution scheduling with fixed delivery departure dates (IPODS-FD). The models and methods presented in this work cover various scenarios that are not already considered by the present literature. They consider hard time window restrictions for the production and distribution of orders, and they investigate the conflicting objective of simultaneously minimizing holding and transportation costs. The trade-off between these cost types stems from considering fixed transportation costs that are reducible by delivering products in batches, which leads to the intermediate storage of batched products and therefore to an increase in holding costs.

This dissertation addresses *static planning processes* with known orders for the considered planning horizon (e.g., a day or a week). It also addresses a *dynamic environment* with the objective of integrating dynamically arriving orders into the execution plan in real time. A specifically designed real-time control (RTC) approach schedules tasks in this real-time setting. Apart from a single study that proposes simple dispatching rules for a dynamic context, the contemporary research does not cover dynamic IPODS-FD. Therefore, the presented dynamic approach introduces an elaborate RTC approach to expand such models' research in this direction. The introductory chapter continues with Section 1.1, which briefly categorizes this dissertation's contents. The central research aim and objectives follow this categorization in Section 1.2. Finally, Section 1.3 outlines the subsequent chapters of this dissertation.

1.1 Categorization

This dissertation contributes to the field of operations research (OR), which Hillier and Lieberman (2012) outline as a scientific approach to managing organizations. This broad definition includes the following non-exhaustive list of application areas: manufacturing, transportation, construction, telecommunications, health care, and the military. As the *research* part of the name suggests, OR is driven by the application of a scientific method (Hillier and Lieberman, 2012, p.3). The following six overlapping phases summarize an OR study:

1. Analyzing a real-world problem and gathering relevant data;
2. Formulating an abstract mathematical model that attempts to explain the problem under investigation;
3. Developing computer-based methods that generate solutions to the formulated problem;
4. Testing the developed model via computational experiments and refining the model if necessary;
5. Preparing the application of the model and solution methods as demanded by management;
6. Implementing the application (Hillier and Lieberman, 2012, p. 7).

While OR at its core, is concerned with establishing mathematical results, the first and last phases highlight its practical importance of providing managerial implications for real-world problems. The topics covered in this dissertation are part of the broad area of supply chain management (SCM) that Christopher (2011, p. 3) defines as follows:

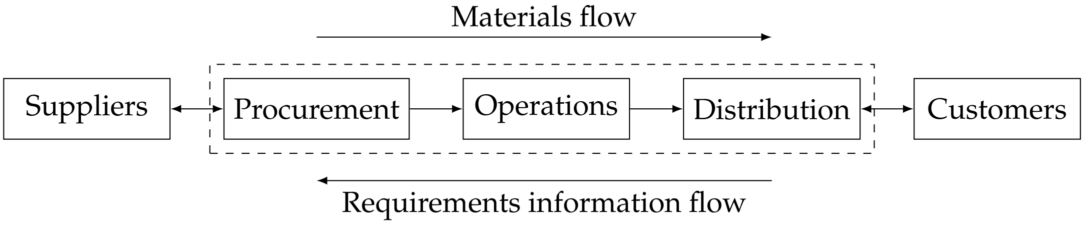
> The management of upstream and downstream relationships with suppliers and customers in order to deliver superior customer value at less cost to the supply chain as a whole.

The central aim of effectively managing a supply chain by applying OR methods is to gain a competitive advantage. This aim is achieved by gaining a cost or value advantage. On the one hand, a cost advantage means that a competitor can offer a product at a lower cost rate than others. In this context, SCM increases efficiency and productivity to reduce unit costs in various ways. On the other hand, a value advantage is gained by offering a product that customers *perceive* to be better than comparable products. The value advantage increases not only through the product itself but also through the service a company offers alongside its product. Moreover, effective SCM improves the level of service in regards to product availability, selectable product configurations, and order lead times (Christopher, 2011, pp. 5–7).

The consideration of the different company activities as an interlinked system is illustrated in Figure 1.1. Logistic activities link the individual stages of procuring raw materials or components from suppliers, producing the final products through different operations, and distributing the final products to customers. There is consequently an oppositely directed flow of information triggered by customer demands in the supply chain view.

Figure 1.1

Logistics Management Process



Note. Adapted from Christopher, M. (2011). *Logistics & supply chain management*. Pearson Education, fourth edition, p. 11. Copyright 2011 by Pearson Education Limited.

As discussed by Chen (2010), production and distribution models at an operational level have received much less attention than their counterparts on the strategic or tactical level. Chen classifies the existing models at this time into *five* classes:

1. Models with individual and immediate delivery;
2. Models with batch delivery to a single customer;
3. Models with batch delivery to multiple customers;
4. Models with batch delivery to multiple customers via a routing method;
5. Models with fixed delivery departure dates.

First, in *models with individual and immediate delivery*, the distribution of completed products occurs at the instance of completion. Models of this type differ from classic scheduling models by either considering a fixed delivery time known beforehand or specifying time windows for the jobs in which delivery must occur.

Second, models that consider *batch delivery to a single customer via a direct shipping mode* allow for the storage of produced items at the production site. Produced and stored items are bundled and delivered to a customer location. A further distinction between these types of models is in the size of orders, which are either equally sized or of different sizes.

Third, *models with batch delivery to multiple customers via a direct shipping method* extend the single customer models by enabling orders from and transportation to different locations. With direct shipping, transportation occurs only on a direct path from the production site to customer locations. As the locations differ, these types of models assume unequal transportation times and costs for different customers.

Fourth, in contrast to the previous types of models, *models with batch delivery to multiple customers via a routing method* consider routing decisions. That is, a single delivery activity is allowed to visit multiple customers. The routing problem for each delivery resembles the difficult to solve traveling salesman problem (Karp, 1972). Models that consider a limited fleet of vehicles comprise vehicle routing problem variants as subproblems. An overview of routing problems is, provided by, for example, Bodin (1981). Due to the complexity of these problems', integrated problems with routing are usually not optimally solvable with limited resources.

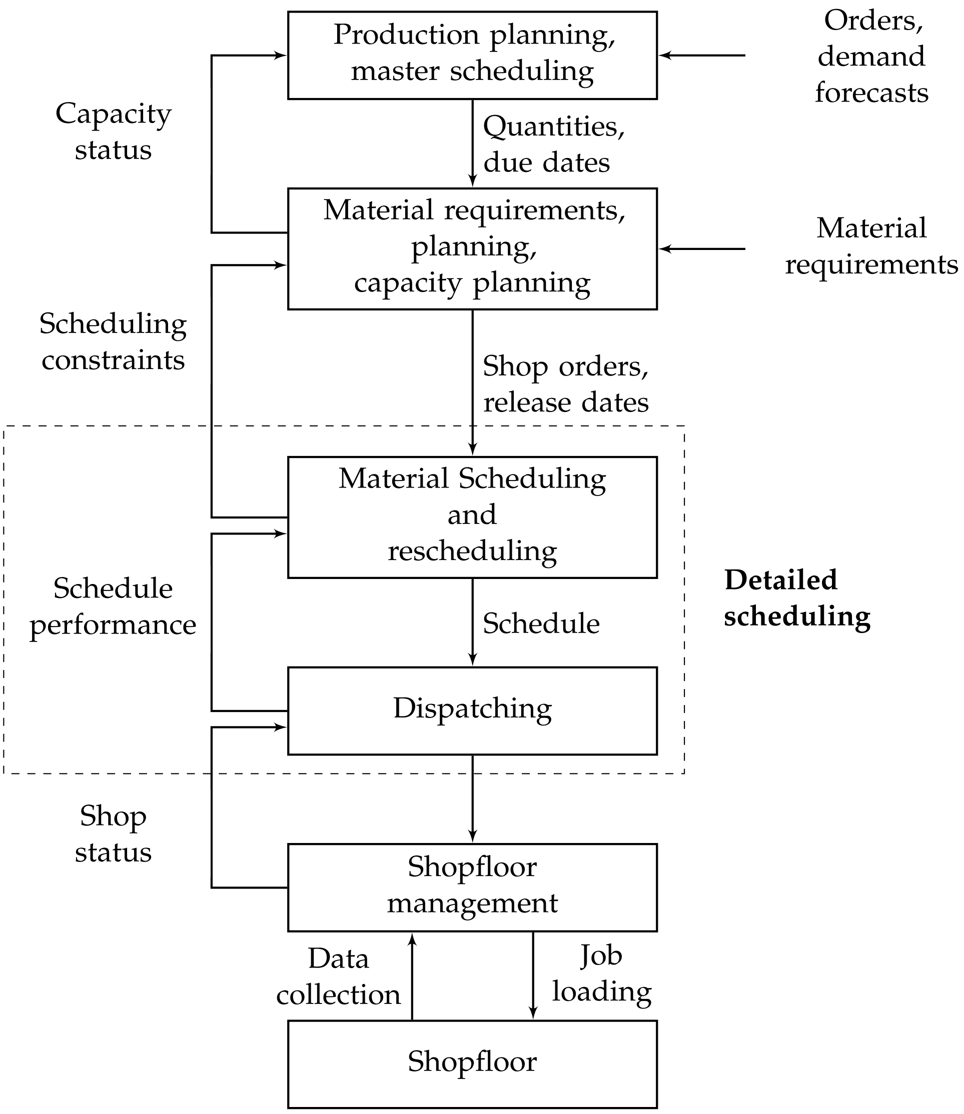
Finally, *models with fixed delivery departure dates* (IPODS-FD models) define feasible departure times as input parameters. This context assumes that a vehicle leaves the production site at specified times to carry out deliveries. These types of problems often appear in the utilization of 3PL providers, which carry out pickup requests from manufacturer at times given by a time table.

The IPODS-FD models discussed in this dissertation focus on optimizing the operations between the final production or assembly stage and the outbound distribution of the final products. In particular these models are concerned with detailed scheduling decisions. In general, scheduling is understood as a decision-making process that allocates resources to tasks within a certain time frame. Its goal is to optimize one or more objectives (Pinedo, 2016, p. 1). In the manufacturing context, resources are commonly machines, workers, and raw materials in a production environment, and tasks may refer to operations in a production process, for example the assembly of components to a final product on a machine. The objectives considered in such decision-making processes are manifold. Most objectives are related to the completion times of tasks, such as minimizing the completion time of the last task for a given planning horizon. While much of the scheduling research applies to the manufacturing context, applications exist for scheduling computer systems and transportation and distribution settings (Pinedo, 2016, pp. 1–4). Figure 1.2 illustrates the function of scheduling in the manufacturing context.

As the figure indicates, scheduling decisions are made in a hierarchical way. Earlier production planning phases determine the availability of resources, the scheduling con-

Figure 1.2

Information Flow Diagram in a Manufacturing System



Note. Adapted from Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer, p. 5. Copyright 2016 by Springer Science+Business Media, LLC.

straints, and the tasks to schedule. The scheduling process itself subsequently attempts to optimize the allocation of the selected resources and tasks under a set of constraints. In the context of the considered IPODS-FD applications, the scheduling decisions focus not only on the completion of tasks but also on distributing the finished products. Therefore, the completion times and departure times of finished products impact the measurable schedule performance.

1.2 Purpose of the Study

This dissertation studies an IPODS-FD application with the objective of minimizing holding costs and batch delivery transportation costs in a static and a dynamic environment. The research attempts to provide insights into the application of OR methods for the considered models that are directly implementable and provide a starting point for further model extensions in related or more complex applications. The central research question of this dissertation is as follows:

What is the underlying optimality structure of the considered IPODS-FD application with the objective of minimizing final product holding costs and batch delivery transportation costs and how can this structure be utilized to develop effective optimization algorithms in static and dynamic settings?

To answer this question, this dissertation aims to mathematically formalize the considered IPODS-FD application and develop procedures that use problem-specific properties usable in effective optimization frameworks to generate measurable optimal or near-optimal solutions. This research goal is achieved by meeting the following objectives:

- Formulating a novel scheduling model that captures the problem of minimizing holding and batch transportation costs for an integrated production scheduling and outbound distribution problem with fixed delivery departures.
- Constructing a mixed-integer linear programming problem (MILP) that is solvable with available optimizer suites.
- Developing an exact branch-and-bound (B&B) procedure that benefits from the results of the problem structure analysis and the definition of various dominance properties and lower bounds as a superior alternative to the MILP approach.
- Developing a heuristic greedy randomized adaptive search procedure (GRASP) to construct high-quality solutions in a short time-frame and to provide initial bounds for the B&B algorithm.
- Providing an instance generator to construct a variety of problem instances.
- Implementing the solution approaches and the generator for a static problem environment
- Testing the developed model and procedures through computational experiments on the specifically generated instances.
- Extending the static problem definition to the dynamic case in which some orders arrive dynamically during the planning and execution process.
- Developing a RTC approach that provides schedules in a dynamic environment, where customer orders arrive at various points in time.

- Tailoring the heuristic GRASP to solve static problem instances generated as snapshots of the real-time situation during the application of the RTC approach to provide high-quality dynamic schedules.
- Developing workload balancing methods that are essential to control the workload of the production machine in a dynamic environment to reduce costs and improve order acceptance rates of dynamic schedules in the considered uncertain environment.
- Providing a modification procedure to extend the instances for the static problem to include order arrival times.
- Implementing the RTC approach, modifying the GRASP implementation to work in the dynamic setting, and implementing workload balancing methods.
- Testing the RTC approach on the generated dynamic instance sets through computational experiments.

1.3 Outline

The remainder of this dissertation is structured as follows. In Chapter 2, the basic terminology and notation of general OR and scheduling research are introduced that are relevant for the contents of the subsequent chapters of this dissertation.

Chapter 3 presents the topic of IPODS-FD by reviewing the literature concerning its applications. The chapter provides an overview of the state-of-the art models and applied solution methods.

The proposed IPODS-FD model is presented in Chapter 4. This chapter includes a MILP formulation that is implementable with various optimization suites. Thereafter, Chapter 5 describes the developed B&B algorithm, including various problem properties and dominance criteria. Furthermore, this chapter presents preprocessing techniques and lower bounds for the problem.

Chapter 6 describes the developed GRASP algorithm, which is a competitive heuristic for time-critical applications and, on top of that, provides initial upper bounds for the B&B approach. The approach is specifically designed to construct feasible solutions by applying a repair procedure that iteratively eliminates infeasibility in initially constructed solutions. Moreover, the approach applies a path-relinking procedure to further improve generated solutions.

Chapter 7 discusses the introduced model's extension towards a dynamic environment with dynamic job arrivals and the specifically tailored RTC approach for dynamic schedule generation. The designed approach utilizes a GRASP that solves snapshot instances at periodic intervals of time. Moreover, this chapter discusses workload balancing methods that pro-actively schedule jobs to improve the chances of integrating future order requests.

Chapter 8 presents a comprehensive computational study to evaluate all developed and implemented approaches. The chapter describes static and dynamic instance generators, as well as procedures to generate prediction values for the pro-active workload balancing methods. The optimization methods are tested on instances with different time window settings and cost structures. To evaluate the RTC approach, the experiments vary the number of dynamic order arrivals and the available time to optimize.

Finally, Chapter 9 summarizes the findings and their implications, and discusses future research opportunities.

Chapter 2

Selected Topics of Operations Research and Scheduling

The purpose of this chapter is to briefly introduce the basic concepts and terminology of the general research field of OR and the specific field of *scheduling*. Section 2.1 introduces the selected topics of OR that are relevant for the discussion regarding IPODS-FD models. Then, Section 2.2 presents the deterministic scheduling models described by Pinedo (2016) and Blazewicz et al. (2019). Finally, since a dynamic scheduling problem is considered in Chapter 7, a brief introduction to this problem context is provided in Section 2.3.

2.1 Operations Research

This section is structured in three parts. Section 2.1.1 defines the notion of a general optimization problem, since scheduling problems are optimization problems in the area of scheduling. Section 2.1.2 follows with basic definitions regarding the analysis of computational complexity. Thereafter, Section 2.1.3 offers a brief overview of optimization methods.

2.1.1 Optimization Problems

As described in Chapter 1, OR is concerned with building abstract mathematical models that capture the essence of real-life problems in managing operations—the literature refers to the developed abstract mathematical models simply as *problems*. A problem is a general question to be answered, including parameters or free variables left unspecified. Moreover, a problem comprises a definition of all relevant parameters and a statement of the requirements that an answer or *solution* to the problem must satisfy. An *instance* of a problem specifies all values for the problem parameters (Garey and Johnson, 1979). Many practical and theoretical problems seek a "best" configuration of the parameters to reach a particular goal. Optimization problems, specifically, can be divided into *continuous optimization problems* and *discrete optimization problems*. The former type seeks a set of real numbers or a function, and the latter type seeks discrete numbers as an answer to a problem (Papadimitriou and Steiglitz, 1982).

According to Blum and Roli (2003), an optimization problem $\mathrm{P} = (\mathcal{S}, \mathrm{f})$ is characterized by

- a set of variables $X = \{x_1, ..., x_n\}$,
- variable domains $D_1, ..., D_n$,
- constraints regarding the variables, and
- an objective function f to be minimized, with $f \colon D_1 \times ... \times D_n \to \mathbb{R}^+$.

All feasible solutions to a problem are denoted as

$$\mathcal{S} = \{s = \{(x_1, v_1), ..., (x_n, v_n)\} \mid v_i \in D_i \text{ and } s \textit{ satisfies all constraints}\},$$

where $v_1, ..., v_n$ are the values assigned to the decision variables $x_1, ..., x_n$. This set is also called the search or solution space to a problem. Finding a solution $s^* \in S$ with a minimum objective value – that is,

$$f(s^*) \leqslant f(s) \; \forall s \in \mathcal{S}$$

– is said to *solve* an optimization problem. The solution s^* is also called a global optimum of $(\mathcal{S}, f)$, while $S^* \subseteq S$ is the set of globally optimal solutions. Note that in practice, not all objective functions are of the minimization type (e.g., maximization of profit), but each maximization function can be transformed into an equivalent minimization function. The above definition is consequently sufficient. Generally, (Optimization) problems are generally defined as "templates" for applicable *problem instances*. That is, the parameters of an optimization problem can be changed to express different scenarios of the same problem. In practical applications, it is not only of interest that optimal solutions to a problem exists, but also that there is a procedure that finds such an optimal solution. An *algorithm* is a procedure that solves *instances* of a problem; that is, it produces a solution that satisfies the predefined requirements (Garey and Johnson, 1979).

2.1.2 Complexity Theory

For large solution spaces that cannot be reduced considerably by an optimization algorithm, solving an optimization problem optimally is not always possible in a reasonable time frame. This section provides a brief introduction to the analysis of computational complexity that addresses the issue of tractable and intractable problems. *Complexity theory*, as a field in theoretical computer science and mathematics, studies the difficulty of algorithmic problems. The *theory of NP-completeness* (Garey and Johnson, 1979) provides a framework to categorize problems based on their computational complexity. This section presents a brief overview of the important terminology that is used throughout this dissertation.

The computational complexity of an algorithm can be measured by its *time complexity function* $T(n)$. This function measures the running time of an algorithm, as the number of computational steps required to perform an algorithm given a certain input of size n. Most commonly, one is interested in the asymptotic worst-case time-complexity of an algorithm, which can be expressed using the big $\mathcal{O}$ notation. This notation characterizes functions according to their growth rate; that is, it classifies algorithms according to how their running time (or space requirements) grows as the size of the input provided to an algorithm grows. The asymptotic notation is defined as follows:

Definition 2.1.2.1. A function $T(n)$ is $\mathcal{O}(g(n))$ whenever a positive constant $c > 0$ and non-negative integer n_0 exist such that $T(n) \leqslant c \cdot g(n)$ for all integers $n \geqslant n_0$ (Brucker, 2007).

In the context of OR, one is usually interested in finding *polynomial-time algorithms* to solve optimization problems. A polynomial-time algorithm is an algorithm with a time complexity function that is $\mathcal{O}(p(n))$, for some polynomial-time function p of the form n^k, with a constant k and n as the input length. One consequently calls an algorithmic problem *polynomially solvable* if there exists a polynomial-time algorithm for this problem.

The characteristic of polynomial solvability depends on the encoding of the input. The regular assumption is that numerical data describing a problem is *binary encoded*. For example, numbers $(1, 2, 3, \dots)$ are represented as bits $(1, 10, 11, \dots)$ in a binary encoding. For some problems, changing the encoding from binary to unary enables polynomial solvability. A *unary encoding* for numbers $(1, 2, 3, \dots)$ is a sequence of *ones* in a unary encoding $(1, 11, 111, \dots)$. Such an algorithm is referred to as *pseudopolynomial*, a problem solved by such an algorithm is hence called *pseudopolynomially* solvable (Brucker, 2007).

The complexity theory classifies *decision problems* based on their solvability. A decision problem has only two answers; *yes* or *no*. Fortunately, each optimization problem can be translated into a decision problem by defining a threshold for the objective function value that is to be minimized or maximized. For example, an optimization problem P with cost coefficients c_i for $i = 1, \dots, n$ that minimizes a total cost value $\min z = \sum_{i=1}^{n} c_i x_i$ and comprises a set of constraints, is translated into a decision problem as follows: The decision problem comprises all constraints and replaces the objective by the following question, with θ being a threshold for objective function value z: *Is there a solution to* P *for which* $z \leqslant \theta$ *holds?* If the answer to this question is *yes*, then the given instance is a *yes*-instance, otherwise it is a *no*-instance.

A formal definition of the following complexity classes can be found in Garey and Johnson's (1979) work. For reasons of brevity, we provide the rather informal Definitions 2.1.2.2 and 2.1.2.3:

Definition 2.1.2.2. A decision problem D is in complexity class $\mathcal{P}$ if it is polynomially solvable.

Definition 2.1.2.3. A decision problem D is in complexity class $\mathcal{NP}$ if a *yes* instance can be verified by a polynomial-time algorithm.

In other words, problems in class $\mathcal{P}$ are efficiently solvable. Moreover, for problems in class $\mathcal{NP}$, at least a suggested solution to a problem instance can be efficiently verified to be a valid solution. It holds that $\mathcal{P} \subseteq \mathcal{NP}$, and there is strong evidence that $\mathcal{P} \neq \mathcal{NP}$, although the latter is still an open problem. Garey and Johnson (1979) provide a discussion about the so-called *P versus NP problem*. More recently, an overview of the status of this open problem and further references have been provided (Fortnow, 2009).

Another important complexity class is the class of $\mathcal{NP}$-**complete** problems. Following the assumption that $\mathcal{P} \neq \mathcal{NP}$, a problem that belongs to the $\mathcal{NP}$ class, but not to $\mathcal{P}$, is called $\mathcal{NP}$-**complete**. A convenient method of proving that a problem is $\mathcal{NP}$-**complete** is to conduct a polynomial reduction as described by Definition 2.1.2.4 from a known $\mathcal{NP}$-**complete** problem to the unclassified problem.

Definition 2.1.2.4. A decision problem P polynomially reduces to a decision problem Q (also written as $P \leqslant_p Q$) if a polynomial-time function g exists that transforms inputs for P into inputs for Q such that x is a yes-input for P if and only if $g(x)$ is a yes-input for Q.

This approach is summarized by the following lemma.

Lemma 2.1.2.1. *If the decision problems P and Q are in the class $\mathcal{NP}$, the problem P is $\mathcal{NP}$-**complete**, and $P \leqslant_p Q$, then the problem Q is $\mathcal{NP}$-**complete** as well (Brucker, 2007).*

Not all *hard* problems belong to the class $\mathcal{NP}$. Problems that are not easier than the hardest problem in $\mathcal{NP}$ are called $\mathcal{NP}$-**hard**. More precisely, a problem P is $\mathcal{NP}$-**hard** when every problem of $\mathcal{NP}$ is polynomially reducing to P. Furthermore, Brucker (2007) note that an optimization problem is $\mathcal{NP}$-**hard** if the corresponding decision problem is $\mathcal{NP}$-**complete**. $\mathcal{NP}$-**hard** problems can be further distinguished into two categories: First, an optimization problem is called **strongly** $\mathcal{NP}$-**hard** if it can not be solved by a pseudopolynomial algorithm unless $\mathcal{P} = \mathcal{NP}$; second, an optimization problem is called **weakly** $\mathcal{NP}$-**hard** if it can be solved by a pseudopolynomial algorithm but not by a polynomial algorithm.

Determining the complexity status of an optimization problem is the sensible choice before designing a fitting optimization algorithm. By resolving the complexity status, the choice of useful methods to apply is limited. If one proves a problem to be $\mathcal{NP}$-**hard**, then there is no reason to find a polynomial algorithm. One must hence apply a method that either solves $\mathcal{NP}$-**hard** problems optimally (usually for limited input sizes) or uses heuristics to find at least useful solutions. In contrast, finding a polynomial time algorithm for a problem often makes designing more complex optimization algorithms unnecessary.

2.1.3 Optimization Methods

OR is a vast scientific field with numerous applications. Although the problems that arise in different fields are quite unique, the applied methods are similar. This section provides a brief introduction to commonly used OR methods.

2.1.3.1 Linear and Integer Programming

According to Padberg (2013, p. 25), a general linear programming problem (LP) has the following form:

$$\begin{array}{lll} \text{minimize} & \sum_{j=1}^{n} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{n} a_{i,j} x_j \leqslant b_i & \text{for } i = 1, \dots, p \\ & \sum_{j=1}^{n} a_{i,j} x_j \leqslant b_i & \text{for } i = p+1, \dots, m \\ & x_j \geqslant 0 & \text{for } j = 1, \dots, q \end{array}$$

where $q \leqslant n$. In a case where $q < n$ holds, some of the decision variables x_j for $j = q + 1, \ldots, n$ are unrestricted in sign. Such variables are called *free variables*. The parameters $a_{i,j}$, c_j, and b_i are also referred to as the *data* of the problem. The quantities are defined as real, rational, or integer numbers. Linear programs as defined above are solved by the so called *simplex method*, originally developed by George Dantzig (see Dantzig, 1948 and Dantzig et al., 1951) or by interior point methods (see, for example, Karmarkar, 1984). For many optimization problems, there is the additional requirement that all variables x_j must be integers. An LP with this requirement is called an integer linear programming problem (ILP). If both types of variables are present in a formulation, then the problem is called a MILP. While LPs can be efficiently solved by interior point methods and, in practice, also by the simplex method, no polynomial approach exists for ILPs or MILPs. Several methods exist that solve integer programs: branch-and-cut (B&C) techniques and B&B techniques. The first technique relaxes a given problem to an LP (i.e., the integer restriction is relaxed), and either the simplex method or an interior point method then solves the resulting LP. The approach iteratively tries to force variables to become integers by cleverly adding new constraints to the LP. If the approach generates a solution with only integer values for the variables x_j, then the solution is an optimal solution for the original ILP. Otherwise, new constraints are added.

The second technique is a sophisticated way in which to enumerate the solution space. The B&B procedure generates different branches (subsets) of the solution space and computes bounding values for the considered subsets. If the bound indicates that a subset comprises no better solution than an already identified solution, then the technique excludes the whole subset from further branching steps. B&B algorithms in the context of LPs also use the relaxation technique. An optimal objective function value for the LP provides a bound for the best objective function value attainable by a corresponding ILP.

Combinations of the two approaches exist in the form of B&C and branch-and-price (B&P) algorithms. The former combines B&B with the cutting plane (CP) technique. A CP, or column generation (CG), algorithm is applied to generate the bounding value. These types of algorithms work with a restricted set of variables of the original problem.

It is worth mentioning that commercial and noncommercial solvers are readily available that apply these techniques to solve LPs. An optimization problem formulated as an ILP consequently needs only to be implemented with the appropriate syntax in one of the available modeling languages and solved by one of the available solvers. Two frequently used commercial solvers (which are, in many cases, free for students and researchers to utilize) are the *IBM ILOG CPLEX Optimization Studio*[1] and the *Gurobi Optimizer* [2]. Note that the conventional approach of "simply" constructing an ILP and solving instances with a solver package is often found to be inferior to more elaborate techniques that include problem-specific knowledge in the solution approaches as proven by the vast number of published research papers.

[1] https://www.ibm.com/analytics/cplex-optimizer

[2] https://www.gurobi.com/

2.1.3.2 Dynamic Programming

Dynamic programming (DP) is an optimization method developed by Richard Bellman that solves problems by combining solutions to subproblems (Bellman, 1952, 1953). According to (Cormen et al., 2009, p. 379), the DP technique is based on two principles: an *optimal substructure* and *overlapping subproblems*.

An optimization problem has an optimal substructure if an optimal solution to a problem comprises optimal solutions to subproblems. By combining the "right" solutions to subproblems, one consequently yields the optimal solution to the overall problem (Cormen et al., 2009, p. 380). For example, if the construction of a solution to an optimization problem requires n decisions $D_1, D_2, \ldots, D_n$, then if the sequence with n decisions is optimal, the previous k decisions $D_n, \ldots, D_{n-k+2}, D_{n-k+1}$ must be optimal as well (Papadimitriou and Steiglitz, 1982, p. 448).

The optimal substructure differs between optimization problems in the following two ways: the number of subproblems an optimal solution requires and the number of choices that exist to determine what subproblems to use in an optimal solution. The product of subproblems and choices considered for each subproblem consequently determines the run-time of a DP algorithm. A recursive algorithm commonly solves a dynamic program by computing a recurrence relation that starts at the last stage (i.e., decision D_n) and then recursively solves the previous stages.

This concept leads to the second principle, called overlapping subproblems, requiring an optimization problem to have a relatively small subproblem space. That is, the number of subproblems should be polynomial in the input size. Therefore, a recursive algorithm that solves a dynamic program solves identical subproblems over and over instead of generating new subproblems to solve the overarching problem (Cormen et al., 2009, pp. 380, 384).

2.1.3.3 Branch and Bound

In general, for a combinatorial optimization problem, B&B algorithms search the entire solution space for either a single solution or all optimal solutions. Similar to the DP approach, this entails subdividing the overarching problem into smaller subproblems. B&B algorithms avoid searching the whole solution space by incorporating techniques to exclude significant parts of the solution space from the search (Papadimitriou and Steiglitz, 1982, p. 433). In a general sense, the technique, applicable in the context of ILP as mentioned above, is a framework to search the solution space of combinatorial optimization problems. Researchers may consequently define problem-specific procedures to evaluate subproblems and provide lower and upper bounds without solving LPs. Such an algorithm is proposed in Chapter 5, with Section 5.1 covering the B&B procedure in greater detail.

2.1.3.4 Heuristics and Metaheuristics

The previous *exact* solution methods commonly aim to solve a problem by proving optimality for the found solution. Given sufficient computing time, the entire solution space's consideration by the discussed methods guarantees the optimality property. Due to the computational complexity of difficult optimization problems, these methods often provide optimal solutions to relatively small problem instances in reasonable computational time. Heuristic algorithms undergo a comprehensive but not exhaustive search of an optimization problem's solution space to tackle larger problem instances. Therefore, these methods do not guarantee the optimality of the found solutions. Apart from their application as standalone solution methods, heuristics often supplement exact methods by, for example, providing bounding values for the objective function that exclude regions of the solution space (Gendreau et al., 2010, pp. ix, x).

Typically, one categorizes heuristic algorithms into two types: constructive algorithms and local search (LS) methods. On the one hand, constructive algorithms start with an empty solution and iteratively generate a complete solution to an optimization problem. On the other hand, LS methods start from an initial complete solution and aim to improve the solution by iterative modification. Establishing a so-called *neighborhood* of a solution drives the transition from one solution to another. Definition 2.1.3.1 by Blum and Roli (2003) defines this neighborhood as follows:

Definition 2.1.3.1. A neighborhood structure is a function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ that assigns a set of neighbors $N(s) \subseteq S$ to every solution $s \in S$. $\mathcal{N}(s)$ is called the neighborhood of s.

The best solution of a neighborhood is called a *local minimal solution*, as stated in Definition 2.1.3.2 by Blum and Roli (2003):

Definition 2.1.3.2. A local minimal solution (or local minimum) with respect to a neighborhood structure $\mathcal{N}$ is a solution $\hat{s}$ such that $\forall s \in \mathcal{N}(\hat{s})\colon f(\hat{s}) \leqslant f(s)$. One calls $\hat{s}$ a strict locally minimal solution if $f(\hat{s}) < f(s) \quad \forall s \in \mathcal{N}(\bar{s})$.

Glover (1986) introduced the term metaheuristic (MH) for his tabu search (TS) procedure, which utilizes an LS procedure. Blum and Roli (2003) remarked that there is no commonly accepted definition of MH, although many researchers offer their own definitions. MHs can be described as strategies that guide the search process intending to explore the solution space to find optimal or near-optimal solutions. The procedures used by MHs range from simple search procedures to complex learning processes. The search process is usually non-deterministic, since it incorporates randomized mechanisms to avoid getting trapped in local optima. MH-frameworks are not problem-specific but allow for the integration of procedures that exploit problem-specific knowledge, such as specifically designed heuristics. MHs provide concepts for the two principles of diversification and intensification. Diversification refers to a robust exploration of the search space, which is not confined to small regions. The term intensification refers to the exploration of concentrated regions to find local optima. The success of the solution space exploration of a MH implementation relies on the problem-specific tailoring of these two principles

(Blum and Roli, 2003). Some prominent examples are TS algorithms, genetic algorithms (GA), simulated annealing (SA) algorithms, and GRASP algorithms.

A **TS** is a heuristic method originally proposed by Glover (1986). This procedure's basic idea is to allow LS procedures to overcome local optima. This property is achieved by storing applied neighborhood moves or visited solutions of previous iterations to avoid cycling. In its basic version, a so-called *tabu list* records the history of the search. Elements of the tabu list are *tabu* in subsequent iterations; therefore, the application of certain neighborhood moves are marked as forbidden and will be disregarded in the choice of the next neighborhood move to apply to the current solution. Since a TS is an improvement heuristic, the initial solution must be generated by another procedure (Gendreau and Potvin, 2010). Since a TS extends LS, the core of a basic TS algorithm is essentially an LS procedure. The key difference in a pure LS procedure is that an LS terminates if it finds no improving neighbor in an iteration; in contrast, a TS also accepts non-improving moves. Once a TS carries out a non-improving move from solution s to the neighboring solution $s' \in \mathcal{N}(s)$, the former solution s is now a neighbor of s' and potentially the target of a neighborhood move. This scenario leads to cyclic computations that switch between both solutions indefinitely. Tabus are used to avoid this form of cycling by disallowing moves that reverse a previous move's effect. The so-called *tabu list* stores these tabus. Furthermore, a circular list can implement a tabu list with a fixed capacity. Once the tabu list reaches its capacity limit, it removes the oldest tabu. Hence, the tabu list is known as the short-term memory of the search (Gendreau and Potvin, 2010). Note that there are numerous extensions to this basic form of TS not covered by this dissertation. A good starting point for further information is (Gendreau and Potvin, 2010) that gives an introduction to TS.

SA, similar to TS, primarily uses a LS heuristic to improve a solution iteratively. The key element of SA is a mechanism that allows escaping local optima by hill-climbing moves, which are a sequence of neighborhood moves that are allowed to worsen the solution's objective function value. Annealing refers to the process of physically annealing solids by initially heating and slowly cooling materials. Similarly, SA starts with an initial *temperature* set to a sufficiently high-temperature value. The algorithm iteratively generates and evaluates neighborhood moves one-by-one based on a *generation probability function* that prioritizes the early testing of promising neighborhood moves. A neighborhood move is accepted based on an *acceptance probability function* that depends on the current temperature of the system and the objective value of the neighboring solution compared to the current solution. Accepting worse solutions at high-temperature values is more likely than for lower temperature values. SA lowers the temperature in steps based on an efficient *cooling schedule*. The procedure terminates upon reaching a defined stopping criterion, for example, the number of total iterations (Nikolaev and Jacobson, 2010).

GRASP is a so-called multi-start MH. The iterative process applies two phases in each iteration: solution construction and LS. The first phase constructs a feasible solution from scratch, and the second phase improves the constructed solution by applying an LS heuristic. Upon executing a targeted number of iterations or after a time-limit, the procedure terminates and returns the best-found solution. The construction phase typically constructs solutions step by step, where at each step, the procedure decides which viable construction move to implement. To diversify the constructed solutions, the procedure initially constructs a *candidate list* of construction moves, selects a subset of promising moves as the *restricted candidate list*, and lastly chooses a random candidate from this restricted list (Resende and Ribeiro, 2010). Chapter 6 describes a developed GRASP heuristic and provides more information on the general framework.

A **GA** is a MH framework first introduced by Holland (1992). The idea of GAs heavily draws on natural occurrences. The original motivation stems from the selective breeding of plants or animals with the aim of creating offspring with desirable properties as a combination of the parents' properties. The concept of a GA approach allows for the separation of the solution representation of an optimization problem and its decision variables. The solution representation is also called the *genotype*, while the set of decision variables is known as the *phenotype*. Given a discrete search space X of an optimization problem, a solution in a GA is represented by a string s of length l, with symbols drawn from an alphabet $\mathcal{A}$ using a mapping $c : \mathcal{A}^l \to \mathcal{X}$. Therefore, the heuristic works on a subset of the search space $\mathfrak{S} \subseteq \mathcal{A}^l$ with the objective of finding

$$\arg\min_{s \in \mathfrak{S}} g,$$

where $g(s) = f(c(s))$, with f being the objective function of the optimization problem. The defining characteristic of GAs is that the algorithm works on a set (*population*) of strings, which are often referred to as *chromosomes* or *individuals*. In contrast to TSs, SA, and GRASP, GAs rely on combining solution representations by reproduction rather than an LS on individual solutions. The two concepts to carry out the reproduction process are crossover and mutation. The crossover combines the genes of two (or more) parent individuals to construct one or multiple offspring. For example, for two individuals with bit-strings $(1, 0, 0, 1)$ and $(1, 1, 0, 0)$ a possible offspring representation might be $(1, 0, 0, 0)$, which is constructible by taking the first two bits from the first parent and the last two bits from the second parent. Mutation modifies the strings of the offspring randomly such that the gene pool, and by extension the searched solution space, is diversified. For example, the bit-string $(1, 1, 0, 0)$ might be randomly modified to $(1, 0, 0, 0)$ by flipping the second bit. Furthermore, GAs implement mechanisms to select and remove individuals from a population, usually by replacing worse individuals with better ones. The quality of individuals in a population consequently improves over several iterations such that the best individuals, after multiple rounds (*generations*) of breeding and selecting, represent high-quality or even optimal solutions to the optimization problem (Reeves, 2010).

2.1.3.5 Approximation Algorithms

In addition to heuristic or meta-heuristic algorithms, a frequently studied type of non-exact algorithm is an approximation algorithm. The main characteristic of an approximation algorithm is its performance guarantee. An approximation algorithm for a combinatorial optimization problem constructs a near-optimal solution whose cost is at most a factor ρ away from the optimal cost, where $\rho > 1$ is some real number. Such an algorithm is hence also called a ρ-approximation algorithm. A polynomial-time approximation scheme (PTAS) is a $(1 + \epsilon)$-approximation algorithm with $\epsilon > 0$ that runs in polynomial time. Moreover, if the time complexity of a PTAS is polynomially bounded in $1/\epsilon$, it is called a fully polynomial-time approximation scheme (FPTAS) (Woeginger, 2000).

The first FPTASs were developed in the mid-70s (Ibarra and Kim, 1975; Horowitz and Sahni, 1976; Sahni, 1976). These FPTASs and, according to Woeginger (2000), all FPTASs developed since then are based on dynamic problem algorithms that solve problems optimally, but not in polynomial time. Two established types of techniques use a dynamic problem algorithm as an FPTAS. The first technique is called *rounding the input data* and it transforms the input data to rounded values, such that the transformed instance can be solved in polynomial time. The second technique, called *trimming the state space* which iteratively thins out the search space during algorithm execution, for instance by collapsing states close to each other. Pinedo (2016, p. 599) mentions two additional concepts that transform the input data of scheduling problems: *merging* and *aligning*. *Merging* joins instance data together; for example, jobs that require short processing times on a machine are merged into a single job. *Aligning*, aligns the processing times of similar jobs: instead of having several jobs with almost identical processing times, an average processing time value replaces the processing times for these jobs.

2.2 Deterministic Scheduling

This dissertation is primarily concerned with scheduling problems that are a specific type of optimization problem. This section introduces the basic notation and concepts for deterministic scheduling problems, as introduced by Pinedo (2016) and Blazewicz et al. (2019). Since scheduling is a popular research field due to its practical relevance, a common notation has evolved that applies to various scheduling problems. This notation is presented in Section 2.2.1. Moreover, as this dissertation considers production and delivery decisions, Section 3.1 extends this notation by additional components that are relevant for IPODS-FD models.

2.2.1 Framework and Notation

Scheduling problems come with a wide variety of different modeling assumptions, and this subsection introduces only a basic list of notation. A scheduling problem is characterized by a set $\mathcal{N} = \{1, 2, \ldots, n^N\}$ of n jobs and a set $\mathcal{M} = \{1, 2, \ldots, m^M\}$ of m machines or processors. Parameters and variables associated with jobs and machines will usually be sub-scripted by i if referring to the i-th machine and by j if referring to the j-th job. In complex machine environments, a job j comprises operations on the different machines. Depending on the assumptions made in a scheduling model, jobs may be associated with the following data:

- *Processing time* ($p_{i,j}$). This is the time needed to process job j on machine i. In the case of a single machine environment, the subscript i can be omitted.
- *Release date/arrival time* (r_j). In dynamic settings, jobs usually arrive at various points in time. The release date r_j marks the earliest time job j can start processing on any machine.
- *Due date* (d_j). The due date d_j of a job marks the targeted completion time for all operations of a job j.
- *Deadline* ($\bar{d}_j$). The deadline d_j of a job j is a targeted completion time that can not be exceeded.
- *Weight/priority* (w_j). In some models, certain jobs need to be prioritized over others. The w_j parameter may represent the cost of keeping job j in the system, for example inventory costs.

A solution to a given scheduling problem instance is an assignment of tasks (jobs) to resources (machines) at specific times. A solution thus defines the *completion time* $C_{i,j}$ of the operation of job j on machine i. The completion time of the last operation of a job (or in single machine models, the completion time of job j) is denoted as C_j. Most commonly, solutions to a given problem instance are represented by either a *sequence* or a *schedule*. A *sequence* refers to a permutation of the n job indexes that describes the processing order on a given machine. A *schedule* is a variable setting for a given problem, for example the explicit setting of all $C_{i,j}$ variables. In many scheduling problems, a sequence is sufficient to unambiguously define a locally optimal schedule. That is, given a sequence, the time-dependent decision variables, for example, optimal completion time values (in the context of the applied objective function), can be computed without any ambiguity.

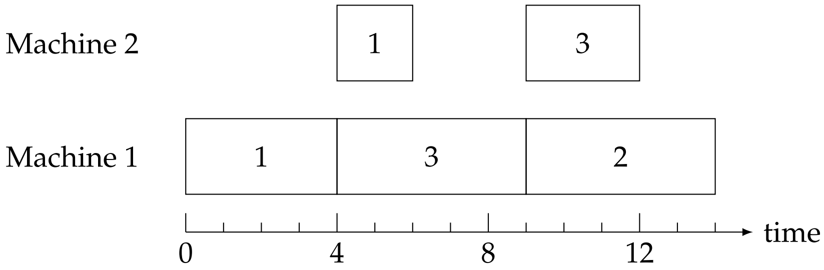
A useful visualization of a schedule is one via a so-called Gantt chart, which is a type of bar chart. Figure 2.1 presents such a chart for a schedule with *two* machines ($m = 2$) and *three* jobs ($n = 3$). In this example, the first job must be processed on both machines, with processing times $p_{1,1} = 4$ and $p_{2,1} = 2$, the second job must be processed only on machine 1, with processing time $p_{1,2} = 5$; and the third job has processing times $p_{1,3} = 5$ and $p_{2,3} = 3$ on the two machines. The processing of each operation, labeled by the corresponding job index, is visualized as a bar that occupies a certain amount of machine time. The specific machine allocation times of the operations can be read directly from the chart.

The objective in machine scheduling literature is usually a function of the completion times of the jobs alone or in conjunction with a specified due date. Some metrics are subsequently defined for scheduling problems with due dates present. The *lateness* of job j is defined as

$$L_j = C_j - d_j,$$

Figure 2.1

Illustration of a Gantt Chart



which takes a positive value if a job is completed *late*, and a negative value if a job is completed *early*. The *tardiness* of a job assigns a zero value to the earliness of a job and is defined by

$$T_j = \max\{0, C_j - d_j\} = \max\{0, L_j\}.$$

Similarly, the *earliness* of a job is defined as

$$E_j = \max\{0, d_j - C_j\} = -\min\{0, L_j\}.$$

Variable U_j indicates whether a job is late or not and is defined as

$$U_j = \begin{cases} 1 \text{ if } C_j > d_j \\ 0 \text{ otherwise} \end{cases} .$$

The objective function in classic scheduling problems is usually composed of one or more of the previously defined metrics. In some cases, these metrics can be weighted, due to a known cost value associated with the earliness or tardiness of jobs or the duration a job is stored in the system. In multi-objective problems, each job usually has a different weight for each objective component. Some commonly found objectives in the literature are the minimization of (weighted) completion times

$$\min \sum_{j=1}^{n} (w_j) C_j,$$

the minimization of the makespan (usually denoted as C_{max}) of the schedule

$$\min \max_{j=1}^{n} \{C_j\},$$

or the minimization of the (weighted) tardiness

$$\min \sum_{j=1}^{n} (w_j) T_j.$$

2.2.2 Complexity Hierarchy

Although many scheduling problems have similar or even identical parameters and constraints, a change in the objective function usually requires a specialized solution method. An easily solvable problem for one objective becomes difficult to solve for a different objective. In other words, the addition, removal, or modification of parameter domains significantly affects a scheduling problem's complexity status. As scheduling problems are a type of optimization problem, the concepts introduced in Section 2.1.2 apply to scheduling problems. The intimate relationships between similar scheduling models also come with a benefit: An algorithm that solves a more complicated scheduling problem solves easier versions of the problem. An example is the one machine scheduling problem with the objective min $\sum_{j=1}^{n} C_j$: Applying the algorithm that solves the more complex version with the weighted objective min $\sum_{j=1}^{n} w_j C_j$ also solves the unweighted version. On the other hand, adding release dates even to the unweighted version yields a **strongly $\mathcal{NP}$-hard** problem (Pinedo, 2016, p. 605). For many deterministic scheduling problems, such relationships are well known and published. For example, Pinedo (2016) and Brucker (2007) present tables and graphs that state the complexity hierarchies for many variants.

2.2.3 Scheduling Algorithms

Many scheduling problems, such as the two problems without release dates discussed in Section 2.2.2, can be solved efficiently. Simple scheduling rules solve some problems; others can be solved by using standard OR techniques by reducing scheduling problems to well-known optimization problems, such as shortest path problems, flow problems, and transportation problems. Some more complicated problems can be solved efficiently by using, for example, DP. Furthermore, the methods described in Section 2.1.3 can be and are applied to solve hard scheduling problems (cf. Chen et al., 1998; Brucker, 2007; Pinedo, 2016). Furthermore, in scheduling literature, the following two types of scheduling rules are widely employed: *dispatching or priority rules* and *composite dispatching rules*.

Dispatching rules can be classified into static and dynamic dispatching rules. A static dispatching rule establishes a job priority based on the given instance parameters only, while a dynamic dispatching rule is time-dependent. That is, dynamic dispatching rules schedule jobs one-by-one where the relative priority order of jobs may change after each step. In multi-machine environments, one can further distinguish rules into local and global rules. Local rules are isolated to information on a single machine. In contrast, global rules account for the overall system state's information.

Composite dispatching rules are composed of ordinary static or dynamic dispatching rules. Such rules are used in constructive algorithms that build schedules iteratively by choosing the highest-ranking job, determined by the composite dispatching rule, to be scheduled next. Each of the dispatching rules is usually scaled by a scaling factor that is either pre-computed from the instance data or updated alongside the scheduling process (Pinedo, 2016, pp.371–374).

2.3 Dynamic Scheduling

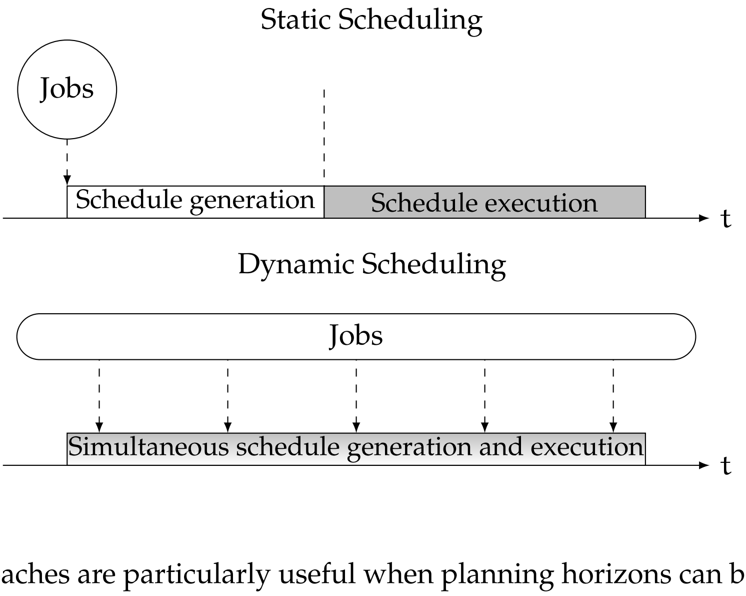
Dynamic scheduling can be understood as the "problem of scheduling in the presence of real-time events" Ouelhadj and Petrovic (2009). In scheduling literature, the terminology *on-line* scheduling, as opposed to *off-line* scheduling, is often used. In off-line scheduling, a schedule is computed entirely before the execution of tasks is realized. Therefore, off-line scheduling considers scheduling with complete (but not necessarily deterministic) information on the scheduled tasks. In contrast, on-line scheduling decisions are made at run-time with incomplete information. In a technical sense, on-line scheduling provides solutions to incomplete problem instances, with information revealed piece by piece (Blazewicz et al., 2019, p. 243).

In dynamic scheduling, real-time events can be classified into two categories: *resource-related* events and *job-related* events. The first category comprises, for example, machine breakdowns or shortages of materials, while the second category comprises new incoming jobs; cancellation of jobs; or changes in job characteristics, such as processing time or due date changes.

Figure 2.2 illustrates the difference between static and dynamic scheduling for a problem with dynamically arriving orders. On the time-axis schedule generation and execution appear sequentially for static scheduling models. All relevant problem information (in the illustration, orders) are registered before the schedule generation commences. Only after the schedule generation ends, the execution starts.

Figure 2.2

Schedule Generation and Execution for Static and Dynamic Scheduling



Static approaches are particularly useful when planning horizons can be planned separately and when it is sufficient to fulfill incoming production requests in the subsequent planning horizon. In contrast, in dynamic scheduling applications, reacting to changes in the problem information (e.g., new jobs) is instrumental to efficiently plan the considered process. Therefore, schedule generation and execution run simultaneously. Such environments are particular challenging to optimize, since future events of the current planning horizon are unknown and difficult or impossible to predict.

According to Ouelhadj and Petrovic (2009), dynamic scheduling is distinguishable by completely reactive scheduling, predictive-reactive scheduling, and robust pro-active scheduling.

- *Completely reactive scheduling*. This approach makes decisions locally in real-time. Priority dispatching rules usually determine the next job to schedule from a list of available jobs at the time. A reactive scheduling system consequently does not attempt to provide a schedule for all remaining jobs and, therefore, does not make decisions based on the overall schedule performance.

- *Predictive-reactive scheduling*, In comparison to completely reactive scheduling, a predictive-reactive scheduling strategy builds and revises schedules based on real-time events. Strategies that perform minimal adjustments to the schedule are often applied. The deviation between two consecutive proposed schedules is usually of no concern, although some others explicitly consider schedule stability and schedule efficiency simultaneously.

- *Robust pro-active scheduling*. Robust pro-active approaches incorporate a predictive element into schedule generation, for example the likelihood of machine breakdowns influencing schedule generation.

Real-time schedulers react to incoming events either by *schedule repair* or by *complete rescheduling*. The first method makes only minor adjustments to the current implemented schedule to incorporate dynamic events. On the other hand, complete rescheduling generates an entirely new schedule with the additionally revealed information present. Schedule repairing is usually computationally less expensive, but complete rescheduling potentially leads to a more efficient schedule. The issue of reschedule timings can be distinguished into three strategies: *periodic*, *event-Driven* and *hybrid*. Periodic policies generate schedules at regular intervals; therefore, a generated schedule is always valid for some time. Event-driven policies trigger responses directly upon new events. Periodic policies lead to more schedule stability and less nervousness, but they react slower to new events. Hybrid approaches reschedule periodically and in an event-driven manner when exceptional events occur. The scheduling techniques applied vary vastly: heuristics, meta-heuristics, knowledge-based systems, fuzzy logic, neural networks, multi-agent systems, and hybrid approaches (Ouelhadj and Petrovic, 2009).

Chapter 3

Integrated Production and Outbound Distribution Scheduling with Fixed Delivery Departure Dates

Chapter 1 motivated the investigation into integrated production and distribution problems on a detailed scheduling level. This chapter presents the current state of research of IPODS-FD models and solution approaches; it aims to provide an overview of IPODS-FDs and discuss gaps in the published studies that motivate the research in this dissertation. Section 3.1 extends the previously introduced notation in Section 2.2.1 by introducing fixed delivery departure times. Thereafter, Section 3.2 presents a classification scheme for IPODS-FD problems that extends the general classification scheme for IPODS problems by Chen (2010). This classification scheme displays the different modeling assumptions of the present IPODS-FD literature without requiring a detailed comparison of the proposed models. The literature review of IPODS-FD models in Section 3.3 then classifies the current literature with the help of the introduced classification scheme and discusses each study. The chapter concludes with Section 3.4, which discusses the state of research and aspects that deserve increased attention.

3.1 Considering Fixed Delivery Departure Dates

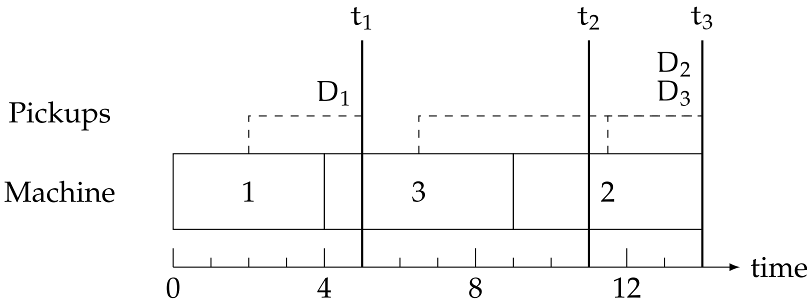
The production scheduling problems described so far do not cover any explicit transportation decisions except for due dates that may indicate the planned product's delivery. When due dates are present in a scheduling model, completing a job earlier or later than its due date usually leads to earliness or tardiness penalties. The question in these models remains whether a late completion time is simultaneously the new delivery departure time, or, if this is not the case, then when and at what cost the unplanned delivery takes place. Scheduling models that integrate transportation decisions answer these questions by unambiguously defining the delivery departure time of a job by either explicitly setting delivery variables or introducing a dependency relation between completing and delivering a job. This section introduces the modeling assumptions of these types of models in a general way. The notation presented is similar to the one used by Chen (2010).

In IPODS models, each job j has a completion time C_j and a *delivery departure time* (or *pickup time*) D_j that represents the point in time where the transportation of the produced products of job j begins. It immediately follows that $D_j \geqslant C_j$ must hold in a feasible solution for any scheduling problem that considers delivery decisions. Several literature models assume that the delivery departure time D_j coincides with the completion time C_j of a job. Other models impose certain restrictions on the choice of D_j. For example, vehicles' availability or earliest and latest departure times restrict the choice of D_j. As this dissertation is concerned with schedule problems that include fixed delivery departure dates, the following description focuses on these types of problems.

In IPODS-FD models, a given number of transports $I^T = \{1, ..., n^T\}$ exist that carry produced products. A transport $i \in I^T$ has fixed delivery departure time $t_i \in T$, which marks the time-point for this transport at which a produced product that uses the i-th transport must be ready for handling. IPODS-FD models do not explicitly model the handling activities, as they are assumed to require a constant time. The delivery departure time variable D_j for a job $j \in \mathcal{N}$ must equal one of the given fixed departure times $t \in T$ in a feasible schedule. Figure 3.1 illustrates a schedule for an IPODS-FD.

Figure 3.1

Illustration of an IPODS-FD schedule*



* integrated production and outbound distribution scheduling with fixed delivery departure dates

Note. The fixed delivery departure times t_1,t_2, and t_3 are input parameters and form the domain for the assignable delivery departure times of the three jobs. The dashed lines and the placement of the D_j variables indicate the allocated times of the schedule. In this example, the first job departs at time $t_1 = 5$, and the second and third jobs depart at time $t_3 = 14$.

Basic models of this type assume that each job j is generally schedulable to depart at any time $t \in T$. More complex models consider multiple customers' orders where dedicated vehicles transport finished products to individual customer locations. In these multi-customer models, a set of customers $G = \{1, ..., n^G\}$ exists, with each job $j \in N$ belonging to one of the $\mathcal{N}^G$ customers. To distinguish jobs with different customers, let $\mathcal{N}_g \subseteq \mathcal{N}$ define the subset of jobs that belongs to customer g. Furthermore, the transports I^T and the fixed delivery departure times T are distinguished by subsets $I_1^T, I_2^T, \dots I_{n^G}^T$ and $T_1, T_2, \dots T_{n^G}$ such that the delivery departure time D_j for job $j \in \mathcal{N}_g$ is restricted to transports I_g^T that depart at times T_g.

Jobs may also be associated with additional data concerning delivery decisions. To differentiate between parameters and variables concerning the completion time and the departure time of a job, a hat above a symbol (e.g., $\widehat{x}$) appears whenever a distinction is required. Otherwise, the use of the additional symbol is omitted. IPODS-FD models comprise some of the following parameters:

- *Departure due date* ($\widehat{d}_j$). The departure due date $\widehat{d}_j$ of a job denotes the targeted departure time of a job j. Similarly to classic production scheduling models, a tardiness cost penalizes late deliveries.
- *Departure deadline* ($\widehat{\bar{d}}_j$). The departure deadline marks the latest departure time of a job j. With deadlines present, any feasible schedule must adhere to the deadline restrictions.
- *Departure window* ($[a_j, b_j]$) A departure window is given if the departure time is constrained. Due to this restriction, early delivery before time a_j is not allowed. Instead, produced products are stored until the first allowed delivery time.

Modified scheduling metrics concerning the completion times can be defined equivalently for the departure times:

$$\widehat{L}_j = D_j - \widehat{d}_j$$

is the lateness of a job;

$$\widehat{T}_j = \max\{0, D_j - \widehat{d}_j\} = \max\{0, \widehat{L}_j\}$$

is the tardiness of a job;

$$\widehat{E}_j = \max\{0, \widehat{d}_j - D_j\} = \max\{0, -\widehat{L}_j\}$$

is the earliness of a job; and being late is indicated by

$$\widehat{U}_j = \begin{cases} 1 \text{ if } D_j > \widehat{d}_j \\ 0 \text{ otherwise} \end{cases} .$$

Furthermore, the time between completion and departure may be of interest. The expression

$$H_j = D_j - C_j$$

describes the holding time of a completed job $j \in N$. In some models, optimal D_j values arise from the optimized completion times; in others, allocation of jobs to departure times is part of the optimization problem.

3.2 A Classification Scheme for Integrated Production and Outbound Distribution Scheduling with Fixed Delivery Departure Dates

This section presents a classification scheme for scheduling problems with fixed delivery departure dates, which builds on the classification scheme introduced by Chen (2010). The aim of an optimization problem classification scheme is generally to concisely describe related optimization problems without the need to explicitly state all parameters, variables, objectives, and constraints in a mathematically precise way. A prominent example of such a classification scheme appears in Graham et al.'s (1979) study that classifies machine scheduling problems by a *three-field notation*. Each field describes a set of assumptions about the classified scheduling problem. This notation denotes each problem in the form

$$\alpha \mid \beta \mid \gamma,$$

where the α field describes the machine configuration, the β field lists the restrictions and constraints associated with jobs, and the γ field specifies the objective function.

Similar tuple notations exist for problems for queuing systems (Kendall, 1953), project scheduling (Brucker et al., 1999), cutting and packing (Dyckhoff, 1990), assembly line balancing (Boysen et al., 2007), and crane scheduling with interference (Boysen et al., 2017). Chen (2010) presents a classification scheme dedicated to IPODS problems. This scheme extends the notation of Graham et al. (1979) by two additional fields. The field π describes the characteristics of the delivery process, and the field δ represents the number of customers. The model representation introduced by Chen (2010) tries to incorporate many different model types with various assumptions surrounding the transportation environment. Therefore, the utilized notation is imprecise in some aspects. For example, the objective field distinguishes between total trip-based transportation costs, vehicle-based transportation costs, total production costs, and the indication of a revenue-based performance measure. Since the focus of this thesis is on integrated production and distribution scheduling models with fixed departure dates, the classification scheme introduced next attempts to expand the notation for this model type. The classification scheme describes an optimization problem in five fields

$$\alpha \mid \beta \mid \pi \mid \delta \mid \gamma,$$

where α is the machine configuration, β lists the restrictions and constraints associated with jobs, π describes the characteristics of the delivery process, δ represents the number of customers, and γ specifies the objective function.

The following paragraphs define an inherent order for the attributes of the five fields. This structuring approach adopts Graham et al.'s (1979) classification scheme, which describes ordered attribute lists for machine scheduling problems. Chen et al. (1998) and Boysen et al. (2007, 2017) structure the attributes in a similar way. This section's listed attributes define only a fitting selection to express the literature's current state. Additional attributes can freely extend the classification scheme to represent new models. The symbol $\circ$ marks default values in the classification scheme that do not need to appear in the tuple representation.

Machine Configuration (α)

The machine configuration is specified by two attributes: $\alpha = \alpha_1\alpha_2$[1]. The attribute $\alpha_1 \in \{\circ, P, Q, R, F, J, O\}$ denotes the machine environment. In the case of $\alpha_1 \in \{\circ, P, Q, R\}$,

[1] Note that α is a string with two characters α_1 and α_2 and not a mathematical expression.

a job consists of a single operation that can be processed on any machine or a subset of machines.

$\alpha_1 = \circ$ Single-machine configuration. All jobs are processed on a single machine ($p_{1,j} = p_j$).

$\alpha_1 = P$ Parallel-machine configuration. There are a number of identical machines with $p_{i,j} = p_j$.

$\alpha_1 = Q$ Uniform parallel-machine configuration. The processing speed of each machine i may differ such that $p_{i,j} = q_i p_j$, where q_i is a given machine speed factor.

$\alpha_1 = R$ Unrelated parallel-machine configuration. The processing times on the machines are arbitrary.

In some models, a job consists of more than a single operation. In these cases, symbols indicating a multi-stage model are used. Each job comprises a chain of operations $(o_{1,j}, ..., o_{m,j})$, where each operation $o_{i,j}$ requires $p_{i,j}$ time units of processing time.

$\alpha_1 = F$ Flow shop configuration. Each job must be processed on each machine. All jobs must follow the same path through the system; jobs must be processed in the same machine order. In permutation flow shop configurations, the processing order of jobs is identical on all machines.

$\alpha_1 = J$ Job shop configuration. Each job has an individual machine sequence it must follow. In the base case, each job needs to be processed only once on each machine. In more general cases, a job may visit the same machine multiple times.

$\alpha_1 = O$ Open shop configuration. No restrictions with regards to the routing of jobs through the machine environment exist.

The second attribute $\alpha_2 \in \{\circ, m\}$ describes the number of machines or stages considered.

$\alpha_2 = \circ$ The number of machines is a variable.

$\alpha_2 = m$ There is a fixed number of m machines.

In the case of a single machine problem, $\alpha_1 = \circ$ and $\alpha_2 = 1$; for other problem types, it holds that $a_1 \neq \circ$ and $a_2 \neq \circ$ (see Chen et al., 1998). For example, a parallel machine problem with m machines, where m is specified by the input, is denoted as $\alpha = Pm$.

Processing Characteristics and Constraints (β)

The first attribute indicates whether the processing of jobs can be interrupted once started: $\beta_1 \in \{\circ, pmtn\}$.

$\beta_1 = \circ$ The processing of jobs can not be interrupted.

$\beta_1 = pmtn$ Preemption constraints. Processing of a job can be interrupted and continued at a later time.

The second attribute indicates whether processing consumes limited resources: $\beta_2 \in \{\circ, res\}$.

$\beta_2 = \circ$ No resource constraints.

$\beta_2 = res$ During processing a job uses $r_{h,j}$ resources.

The third attribute indicates precedence constraints: $\beta_3 \in \{\circ, prec, tree\}$.

$\beta_3 = \circ$ No precedence constraints exist.

$\beta_3 = prec$ Precedence constraints. The order of jobs is restricted by precedence constraints of the form: job a must be scheduled before job b.

$\beta_3 = tree$ Precedence constraints. The order of jobs is restricted by precedence constraints that together form a tree.

The fourth attribute indicates the arrival of orders at the system: $\beta_4 \in \{\circ, r_j\}$.

$\beta_4 = \circ$ All orders are available at time 0.

$\beta_4 = r_j$ Release dates. The release date represents the time when a job arrives at the system. This is the earliest time a job j can start its processing.

The fifth attribute indicates whether a common due date exists: $\beta_5 \in \{\circ, d_j, d_j = d\}$.

$\beta_5 = \circ$ No due dates are specified.

$\beta_5 = d_j$ Each job has an individual due date.

$\beta_5 = d_j = d$ All jobs have a common due date.

The sixth attribute indicates whether orders are assigned a deadline: $\beta_6 \in \{\circ, \bar{d}_j\}$.

$\beta_6 = \circ$ No due date is specified for the orders.

$\beta_6 = \bar{d}_j$ Deadline. Represents the committed completion or shipping time of a job. A completion time or departure time greater than $\bar{d}_j$ is forbidden for job j.

$\beta_6 = \bar{d}_j = \bar{d}$ All jobs have a common deadline.

The seventh attribute describes processing characteristics and constraints. In a single machine problem, the machine index i is omitted: $\beta_7 \in \{\circ, p_{i,j} = 1, p_{i,j} = p, p_{i,j} = 0\}$.

$\beta_7 = \circ$ Each job j has processing time $p_{i,j}$ on machine i.

$\beta_7 = p_{i,j} = 1$ All jobs have unit processing times.

$\beta_7 = p_{i,j} = p$ All jobs have equal processing times.

$\beta_7 = p_{i,j} = 0$ Jobs have no processing time.

Delivery Characteristics and Constraints (π)

Various assumptions exist regarding the transportation stage of integrated scheduling models. The first attribute specifies the *number of departure dates* $\pi_1 \in \{1, 2, T\}$. To emphasize the classification as an integrated model, the value of attribute π_1 is always denoted.

$\pi_1 = 1$ Only a single departure date exists. All orders are delivered at this date.

$\pi_1 = 2$ Two departure date exist. All orders are delivered either on the first or the second date. Usually, the objective function penalizes the assignment of jobs to the second date.

$\pi_1 = T$ Multiple departure dates exist and are part of the input. Orders have to be assigned to specific departure dates.

The second attribute specifies the *transportation limit* $\pi_2 \in \{I, C, Q, \circ\}$ for departure at a specific time.

$\pi_2 = I$ Individual shipment. Each delivery consists of only one order.

$\pi_2 = C$ Number restricted shipment. Each delivery can consist of up to c orders where $c < n$.

$\pi_2 = Q$ Capacity restricted shipment. Each delivery can consist of at most Q units, where each job j consists of a different number of units.

$\pi_2 = \circ$ Unrestricted shipment. Each delivery can consist of any number of orders.

The third attribute describes the *delivery mode* $\pi_3 \in \{\circ, \mathtt{routing}\}$. This attribute only applies to multi-customer models.

$\pi_3 = \circ$ Batch delivery by direct shipping. Each shipment transports goods only to a single customer.

$\pi_3 = \mathtt{routing}$ Batch delivery with routing. Each shipment can deliver goods to multiple or all customers.

The fourth attribute indicates *transportation splitting*. In some models, a customer order comprises multiple production jobs, and either the finished products must be transported together or splitting the delivery of orders is allowed: $\pi_4 \in \{\circ, \mathtt{split}\}$.

$\pi_4 = \circ$ Orders must be delivered as a whole.

$\pi_4 = \mathtt{split}$ Splittable delivery. Orders are allowed to be split and can be delivered in multiple batches.

The fifth attribute states whether orders are pre-assigned to departure dates or if the choice is part of the decision problem: $\pi_5 \in \{\circ, \mathtt{pre}\}$.

$\pi_5 = \circ$ Orders must be assigned to departure dates.

$\pi_5 = \mathtt{pre}$ Orders are assigned to departure dates as part of the input.

The sixth attribute describes the *departure date structure*: $\pi_6 \in \{\circ, \mathtt{eqdist}\}$.

$\pi_6 = \circ$ No structure is required for departure dates.

$\pi_6 = \mathtt{eqdist}$ The distance between two consecutive departure dates is a constant. In other words, all consecutive pairs of departure dates are equidistant to one another.

Number of Customers (δ)

Since the number of customers considered in integrated models is an important parameter, it is specified in a separate field.

$\delta = 1$: Single customer. All deliveries are made to a single customer.

$\delta = K$: Multiple customers. Deliveries are made to at least two customers.

$\delta = N$: One time customers. Each job is delivered to a different customer.

Note that considering multiple customers in a model influences other model aspects. For example, transportation costs may depend on the ordering customer. Additionally, each transport may only transport products to a single customer location or a subset of locations.

Objective Functions (γ)

In most production scheduling models, the objective function takes the completion times of jobs into account (Pinedo, 2016). Some objectives may target the production completion time; other models use functions depending on the actual departure time. In line with Hall et al. (2001), variables associated with the delivery time of jobs instead of the completion time are made distinguishable by the symbol ($\widehat{\ }$). Note that the delivery time itself is denoted as D instead of $\widehat{C}$. The objective function γ may comprise a combination of multiple objectives, for example the sum of multiple cost-related measures.

$\gamma = C_{max}/D_{max}$ Makespan. This objective is defined as $\max\{C_1, ..., C_n\}$ which is the completion time of the last job to leave the system. This measure is utilized mostly if no delivery assumptions are made. In integrated models the related objective measures the latest delivery time.

$\gamma = L_{max}/\widehat{L}_{max}$ Maximum Lateness. This objective is defined as $\max\{L_1, ..., L_n\}$ which measures the largest lateness, this can be either production or delivery lateness of all jobs.

$\gamma = \sum(w_j)C_j/D_j$ Total (weighted) completion or departure time. This is the weighted sum of completion or departure times. If the weights represents inventory holding costs, the objective value is equivalent to the total cost value.

$\gamma = \sum(w_j)F_j/\widehat{F}_j$ Total (weighted) flow time. This objective measures the time of a job in the system if jobs have *nonzero* release dates. This time is either defined as $F_j = C_j - r_j$ or as $\widehat{F}_j = D_j - r_j$.

$\gamma = \sum(w_j)T_j/\widehat{T}_j$ Total (weighted) tardiness. This objective generalizes the total (weighted) completion time objective, since a solution to an instance with all due dates set to value 0 has the same objective function value under both objectives.

$\gamma = \sum(w_j)E_j/\widehat{E}_j$ Total (weighted) earliness. This objective measures the distance between the completion time from an agreed-upon due date/departure date.

$\gamma = \sum(w_j)U_j/\widehat{U}_j$ (Weighted) number of tardy jobs. This objectives measures all jobs that are tardy for a given production due date or delivery due date.

$\gamma = \sum(w_j)V_j/\widehat{V}_j$ (Weighted) number of early jobs. These objectives measure all jobs early for a given production due date or delivered earlier than expected.

$\gamma = \sum \widehat{v}_j$ Cumulative payoffs. This measure calculates a payoff related to the delivery departure dates of jobs that is cumulative. The payoff for a job j is calculated as a non-increasing step-wise function of the job's delivery departure time. Early deliveries are consequently beneficial to the objective function value.

Some models consider a finished goods storage penalty that applies between completing a job and its pickup. This objective is captured as follows:

$\gamma = \sum(w_j)H_j$ Total (weighted) holding time or sum of holding costs. This objective is concerned with the dwell time of jobs between the completion time and the actual departure time ($H_j = D_j - C_j$). Note that this objective is identical to the $\sum(w_j)E_j$ objective when the departure time of a job is given by the input. The objective is different when there exists a planned departure time but the model allows departures at other times.

The above objectives rate the time of completion or departure of orders. Most models considered in the literature use at least one of the aforementioned departure-time-dependent objectives. Some models also include additional costs based on the delivery decisions or, in rare cases, production costs. Unlike the above completion- or departure-time-based objectives, no commonly used notation for delivery-related measures exists. To indicate a fixed transportation cost or the minimization of the number of transports, we introduce B_i as a binary variable set to *true* if at least one job uses the $i-$th delivery departure date. Therefore, the following objective is defined:

$\gamma = \sum(w_i)B_i$ Fixed transportation cost, or number of deliveries. In the weighted case, using the transport at time t_i costs w_i monetary units. In the unweighted case, the minimization of the number of deliveries is sought.

As the structure of transportation decisions varies largely between problem formulations, we abstain from capturing other cost-related objectives in detail. Instead, we use the following notation to capture the essence of the different cost assumptions:

$\gamma = TC^N$ Variable job dependent transportation cost. This cost assumption defines a different cost value for each job. The motivation for this cost structure is to capture different job characteristics such as size or weight.

$\gamma = TC^M$ Variable mode dependent transportation cost. This cost assumption defines a different cost value for each delivery date to indicate the use of different transportation modes: For example truck transportation is more costly but faster than transportation by train.

$\gamma = TC^V$ Fixed vehicle transportation cost. At each departure time, there are multiple vehicles ready for transportation where using a vehicle incurs a cost value. This cost measure works in conjunction with limited vehicle capacities.

Additionally, some models consider special measures. These measures appear below:

$\gamma = \sum R_j$ Revenue. In acceptance/rejection models, the delivery of an order j results in revenue R_j.

$\gamma = PC^R$ Regular time production cost. Some models consider a periodic planning horizon in which production cost differs between periods. Each period has a certain capacity (in time units) available to schedule jobs.

$\gamma = PC^O$ Overtime production cost. This measure applies in combination with the PC^R cost measure. The production capacity of periods can be enlarged by, for example scheduling overtime for employees. Overtime production is usually more costly than regular time production.

$\gamma = Cap$ Capacity usage. If transportation is capacitated, this objective maximizes the used or minimizes the unused capacity.

3.3 A Review of Models with Fixed Departure Dates

This section presents an overview of published articles that deal with IPODS-FD problems that appear in the literature. The main modeling assumption that is shared across all of the discussed models is the presence of fixed delivery departure times in the context of outbound distribution. Hence, models that focus on inbound transportation activities are not discussed. The previously introduced classification scheme classifies the studied models. Table 3.1 lists the reviewed publications, the tuple-representation of the studied model(s), and the applied solution methods.

Common model characteristics are investigated in the following literature review. The majority of models are of the following five types:

- *Models with predetermined deliveries* (Section 3.3.1)
- *Models with equidistant deliveries* (Section 3.3.2)
- *Models with arbitrary deliveries* (Section 3.3.3)
- *Models with multiple customers* (Section 3.3.4)
- *Models with batch deliveries* (Section 3.3.5)

Note that some models can be characterized by more than one type.

3.3.1 Models with Predetermined Deliveries

This section presents models with *predetermined deliveries*. In a few models, the delivery departure times D_j are agreed upon in advance and given as problem parameters. The designed optimization methods are consequently not concerned with delivery decisions. Instead, the applied approaches must find a schedule with job completion times close to the agreed departure time. The delivery times in these models are modeled either as *due dates* or as *deadlines*. In the first case, completion times are allowed to exceed the planned delivery time, and a tardiness penalty applies. In the second case, each job must be completed at or before the planned delivery time. In general, many classic scheduling models that do not explicitly deal with transportation decisions also include due dates or deadlines; we exclude these models from the discussion. Instead, this section only considers problem formulations that minimize earliness costs, as this objective is often considered in IPODS-FDs.

An early work that includes delivery decisions with predetermined deliveries is by Chand and Schneeberger (1988); the studied problem is denoted as $1 \mid \bar{d}_j \mid T, pre \mid 1 \mid \sum w_j H_j$ by the five-field notation. The authors deal with a just-in-time (JIT) production environment where the delivery of jobs occurs exclusively at given points in time. The investigated scenario assumes tight production capacities and significantly fewer due dates or transports than jobs. This assumption requires the joint transportation of finished products. Hence, it is frequently impossible to complete each job exactly at its departure date in this setting. The scheduling problem under consideration minimizes the total weighted earliness (WE) subject to no tardy jobs. More precisely, the problem is to find a feasible schedule for n jobs on a single machine, where each job has a processing time p_j and a planned departure date $\bar{d}_j$ (deadline), and is associated with a holding cost value c_j^H. The objective is to minimize the total weighted holding costs

$$\sum_{j=1}^{n} (\bar{d}_j - C_j) \cdot c_j^H.$$

Table 3.1

Articles that Consider IPODS-FD Problems (in Chronological Order)

Publication	**Model**	**Method**
Chand and Schneeberger (1988)	$1 \mid \bar{d}_j \mid T, \mathit{pre} \mid 1 \mid \sum w_j H_j$	DP, H
Matsuo (1988)	$1 \mid \bar{d}_j \mid 1 \mid 1 \mid \sum w_j \widehat{T}_j + PC^R + PC^O$	H
Lee et al. (1991)	$1 \mid d_j = d \mid 1 \mid 1 \mid \sum w_j^1 E_j + w_j^2 T_j + \sum w_j^3 U_j$	DP
Chhajed (1995)	$1 \mid \bar{d}_j \mid 1/2, \mathit{eqdist} \mid 1 \mid \sum H_j + \beta \sum D_j$	PA, H
Lee and Li (1996)	$1 \mid (p_j = p) \mid T, \mathit{eqdist} \mid 1 \mid \sum H_j + \beta \sum D_j$	DP, PA
Yang (2000)	$1 \parallel T \mid 1 \mid \sum (w_j) H_j / \max (w_j) H_j$	PA
Hall et al. (2001)	$\alpha \parallel T \mid 1 \mid \gamma,$	PTAS, DP
	$\alpha \in \{1, Pm, P, F2, F3, J2, J3, O2, O3, O\},$	
	$\gamma \in \{D_{max}, \sum (w_j) D_j, \widehat{L}_{max}, \sum (w_j) \widehat{U}_j, \sum (w_j) \widehat{T}_j\}$	
Wang et al. (2005)	$1 \mid r_j \mid T, C \mid G \mid Cap$	H, PDR
Li et al. (2005, 2006)	$1 \mid r_j \mid T, \mathit{pre} \mid K \mid \sum w_j^1 H_j + w^2 \sum (U_j)^*$	MILP, H
Stecke and Zhao (2007)	$1 \mid \mathit{pmtn}, \bar{d}_j \mid T, (\mathit{split}) \mid 1 \mid \sum f(D_j)\}$	PA, H
Chen (2010)	$1 \parallel T, C \mid 1 \mid \gamma$	DP
	$\gamma \in \{f(D_j), f(D_j) + TC, \sum D_j + TC\}$	
Carrera et al. (2010a,b)	$1 \mid \mathit{res} \mid 1 \mid 1 \mid C_{max}$	PTAS
	$1 \mid d_j = d, \mathit{res} \parallel 2 \mid 1 \mid (1 + \sum w_j \widehat{U}_j) \cdot C_{max}$	
	$1 \mid \mathit{res} \mid T \mid 1 \mid \sum w_j D_j$	
Fan (2010)	$1 \parallel T, (\mathit{eqdist}) \mid 1 \mid D_{\mathrm{max}} + \sum w_j V_j$	PTAS
Fu et al. (2012)	$1 \mid r_j, c_j, \bar{d}_j, \mathit{accept} \mid T, Q \mid 1 \mid \sum R_j$	PTAS
Seddik et al. (2013)	$1 \mid (r_j), (\mathit{pmtn}) \mid 2/T \mid 1 \mid \sum \widehat{\mathcal{V}}_j$	PA,DP
Leung and Chen (2013)	$1 \parallel T, C \mid 1 \mid \gamma$	PA
	$\gamma \in \{\widehat{L}_{max}, \widehat{L}_{max} + \sum B_i, \alpha \widehat{L}_{max} + \beta \sum B_i\}$	
Ma et al. (2013)	$1 \mid r_j \mid T \mid 1 \mid \sum w_j^1 H_j^1 + w_j^2 H_j^2 + \sum w_j^3 \widehat{T}_j + TC^M$	GA
Tyagi et al. (2013)	$1 \parallel T \mid 1 \mid \sum w_j^1 H_j + w^2 \sum B_i$	-
Mensendiek et al. (2015)	$Pm \parallel T \mid 1 \mid \sum w_j \widehat{T}_j$	B&B,GA,TS
Liu and Hsu (2015)	$Jm \mid a_j \mid T, \mathit{eqdist} \mid 1 \mid \sum w_j^1 \widehat{F}_j + \sum w_j^2 H_j$***	PDR
Li et al. (2017)	$1 \mid [\hat{a}_j, \hat{b}_j], \mathit{noidle} \mid T, (\mathit{force\ batch}) \mid K \mid \sum w_j^1 H_j + TC^{N,M}$	CG, TS
Han et al. (2019)	$F(1,1) \parallel T, (Q), \mathit{split} \mid K \mid \gamma$	PA, DP
	$\gamma = \sum w_j^1 H_j + \sum w_j^2 \widehat{U}_j + \sum w_j^3 (1 - \widehat{U}_j) + TC^V$	
Wang et al. (2020a,b)	- **	GA

DP: Dynamic Programming B&B: Branch-and-bound TS: Tabu Search GA: Genetic Algorithm PDR: Priority Dispatching Rule PTAS: Polynomial-time Approximation Scheme PA: Polynomial-time Algorithm CG: Column Generation H: Heuristic MILP: Standard solver approach

* Notation for the scheduling problem
** Multi-stage model for a port-centric supply chain that is not captured by the introduced notation
*** The problem is a dynamic scheduling problem with job arrival times a_j

Due to the deadline assumption, the constraints

$$C_j \leqslant \bar{d}_j \quad \forall j \in \{1, 2, \ldots, n\}$$

must additionally hold. This so called WE problem allows idle times on the machine. A variant of this model is the constrained weighted earliness (CWE) problem that disallows machine idle time by introducing the constraint

$$C_j \leqslant \sum_{i=1}^{n} P_i$$

for each job $j \in \{1, 2, \ldots, n\}$. Chand and Schneeberger argue that the CWE problem is equivalent to the $\mathcal{NP}$-**hard**[2] weighted completion time problem subject to no tardy jobs $1|\bar{d}_j|\sum w_j C_j$ and prove that the objective functions only differ by a constant. The CWE problem is consequently $\mathcal{NP}$-**hard** even for an identical weights setting.

Lee et al. (1991) formulate the second model with predefined departure times denoted as $1 \mid d_j = d \mid 1 \mid 1 \mid \sum w_j^1 E_j + w_j^2 T_j + \sum w_j^3 U_j$. Only a single departure date (common due date d) exists in the researched model. The completion of a job after the planned delivery time d generates a penalty cost. Two different tardiness penalties are part of the objective function: a unit tardiness penalty and a fixed job-dependent tardiness penalty, as stated in Objective Function (3.1)

$$\min \sum_{j=1}^{n} w_j^1 \max\{0, d - C_j\} + w_j^2 \max\{0, C_j - d\} + w_j^3 U_j. \tag{3.1}$$

The above parameters w_j^1, w_j^2 and w_j^3 are the earliness cost per unit of job j, the tardiness cost per unit of time of job j, and the fixed penalty cost that applies when job j is tardy. The authors consider two versions of the problem: In the first one, the common due date d is a parameter; in the second one, d is a decision variable. Pseudopolynomial DP algorithms solve both versions. Furthermore, the authors describe a polynomial DP for the case where all cost types are equal for all jobs, with d as a variable and a pseudopolynomial DP for d as a parameter.

Li et al. (2005) investigate a problem of assembly scheduling with air-transportation in a consumer electronics supply chain. The problem is to coordinate transportation and assembly decisions. In this study, the problem is formulated as two sub-problems and solved by a hierarchical approach. The first subproblem requires orders to be allocated to transports to minimize earliness, tardiness, and transportation costs. The second subproblem seeks a schedule that minimizes the average waiting time before transportation for the allocated orders. The scheduling problem is denoted as $1 \mid r_j \mid T, pre \mid K \mid \sum w_j^1 H_j + w^2 \sum(U_j)$and solved heuristically by a dispatching rule. Note, while Li et al. (2005) describe the allocation problem formally, the scheduling problem's details are left out. A second study by the same authors (Li et al., 2006) considers an identical problem that additionally incorporates uncertainty. Specifically, the machine environment is subject to process delays for reasons such as machine breakdowns or shortages of materials. A more detailed explanation of

[2] Note. The authors make no distinction between **weak** and **strong** $\mathcal{NP}$-**hardness**.

the scheduling subproblem appears in this publication. The authors segment the whole planning horizon into planning days and formulate a separate MILP for each planning day to model the allocated jobs' scheduling problem. During each planning day, the authors assume prior knowledge of occurring delays. Therefore, the MILP models have no stochastic components. The scheduling problem aims to build a schedule that adheres to the assignment stage's allocation while minimizing earliness penalties. Since machine delays occur (but are not known during allocation), lateness fees are part of the objective function. Moreover, commercial flights transport late orders at additional costs. As the considered scheduling problem is **weakly $\mathcal{NP}$-hard**, the authors developed a scheduling heuristic. Both studies consider only scheduling decisions with delivery dates pre-computed in an initial planning stage.

3.3.2 Models with Equidistant Deliveries

This section discusses models with decidable job delivery departure times that consider a special structure of fixed delivery departures: Departure times occur in intervals of equal length. The considered models define the time between two consecutive departure dates as a constant duration τ. Therefore, there are departures at times $\tau, 2\tau, ..., n^T\tau$ where $n^T\tau \geqslant \sum_{j=1}^{n} p_j$ is required for feasible schedules to exist.

One of the first studies of this type was by Matsuo (1988). The considered problem $1 \mid \bar{d}_j \mid 1 \mid 1 \mid \sum w_j \widehat{T}_j + PC^R + PC^O$combines determining overtime utilization and job sequencing over a periodic planning horizon with periods of equal length. The model includes tardiness costs for jobs that do not meet their due dates. Increasing machine capacity is allowed for each planning period, although using overtime generates higher costs than regular time usage. As the model does not consider transportation costs or delivery restrictions, completing a job in a planning period triggers delivery at the end of the same period. The authors prove the **strong $\mathcal{NP}$-hardness** of the problem in its general form, and a two-stage heuristic solves the problem: First, a problem relaxation determines an initial (possibly infeasible) solution to the general problem; then, a feasible schedule is generated from this information and subsequently improved.

The problem that appears in Lee and Li's 1996 research also assumes an equidistant structuring of departure dates. The researched model's objective is to minimize *due date costs* and inventory holding costs. The five-field notation describes the model as $1 \mid (p_j = p) \mid T, \mathtt{eqdist} \mid 1 \mid \sum H_j + \beta \sum D_j$. First, the due date cost in the considered model is a step-wise increasing cost function that increases with each consecutive departure date. Completing a job at $(i-1)\tau < C_j \leqslant i\tau$ triggers delivery at τ with due date costs βi. Note that the due date cost objective is equivalent to $\beta \sum D_j$, since for $D_j = t_i$, $D_j/\tau = i$. The objective is consequently not denoted differently. Second, the objective jointly minimizes the total holding cost value equivalently to Chand and Schneeberger (1988). Both cost types are job agnostic in the considered models. The model in Lee and Li's 1996 work is a generalized version of the model by Chhajed (1995) that considers only instances with two due dates $\{\tau, 2\tau\}$. Lee and Li (1996) note that, for a setting $\beta = 0$, and under the constraint that the interval length τ is larger than the sum of the processing times of all jobs, the problem is equivalent to the parallel machine problem $Pm||\sum C_j$. Although this problem is generally **strongly $\mathcal{NP}$-hard**, Lee and Li (1996) derive a DP formulation for the problem with a fixed number of due dates that solves the problem in pseudopolynomial time. The authors provide a $\mathcal{O}(n^3)$ time DP algorithm for the special case with identical job processing times.

Fan (2010) considers a single machine problem with equidistant or arbitrary deliveries denoted as $1 \mid\mid T, (eqdist) \mid 1 \mid D_{\max} + \sum w_j V_j$. The objective is to minimize the maximum delivery time ($D_{\max}$) and the total weighted number of early jobs ($\sum w_j V_j$). As this variant is **strongly $\mathcal{NP}$-hard**, the authors developed approximation schemes and demonstrated that the version with arbitrary deliveries has no PTAS with a fixed performance ratio unless $\mathcal{P}= \mathcal{NP}$ is true. The authors consequently studied the equidistant version of the problem and describe a PTAS with a performance ratio of 3/2, which is the best ratio attainable (unless $\mathcal{P} = \mathcal{NP}$) for this problem.

Liu and Hsu (2015) discuss the problem $Jm \mid a_j \mid T, eqdist \mid 1 \mid \sum w_j^1 \widehat{F}_j + \sum w_j^2 H_j$ with equidistant delivery departure dates. The study stands out in the literature discussion for two reasons. First, the authors consider a job shop environment; therefore, the study is one of the few works that deviate from the single machine assumption. Second, the study considers a dynamic scheduling problem with dynamic job arrivals and it is hence the only dynamic IPODS-FD formulation found during the literature search. The authors assume that jobs arrive dynamically over time and are processable on arrival. The objective function minimizes two criteria: the sum of weighted flow times and the total holding costs. In their experimental study, the authors compare different dispatching rules, which select the best job available for processing at each point in time. As future orders are unknown before arrival in the dynamic context, the dispatching rules only select the next operation by considering the revealed orders.

Surprisingly, the authors do not mention the implemented scheduling procedure that was used on the basis of the computed priorities. As the objective also minimizes the holding costs, completing jobs close to the delivery departure time is cost efficient. However, scheduling jobs later than necessary may delay the processing of dynamically arriving jobs. The exact mechanics of the presented approach consequently are not entirely comprehensible. Furthermore, although the computational study compares the results of the different dispatching rules, the quality of the final schedules (by comparing the schedules to those produced with full information) is not discussed.

3.3.3 Models with Arbitrary Delivery Departure Dates

In models with arbitrary delivery departure dates, the distance between two consecutive dates may vary. Yang (2000) uses the term “generalized deliveries” to describe such models. This paper analyzes a single machine problem with two alternative objective functions: the minimization of the sum of holding costs $\left(\sum (w_j) H_j\right)$ and the minimization of the maximum holding cost $\left(\max(w_j) H_j\right)$. The problems are denoted as $1 \mid\mid T \mid 1 \mid \sum (w_j) H_j / \max(w_j) H_j$. In the investigated problems, there are n^T delivery departure dates, where $t_{n^T} = \sum_{j=1}^{n} p_j$. Hence, the study does not deal with idle times on the machine. The k-th delivery batch is defined as job set $B_k = \{j \mid t_{k-1} < C_j \leqslant t_k\}$ and therefore $D_j = t_k \ \forall j \in B_k$. In general, even the unweighted problems are **strongly $\mathcal{NP}$-hard**. If all n jobs have equal processing times, then the problems can be modeled as assignment problems that are solved in $\mathcal{O}(n^3)$ time optimally. The authors note that the assignment problem's particular structure allows for an even faster algorithm that minimizes the WE by sorting the jobs accordingly. Due to this approach, the worst time complexity of $\mathcal{O}(n \log(n))$ is attainable.

Hall et al. (2001) present numerous scheduling problems with generalized deliveries. The authors study the complexity and provide solution approaches to classic scheduling problems extended by considering fixed departure dates. As described in Section 3.2, Hall et al. define time-oriented scheduling variables based on the departure time instead of the completion time of a job. The problems $\alpha \parallel T \mid 1 \mid \gamma$ with $\alpha \in \{1, Pm, P, F2, F3, J2, J3, O2, O3, O\}$, and the objectives ($\gamma$) (weighted) departure time $(w_j)D_j$, maximum lateness $\widehat{L}_{\max}$, (weighted) number of tardy jobs $(w_j)\widehat{U}_j$, and (weighted) tardiness $(w_j)\widehat{T}_j$ are discussed. The study provides complexity-related results and presents approaches for parallel machines, flow shops, job shops, and open shops. Most of the analyzed problems are either **weakly $\mathcal{NP}$-hard** when the number of departure dates is a constant, or **strongly $\mathcal{NP}$-hard** when the number of delivery dates is part of the input. The authors formulate several pseudopolynomial-time DP algorithms.

Stecke and Zhao (2007) discuss eight model variations for a production and distribution problem in a MTO environment for a single product. The problem can be described as $1 \mid pmtn, \bar{d}_j \mid T, (split) \mid 1 \mid \sum f(D_j)\}$, with $f(D_j)$ denoting a cost function that is based on the delivery time of the individual jobs. In the considered problems, the manufacturer adopts a commit-to-delivery mode; that is, the manufacturer commits to a delivery due date for an order. The authors consider the possibility of using several shipping modes that differ in travel time and costs. Specifically, they consider the cost structure of different transportation modes of 3PL providers such as FedEx and UPS (e.g., overnight, one-day, or two-day delivery). Furthermore, they investigate scenarios with two different shipping cost functions that depend on the shipping time, two modes of delivery (partial delivery allowed or forbidden), and either a single customer location or multiple customer locations. In the presented models, there exists an average production rate of p in items per hour and a daily production capacity c. The production plan comprises producing Q_j items for each order j with delivery departure deadline $\bar{d}_j$ throughout a planning horizon with m days. A solution to a problem instance decides on the number of items produced for customer order j on production day i. Although the models incorporate deadlines, Stecke and Zhao assume that the production capacity is sufficiently large such that constructing a feasible schedule is non-problematic. The shipping date D_j for order j with $D_j \leqslant d_j$ and the used shipping mode for order j are dependents on the production decisions. In the problem with linear shipping costs and allowed partial delivery, the shipping cost for an order j is $b - a(d_j - D_j)$, where value b is the cost per item for overnight shipping, and value a is the dollar value of increasing the shipping time by one more day. Early production and delivery are consequently cost-beneficial in a schedule. Furthermore, some model variations allow partial delivery; in these variations, the optimal delivery date is at the end of the production day. The authors show that a non-preemptive earliest due date (EDD) schedule is optimal in this case. For nonlinear shipping costs that follow a concave cost function, a modified EDD rule constructs optimal schedules. When partial delivery is not allowed, the problem becomes **strongly $\mathcal{NP}$-hard** for linear and nonlinear cost functions. Therefore, the authors offer heuristic solution methods.

Carrera et al. (2010a,b) consider the combination of fixed delivery departure dates and resource availability for jobs. To start the production of a job, the required resources must be present. The production resources arrive several times during the planning horizon, and each production process consumes a certain amount of available resources. The authors describe the arrival of components as a fixed cumulated staircase curve, and they consider the following three models:

- $1 \mid res \mid 1 \mid 1 \mid C_{max}$: The only delivery occurs at time C_{max}. This model technically does not consider fixed departure dates since the value of C_{max} cannot be determined beforehand.

- $1 \mid d_j = d, res \mid 2 \mid 1 \mid (1 + \sum w_j \widehat{U}_j) \cdot C_{max}$: A single fixed departure date t and a second departure at time C_{max} exist. In this scenario, a job j that is completed after time t departs instead at time C_{max}. A lateness penalty $w_j \widehat{U}_j$ penalizes this late delivery. In contrast, jobs that are completed before or at C_{max} contribute no costs to the objective value.

- $1 \mid res \mid T \mid 1 \mid \sum w_j D_j$: In the third scenario, there is a set of fixed departure dates, where the last departure date is sufficiently large, such that all jobs are completed beforehand.

The authors develop lower bounds and approximation-based upper bounds for the three $\mathcal{NP}$**-hard** problems.

Fu et al. (2012) study problem $1 \mid r_j, c_j, \bar{d}_j, accept \mid T, Q \mid 1 \mid \sum R_j$, which maximizes the revenue of jobs. For each job j, there exists a production time window $[r_j, \bar{d}_j]$ on a single machine and additionally a latest departure date $\widehat{\bar{d}}_j$. The time restrictions are assumed to be hard, but rejecting (i.e., not producing) jobs is allowed. Each job j is associated with a size c_j and a profit p_j. A transport at delivery date i has a maximum capacity of C_i. In this model, a job is only profitable if it is completed within its production time window and starts delivery at its promised departure date. The authors also study the version with only one departure date. Both problems are **strongly $\mathcal{NP}$-hard**. The problem with one departure date is solved heuristically by a PTAS. Furthermore, an extension to this PTAS solves the problem with multiple departure dates, assuming that the number of departure dates is a constant.

Seddik et al. (2013) study the problem $1 \mid (r_j), (pmtn) \mid 2/T \mid 1 \mid \sum \widehat{\mathcal{V}}_j$[3] with release dates and a cumulative payoff function. The function rewards the early completion of jobs.

[3] The notation $\gamma = 2/T$ refers to two variants of the problem: The first variant specifically deals with two delivery departure times, while the second variant allows an arbitrary number of delivery departure times specified by the input.

For a given set of n^T delivery departure times $\{t_1, t_2, \ldots, t_n^T\}$ completing a job before time t_1 realizes a payoff of n^T; a completion after t_1, but before t_2, realizes a payoff of $n^T - 1$; and so on. The authors prove the **strong $\mathcal{NP}$-hardness** of the general problem and therefore study relaxed versions of this general problem. The shortest processing time (SPT) rule optimally solves the problem without release dates, and the shortest remaining processing time (SRPT) rule solves the problem with release dates and preemption. The problem with equal processing times with n jobs is solved in $\mathcal{O}(n\ \log(n))$ time. The authors provide a polynomial-time algorithm based on the Moore-Hodgson algorithm (Moore, 1968) for solving $1||\sum U_i$ for the special case with a single delivery date. A DP algorithm optimally solves the problem with two delivery dates.

The model by Ma et al. (2013) considers the transport of items with variable costs that depend on the departure date. The problem can be stated as $1 \mid r_j \mid T \mid 1 \mid \sum w_j^1 H_j^1 + w_j^2 H_j^2 + \sum w_j^3 \widehat{T}_j + TC^M$. The objective function considers two types of inventory holding costs: The first one rates storage in a warehouse at the manufacturing site after processing completes until departure; the second one considers storage at the transportation destination in a distribution center until the warehouse's pickup is due. Additionally, tardy arrivals at the warehouse are penalized, and shipments are associated with a variable shipping cost that differs by shipment but is equal for all jobs. The problem is solved by a GA.

Mensendiek et al. (2015) present the scheduling problem $Pm \mid\mid T \mid 1 \mid \sum w_j \widehat{T}_j$ on identical parallel machines to minimize total tardiness while considering fixed departure dates. The model comprises no further restrictions. As the problem is **weakly $\mathcal{NP}$-hard**, the authors propose a B&B approach and two heuristics to solve it. The considered problem differs from the ordinary production scheduling problem $Pm||\sum T_j$ by considering the delivery-equivalent objective $\sum \widehat{T}_j$.

3.3.4 Models with Multiple Customers

In multi-customer models, different customers G order products, and produced items must therefore be transported to different customer locations. In this type of model, there is a separate set of delivery departure times T_g for each customer g, and a vehicle that arrives at such a customer-specific departure time $t \in T_g$ consequently only transports produced items that belong to the customer g if routing is not part of the model.

Wang et al. (2005) investigate a practical problem of sequencing the processing of incoming mail. In the considered problem $1 \mid r_j \mid T, C \mid G \mid Cap$, multiple mail trays from different origin locations must be processed on a single machine without preemption. Each mail tray comprises mail that is to be delivered to multiple destinations, and all trays from a specific location arrive at the same time. Furthermore, for each destination, a truck leaves at a fixed leaving time. The goal is to process the incoming mail such that the unused truck capacity is minimized. The problem formulation allows for unprocessed trays, and several dispatching rules are compared to solve the daily problem.

Li et al. (2017) present a model that acknowledges orders from and deliveries to multiple customers. The considered problem

$$1 \mid [\hat{a}_j, \hat{b}_j], noidle \mid T, (force\ batch) \mid K \mid \sum w_j^1 H_j + TC^{N,M}$$

arises for manufacturers in the steel industry that adopt an MTO production strategy. In this setting, customers dictate a delivery time window for placed orders, and the manufacturer relies on 3PL providers to transport the ordered items. The presented model assumes that customers' orders comprise multiple jobs that require processing on a single machine. After a job is processed, a transporter delivers items to the specific customer, starting at dedicated departure times. The model restricts the transportation of an order to a subset of departure times within a time window ($[\hat{a}_j, \hat{b}_j]$). The authors consider a case where the delivery of an order can be split over multiple shipments, and a non-split case where all jobs of an order must be delivered together. Additionally, the authors impose a $\mathtt{noidle}$-constraint for the machine, which they justify as a requirement in the studied steel industry. The objective considered in this study is to minimize the total inventory and delivery costs. The model's holding costs originate from the produced items' literal weight in the holding cost function. Moreover, the transportation costs are weight and mode dependent ($TC^{N,M}$); that is, at each departure date, the cost to transport the produced items of a job varies. Furthermore, the model allows the definition of job weights to prioritize a job compared to others. Both variants of the problem are **strongly $\mathcal{NP}$-hard**, while some relaxations are easier to solve. A CG approach is combined with a TS heuristic to solve the problem optimally or near optimally.

Han et al. (2019) consider a three-stage supply chain with a single supplier, a single manufacturer and multiple customers. The problem is represented as

$$F(1,1) \parallel T, (Q), split \mid K \mid \sum w_j^1 H_j + \sum w_j^2 \widehat{U}_j + \sum w_j^3 (1 - \widehat{U}_j) + TC^V$$

with $F(1,1)$ denoting the supplier-manufacturer environment. Orders from customers comprise multiple jobs that must be processed on a single machine at the supplier's site and then on a single machine at the manufacturer's site. Afterward, the manufacturer stores finished jobs in inventory for later pick up by vehicles at fixed departure times. The considered model does not include release dates, but delivery deadlines. Late deliveries are forbidden by the model; instead, jobs can be rejected by paying a rejection fee. Rejected jobs are not part of the generated schedules. Apart from the rejection fees, the objective function minimizes the total holding cost and total transportation costs of finished jobs. Furthermore, multiple possible departure times exist for each customer destination. At each of these departure times, a transporter with limited weight capacity is available that may transport finished jobs with individual weight. Activation of a transport generates a date-dependent cost value that is independent of the number of transported jobs. The authors provide complexity-related results for variations of their model. Most of the problems are **strongly $\mathcal{NP}$-hard** in the general case, and **weakly $\mathcal{NP}$-hard** for a fixed number of orders (=customers). Some problems are shown to be polynomial-solvable by a DP algorithm.

The models presented by Wang et al. (2020a,b) consider a port-centric supply chain with multi-modal transportation that cannot be accurately captured by the introduced five-field notation. Customer orders are produced at an inland industrial park, transported to a single origin port by truck or train and shipped to multiple destination ports. Orders have material requirements that have release dates and material storage costs. At the manufacturing site, multiple production lines (i.e., machines) exist that work in parallel. The authors consider different production speeds and restrict the set of producible orders for each production line. Finished orders can be transported either by a regularly arriving train or on demand by trucks to the origin port, where orders can be stored for a storage cost rate. Ships headed to the different destination ports arrive at the origin port, based on a timetable. The model considers unit production costs and unit maritime transportation costs for each order, railway transportation costs and truck transportation costs per delivery and unit storage costs at the factory and at the origin port, and earliness and tardiness penalty costs. A solution to this problem decides on the assignment of orders to production lines, the schedules for the production lines, the transportation mode and time to transport finished orders to the origin port, and the assignment of orders to cargo ships. The authors describe a GA to heuristically solve this **strongly $\mathcal{NP}$-hard** problem.

3.3.5 Models with Batch Deliveries

This section focuses on models that explicitly encourage the bundling of jobs into delivery batches due to the underlying cost structure. The models discussed so far all assume either no or only job-related (as in Lee and Li (1996) or more recently Li et al. (2017)) costs for transportation. In these models, a job's delivery time is consequently derived directly from the completion time, which is the earliest available time after processing completes. While some models consider variable transportation costs (e.g., Stecke and Zhao (2007) and Li et al. (2017)), bundling jobs has no intrinsic cost benefit in these models.

Leung and Chen (2013) present problems that deal with delivery bundles. The considered problems are $1 \parallel T, C \mid 1 \mid \gamma$ with $\gamma \in \{\widehat{L}_{max}, \widehat{L}_{max} + \sum B_i, \alpha\widehat{L}_{max} + \beta \sum B_i\}$. The presented models assume that a vehicle arrives at each delivery date and can only transport a limited number of orders. The objectives considered are (1) $\widehat{L}_{max}$ and more importantly, (2) $\widehat{L}_{max} + \sum B_i$ and (3) the weighted sum of $\widehat{L}_{max} + \sum B_i$, where B_i is the number of deliveries. The Objectives (2) and (3) are hierarchically structured, and schedules are hence sought that minimize the (weighted) L_{max} objective first and the number of deliveries second. For all three variants, the authors developed polynomial-time algorithms.

Tyagi et al. (2013) propose a model that pursues the minimization of fixed transportation costs and total holding costs. The authors use a nonlinear model to formulate the problem, which is solved by the Lingo 9 software package[4]. However, the authors do not present a computational analysis of the implemented model.

3.4 Conclusion

This chapter presented the integration of outbound distribution decisions with fixed departure times in production scheduling models. Section 3.1 provided a brief introduction to scheduling problems that incorporate delivery decisions for settings with fixed delivery departure times–so-called IPODS-FDs. To categorize the current research that studies

[4] https://www.lindo.com/

such problems, Section 3.2 presented a classification scheme, based on the one presented by Chen (2010), that covers a wide variety of model assumptions in a concise format. Thereafter, Section 3.3 discussed the studied scheduling models and classified them using the scheme introduced beforehand. The discussion of the models was structured according to the common characteristics . This chapter concludes with a description of the current state of research in Section 3.4.1 and research gaps and possibilities in Section 3.4.2.

3.4.1 The Current State of Research

The conducted literature review discussed 25 publications that were mostly published in the previous decade. Therefore, the extension of scheduling models with fixed departure dates is a relatively new and not broadly studied research field. However, the published papers offer models and solution methods for many different applications and assumptions. This section briefly recapitulates the different characteristics regarding the integration of delivery decisions.

The structure of delivery departure dates considered in the different models varies greatly. The first variation is the considered number of departure dates that are specified by the input. Early works by Matsuo (1988), Lee et al. (1991), and Chhajed (1995) consider problems with one or two departure dates. In some publications, a relaxation of the general problem with multiple departure dates to a problem with only one or two departure dates enables the construction of optimal polynomial-time algorithms. Most models consider a finite number of available departure times that either are equidistant to one another or have an arbitrary structure. Most of the models consider only a single customer or a common departure schedule for all ordered jobs. This assumption is extended by the recent multi-customer models (Li et al. (2017), Han et al. (2019), and Wang et al. (2020a,b)). These publications define individual, customer-dependent departure schedules. A job ordered by a certain customer g can only depart with a vehicle that moves to the customer location of g.

Some of the authors assume that transports are subject to capacity constraints. Models presented by Chen (2010) and Leung and Chen (2013) limit transports to a number of jobs, which they justify by considering equally sized goods. In addition, recent publications Han et al. (2019) and Wang et al. (2020a,b), and an earlier publication by Fu et al. (2012) considers different weights for the produced goods and limit vehicle capacity by total weight.

Many of the discussed models do not consider any transportation costs and focus on completion-time or departure-time-oriented objectives. To the best of our knowledge, Stecke and Zhao (2007) first studied models that focus on transportation-related costs. The cost structure they considered is job dependent, and the number of jobs that depart at the same date is of no consequence for the resulting costs. Apart from the job-dependent aspect, the cost structure may depend on the used transportation mode and might be influenced by the customer destination. The assumption made in Stecke and Zhao (2007) is that the vehicle(s) that arrive at a departure time t_i transports the products first to a distribution center that handles the downstream transport to customer locations. Li et al.

(2017) define eligible departure times for each order that comprises multiple jobs. The transportation costs are defined by cost values $c_{i,t}$ for an order i and departure time t. The model does not cover the cost benefit of transporting parts of the order jointly on the same departure time.

The model proposed by Leung and Chen (2013) integrates delivery batching to minimize the number of transports and, consequently, transportation costs. In this model, using fewer distinct departure dates (i.e., fewer vehicles) to pickup produced items reduces the transportation costs. Note that holding costs are not considered in conjunction with batch delivery costs. Furthermore, the multi-customer models by Han et al. (2019) and Wang et al. (2020a,b) consider varying transportation costs that depend on the job and the chosen departure date but do not include fixed transportation costs.

3.4.2 Research Gaps and Opportunities

Analyzing the relevant literature revealed the following research gaps:

- Only a few papers consider hard time window restrictions in their presented models. Many of the studied papers consider no release-date restrictions, which exempts the use of the solution approaches in applications with prior production stages or inbound distribution activities. Furthermore, production or delivery deadlines are only present in a handful of papers. Fu et al. (2012) study a model with release dates and deadlines but avoid the arising issue of generating feasible schedules by rejecting jobs. Including hard time window restrictions into IPODS-FD is consequently an area that requires further study.

- The trade-off between holding and batch transportation costs is not well studied. On the one hand, a few considered models include a finished product holding cost that measures the storage duration of products between completion and the start of delivery – minimizing these costs in isolation results in many transport activities throughout the considered planning period. These activities cause numerous separate handling operations to retrieve the stored products and, depending on the contracts with the 3PL provider, per delivery costs. On the other hand, the model by Leung and Chen (2013) that minimizes transports does not consider holding costs, and the average holding time of finished products is consequently relatively high to reduce the number of transports. A combined consideration of the two interlinked cost types, namely, holding costs and batch delivery costs, is not well studied.

- The only model that describes a dynamic and not a static planning environment is formulated Liu and Hsu (2015). Hence, for the other models, all orders for the planning horizon are known beforehand, and the orders or transports' parameters are certain and do not change. Stochastic models with random parameters and dynamic scheduling models are consequently not well researched for IPODS-FDs. Therefore, many opportunities exist to extend the research of such problems in both areas.

The stated research opportunities motivate the formulation of two scheduling models. First, the scheduling problem with flexible customer-dependent delivery dates that pursues the minimization of holding and transportation costs (SFDDHT) (see Chapter 4). Second, a dynamic problem formulation with dynamically arriving orders in Chapter 7 named scheduling problem with flexible customer-dependent delivery dates and dynamic job arrivals that pursues the minimization of holding and transportation costs (D-SFDDHT).

Chapter 4

The Scheduling Problem with Flexible Definable Departure Dates

The contents presented in this chapter also appear in the work of Bachtenkirch and Bock (2022), titled Finding efficient make-to-order production schedules, which is published in the European Journal of Operational Research. To avoid impairing the reading flow, citations do not reference this article continuously throughout this chapter. However, when introducing a mathematical statement, such as a lemma, proposition, or proof, from the stated paper, the text includes a reference. Moreover, tables, figures, and algorithms that are taken from this paper are referenced as per usual.

The previous chapter reviewed the existing literature concerning IPODS-FD models and discussed the different model assumptions. While the presented studies cover a wide variety of applications, the joint optimization of the holding costs and batch delivery transportation costs of finished goods is an area that has not been covered by the published research.

This chapter introduces the SFDDHT, that is a novel scheduling problem to map various BTO-SC and MTO applications. The proposed model comprises the following notable aspects: the minimization of finished goods' holding costs and batch delivery transportation costs, batch delivery to multiple customer locations, hard production and delivery time windows, and the enabling of machine idle times.

The SFDDHT is informally introduced in Section 4.1 to illustrate the possible applications of the proposed model. An introduction to the problem notation and parameters in Section 4.2 follows the initial informal description. Then, Section 4.3 formulates the optimization problem as a MILP that unambiguously defines the constraints and objective of the SFDDHT. Lastly, Section 4.4 resolves the problem's complexity status by polynomial reduction from a **strongly** $\mathcal{NP}$**-hard** problem.

4.1 Problem Description and Application Areas

This section introduces the SFDDHT alongside its intended areas of application: Several customers task a single manufacturer in a JIT environment with the production of products for multiple orders and their timely distribution via 3PL transportation. The proposed model is concerned with a single machine of the manufacturer's production system, which finalizes the product building or assembly in an MTO or a BTO fashion with multiple previous production and assembly stages.

Due to this underlying complex environment, the model enforces **production release dates** for the jobs that are a consequence of previous operations and raw material or component availability (Jamili et al., 2016). We consider applications in the area of JIT production in MTO or BTO supply chains that usually try to avoid extensive inventories of components and materials. A sufficient inventory to produce or assemble all final products at any given point time hence does not exist.

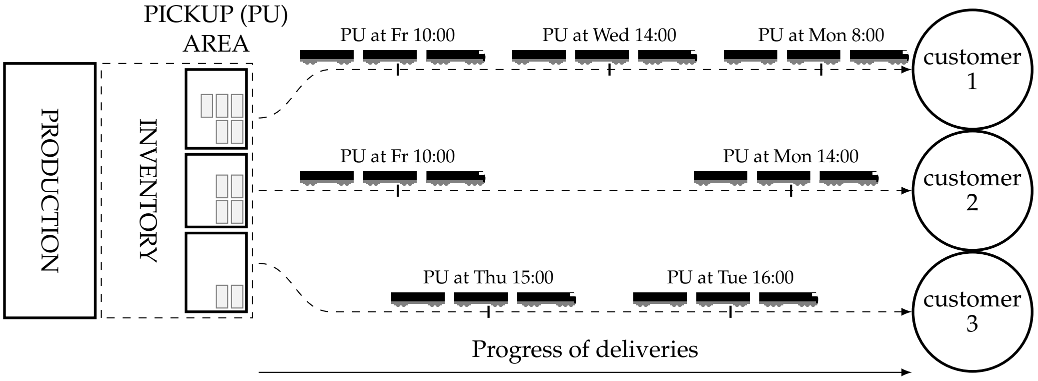
The proposed model additionally includes **production and delivery deadlines**. This modeling assumption makes the model especially suited for applications with perishable products, for example in the context of the production and distribution of fresh fruits and vegetables (c.f. Chen and Vairaktarakis, 2005), concrete paste (c.f. García and Lozano, 2004, 2005), and perishable chemical compounds (c.f. Devapriya et al., 2006; Armstrong et al., 2008; Geismar et al., 2008). A deadline indicates the last possible delivery departure at which the distributor guarantees product arrival without deterioration below an acceptable level. The newspaper industry is another application area for deadline models. Newspapers or magazines are usually offered for a limited amount of time and replaced by newer editions daily, weekly, or monthly, therefore, the distribution of orders is strongly encouraged to take place in a given time frame (c.f. Russell et al., 2008).

The reliance on 3PL for distribution is modeled by **customer-specific fixed delivery departure dates**. For each customer destination, the manufacturer and 3PL provider agree on a delivery timetable with several product pickup and delivery departure times at the production facility. The 3PL provider only sends a vehicle to the manufacturing site at a pre-agreed time if the manufacturer calls for a pickup at this time. Transportation fees consequently arise for each pickup request. Alternatively, a nearby train station or harbor with a fixed transportation schedule is available to the manufacturer. Figure 4.1 by Bachtenkirch and Bock (2022) illustrates the modeled environment.

In all cases, the model distinguishes between orders with different customer locations in terms of delivery departure times and transportation costs. It assumes that the transportation mode is identical for all available transports to the same customer location. Additionally, it does not consider capacity limitations for vehicles. The assumption is that the agreement between the 3PL partner and the manufacturer specifies suitable vehicles that can carry a sufficient number of products. With the mode and capacity assumptions in mind, the model includes **fixed transportation costs** only. Each pickup with products destined to the same customer location costs only a fixed monetary value independent of the number of jobs transported. The variable, namely, product-dependent-transportation costs, is not part of the model. This modeling decision is justified by assuming that these costs are constant and independent of the departure date assignment and delivery batch configuration. Note that the fixed transportation cost may represent general administrative and handling costs that the 3PL provider charges for each delivery.

Figure 4.1

Illustration of the Considered Problem



Note. Reprinted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier.

One of the objectives of the JIT philosophy is the minimization of work-in-process inventory (Huang et al., 1983). Purchasing and manufacturing parts close to the assembly activity that requires these parts accomplishes this objective. Furthermore, the finished products' production and outbound distribution should be synchronized (Schonberger and Gilbert, 1983). The model includes **holding costs for the finished goods**. Considering this cost type and the batch delivery transportation costs, a trade-off between minimizing both objectives arises. On the one hand, an increase in the number of delivery batches allows for few products to be produced near the assigned delivery departure time and reduces holding costs while transportation costs increase. On the other hand, lowering the number of delivery batches and transportation costs increases the average holding time of finished products, which in turn increases holding costs.

The model explicitly allows **idle times on the machine** to encourage the minimization of job holding times before delivery. This assumption is especially useful for planning previous stages in the manufacturing process. A feasible schedule can, for example, aid in planning inbound-transportation activities for critical production parts that are not needed at the initial release date but at the scheduled start time of the corresponding job. In many of the reviewed models in Section 3.3, idle times are not considered. This neglect occurs due to the lack of holding costs or earliness costs in the objective function. Additionally, some models, for example the one presented by Li et al. (2017), prohibit idle times due to the application defined continuous production requirement.

4.2 Problem Definition and Notation

This section formally introduces the problem parameters and assumption of the SFDDHT. The optimization problem seeks to find an integrated production and delivery schedule that defines machine processing intervals and delivery departure times for the set of *jobs*

$$N = \{1, 2, \dots, n^N\},$$

where parameter n^N is the *total number of jobs* considered in the problem instance. The jobs must first be processed on a *single machine* and then transported to

$$G = \left\{1, 2, \dots, n^G\right\}$$

customers, where n^G is the total *number of customers* in a problem instance. Each job $j \in N$ represents an order of a single customer $g \in G$; therefore, $n^N \geqslant n^G$ holds. The corresponding customer $g \in G$ of a job $j \in N$ is given by a surjective function

$$g^N : N \to G.$$

For example, the expression $g^N(5) = 2$ states that the job with index 5 is of the customer with index 2. Additionally, the n^N_g jobs of customer $g \in G$ are

$$N_g = \left\{ j \mid j \in N, g^N(j) = g \right\}.$$

For convenience, let $n_{g,j}$, with $g \in G$ and let $j \in \{1, \dots, N_g\}$ be the j-th job of customer g. Jobs are indexed such that $n_{g,j} = \sum_{h=1}^{g-1} |N_h| + j$ holds for each job $j \in N_g$ and all customers $g \in G$. Each job $j \in N$ requires processing on a single machine for p_j units of processing time. The processing of jobs is non-preemptable. Therefore, once the processing of a job $j \in N$ starts on the machine, the processing continues for p_j time units to completion in a feasible schedule. The production of a job is disallowed outside of the time window $\left[r_j, \bar{d}_j\right]$, which is defined by the *production release date* r_j and the *production deadline* $\bar{d}_j$. Simultaneously, the deadline $\bar{d}_j$ coincides with the latest feasible delivery departure time for job j. The flexible departure times are defined as follows: In total, n^T *transports*

$$I^T = \left\{1, 2, \dots, n^T\right\}$$

occur with delivery departure time t_l, with $l \in I^T$ and $t_1 \leqslant t_2 \leqslant, \dots, t_{n^T}$. The l-th transport is allotted to a specific customer $g \in G$ and only completed jobs of customer g can consequently depart with this transporter. The customer of a transport l is defined by function

$$g^T(l) \colon I^T \to G.$$

The *delivery departure times* for a customer g constitute the set

$$T_g = \left\{ t_l \mid l \in I^T, g^T(l) = g \right\}.$$

Additionally,

$$I^T_g = \{l \mid l \in I^T, g^T(l) = g\}$$

are the *transports for customer* $g \in G$, n^T_g the *number of transports to customer* g, and $t_{g,l}$ the l-th *delivery departure time* of customer g with $g \in G$ and $l \in I^T_g$. The customer-dependent delivery departure times are given in increasing order of time: $t_{g,1} < t_{g,2} < \cdots < t_{g,n^T_g}$.

A schedule or solution for the SFDDHT is unambiguously defined by setting the production *completion time* C_j and the *departure time* D_j for each job $j \in N$. Two cost types are part of the objective function: finished product holding costs and batch delivery transportation costs. A job $j \in N$ that completes processing, but does not immediately start transportation, is held in inventory until its delivery occurs. The storage of such a job induces the finished product *holding cost* c^H_j per unit of holding time. The first part of the objective function is thus to minimize the sum of holding costs

$$\sum_{j \in N} c^H_j (D_j - C_j). \tag{4.1}$$

The proposed model allows for the bundling of jobs from identical customers to delivery batches. That is, multiple jobs that depart at the same delivery departure time use the same transportation vehicle. For each transport, the manufacturer pays a fixed transportation fee. This cost value is identical for all transports I^T_g of customer $g \in G$ but may differ between customers. Furthermore, the transportation cost c^T_g is imposed for each used transportation vehicle and is independent of the number of jobs transported. The capacity of each transportation vehicle is assumed to be sufficient for any number of jobs. The total transportation cost value can be described as

$$\sum_{g \in G} c^T_g \left| \{ D_j \mid j \in N_g \} \right|. \tag{4.2}$$

Expression (4.2) counts the number of distinct departure times for each customer and multiplies this number by the customer-dependent cost value c^T_g.

A feasible schedule for the SFDDHT processes each job $j \in N$ inside its time window $[r_j, \bar{d}_j]$ on the machine and assigns each job to an eligible delivery departure time between the job's production completion time and its deadline. An optimal schedule is a feasible schedule that minimizes the sum of the total holding (4.1) and transportation costs (4.2). The problem is written as

$$1 \mid r_j, \bar{d}_j \mid T \mid K \mid \sum w^1_j H_j + \sum w^2_i B_i$$

in the five-field notation introduced in Section 3.2. The problem's symbols and notation are also listed in Table 4.1 in compact form.

Table 4.1

Notation for the SFDDHT

Parameters:		
n^N	Number of jobs	
n^G	Number of customers	
n^T	Number of available departures	
n^T_g	Number of available departures for customer g	
n^N_g	Number of jobs from customer g	
N	Set of jobs $N = \{1, 2, \ldots, n^N\}$	
G	Set of customers $G = \{1, 2, \ldots, n^G\}$	
N_g	Set of jobs from customer $g \in G$, with $N_g \subseteq N$	
$n_{g,j}$	The j-th job from customer g	
I^T	Index set of the available departures $I^T = \{1, 2, \ldots, n^T\}$	
I^T_g	Index set of the available departures of customer $g \in G$ with $I^T_g \subseteq I^T$	
t_l	Departure time of the l-th departure, with $l \in I^T$	(in TU)
T_g	Departure times for customer $g \in G$	
$t_{g,l}$	Departure time of the l-th available departure for customer g , with $l \in I^T_g$	(in TU)
$g^T(l)$	Customer of the l-th available departure, with $g^T \colon I^T \to G$	
$g^N(j)$	Customer of the j-th job, with $g^N \colon N \to G$	
p_j	Processing time of job $j \in N$	(in TU)
r_j	Production release date of job $j \in N$	(in TU)
$\bar{d}_j$	Production and departure deadline of job $j \in N$	(in TU)
c^H_j	Unit inventory cost of job $j \in N$	(in MU)
c^T_j	Transportation cost per transport to customer $g \in G$	(in MU)
Variables:		
C_j	Completion time of job $j \in N$ in a schedule	(in TU)
D_j	Departure time of job $j \in N$ in a schedule	(in TU)
H_j	Holding time of job $j \in N$ in a schedule, $H_j = D_j - C_j$	(in TU)

TU: time units
MU: monetary units

4.3 A Mixed-Integer Linear Programming Formulation

This section formulates the SFDDHT as a MILP that is solvable by standard combinatorial optimization methods. The MILP uses the following *additional* parameters and decision variables to formulate the problem:

Additional Parameters:

$a_{g,j} \in \{0,1\}$: A binary parameter which indicates whether the products produced by the j-th job must be delivered to customer $g \in G$. The value of parameter $a_{g,j}$ is set to 1 if $g^N(j) = g$ holds, and 0 otherwise.

$B^{max}_{g,l}$: The maximal number of jobs from customer g that can depart at the l-th departure date of customer g. The values for $B^{max}_{g,l}$ are entirely computed from the production time windows of jobs and are not defined independently as a capacity restriction. The values are computed as follows:

$$B^{max}_{g,l} = |\{j \mid j \in N_g, t_{g,l} \in [r_j + p_j, \bar{d}_j]\}| \quad \forall g \in G,\ l \in \{1,2,\ldots,n^T_g\}.$$

Note that modifying this parameter to be independently definable extends the SFDDHT to express capacity restrictions, without the need to introduce additional constraints.

Binary Decision Variables:

$Y_{i,j} \in \{0,1\}$: A binary sequencing decision variable that indicates the direct predecessor of a job. The value of $Y_{i,j}$ equals 1 if job $i \in N \cup \{0\}$ directly precedes job $j \in N \cup \{n^N + 1\}$. Otherwise, its value is set to 0.
The definition of these variables includes a first dummy job with index $i = 0$ and a final dummy job with index $j = n^N + 1$.

$Z_{g,l,j} \in \{0,1\}$: A binary delivery decision variable that indicates the delivery date of a job. The value $Z_{g,l,j}$ is set to 1 if a job j departs at the l-th delivery date of customer g at time $t_{g,l}$; otherwise, it is set to 0.

$Z'_{g,l} \in \{0,1\}$: A binary batching decision variable that indicates whether a transport is used at all. The value of $Z'_{g,l}$ is equal to 1 if a transport is used, and 0 otherwise.

The MILP has the following **minimizing objective function**:

$$\min \sum_{j \in N} c^H_j \cdot H_j + \sum_{g \in G} \sum_{l=1}^{n^T_g} c^T_g \cdot Z'_{g,l} \tag{4.3}$$

The Objective Function (4.3) minimizes the total sum of holding costs and transportation costs for all jobs $j \in N$. The holding cost calculation is identical to calculation (4.1). The transportation costs are calculated with the help of binary decision variables $Z'_{g,l}$ that indicate whether the l-th transport to customer g is used or not.

The **following restrictions** (4.4 - 4.20) ensure the feasibility of a production and distribution schedule:

Sequencing Constraints:

$$\sum_{j \in N \setminus \{i\} \cup \{|N|+1\}} Y_{i,j} = 1, \quad \forall i \in N \cup \{0\} \tag{4.4}$$

$$\sum_{i \in N \setminus \{j\} \cup \{0\}} Y_{i,j} = 1, \quad \forall j \in N \cup \{|N|+1\} \tag{4.5}$$

Constraints (4.4) and (4.5) ensure that a feasible solution unambiguously defines a production sequence for the jobs. Each job $j \in N$ must have a single immediate predecessor and a single immediate successor. The inclusion of "dummy jobs" 0 and $n^N + 1$ ensures that the first scheduled regular job has predecessor 0 and that the last scheduled regular job has successor $n^N + 1$.

Processing Constraints:

$$C_j \geqslant C_i + p_j Y_{i,j} + M_{i,j}(Y_{i,j} - 1), \quad \forall i, j \in N, i \neq j \tag{4.6}$$

The definition of Constraint (4.6) forces the completion time of an immediate successor j of a job i to be larger or equal to the completion time C_i of job i plus the processing time p_j of job j. The completion times are consequently in line with the production sequence defined by Constraint (4.4) and (4.5) and ensures that at most one job is produced simultaneously. Constraints (4.6) include the "big M" parameter $M_{i,j}$, which is defined for each job pair $i, j \in N, i \neq j$. The parameter guarantees that the completion times of jobs, that are not the immediate successor to a job i, are not restricted by the introduction of the processing constraints. Parameter $M_{i,j}$ is set to value $r_j + p_j - \bar{d}_i$ for each $i, j \in N, i \neq j$. Hence, if $C_i = \bar{d}_i$ holds, then the constraint reduces to $C_j \geqslant r_j + p_j$. Otherwise, if $C_i < \bar{d}_i$ holds, then C_j is lower bounded less tight.

Production Time Window Constraints:

$$C_j \leqslant \bar{d}_j, \quad \forall j \in N \tag{4.7}$$

$$C_j \geqslant r_j + p_j, \quad \forall j \in N \tag{4.8}$$

The production time windows of the jobs are defined by Constraints (4.7) and (4.8), which restrict processing within the interval $[r_j, \bar{d}_j]$. The earliest processing start time of a job $j \in N$ is $r_j + p_j$, and the latest completion time is the deadline $\bar{d}_j$.

Delivery Assignment Constraints:

$$\sum_{g \in G} \sum_{l=1}^{n_g^T} Z_{g,l,j} \cdot a_{g,j} = 1, \quad \forall j \in N \tag{4.9}$$

The Constraint (4.9) ensures that a job j of customer g is assigned to exactly one transport to customer g.

Delivery Time Constraints (1):

$$D_j = \sum_{g \in G} \sum_{l=1}^{n_g^T} t_{g,l} \cdot Z_{g,l,j}, \quad \forall j \in N \tag{4.10}$$

The setting of the binary decision variables $Z_{g,l,j}$ for a job j defines the delivery time D_j with the inclusion of Constraint (4.10). The delivery time D_j is set to the l-th delivery time of customer g if $Z_{g,l,j}$ is set to *one*. Hence, the domain of the delivery time variables is restricted to the delivery departure time parameters by combined application of Constraints (4.9) and (4.10).

Delivery Time Constraints (2):

$$D_j \geqslant C_j, \quad \forall j \in N \tag{4.11}$$

$$D_j \leqslant \bar{d}_j, \quad \forall j \in N \tag{4.12}$$

The delivery departure times of jobs are restricted by Constraints (4.11) and (4.12). The delivery departure may start at the earliest at the time of completion and must take place before the given job deadline.

Transport Activation Constraints:

$$Z'_{g,l} \cdot B_{g,l}^{max} \geqslant \sum_{j \in N} Z_{g,l,j}, \quad \forall g \in G, \forall l \in \{1, 2, \dots, n_g^T\} \tag{4.13}$$

Constraint (4.13) ensures that the binary batching variables $Z'_{g,l}$ are set to *one* if at least one job is assigned to the l-th transport to customer g. The $Z'_{g,l}$ variables are used in the Objective Function (4.3) to trigger the transportation cost for this activated transport.

Holding Time Constraints:

$$H_j \leqslant \bar{d}_j - r_j - p_j, \quad \forall j \in N \tag{4.14}$$

$$H_j \geqslant D_j - C_j, \quad \forall j \in N \tag{4.15}$$

The holding times of the jobs are lower bound by Constraint (4.15) by the difference between the delivery departure time D_j and the completion time C_j of job j. Additionally, the holding times are upper bound by Constraint (4.14) by the difference between the earliest completion time $r_j + p_j$ and the latest delivery departure time $\bar{d}_j$. Note that Constraint (4.14) is not required to ensure a correct model formulation. The variables H_j are used to trigger the job holding costs in the Objective Function (4.3).

Variable Domains:

$$Y_{i,j} \in \{0, 1\}, \quad \forall i, j \in \{0, 1, ..., n^N + 1\} \tag{4.16}$$

$$Z'_{g,l'} \in \{0, 1\}, \quad \forall g \in G, l \in \{1, 2, \dots, n_g^T\} \tag{4.17}$$

$$Z_{g,l,j} \in \{0, 1\}, \quad \forall g \in G, l \in \{1, 2, \dots, n_g^T\}, j \in N \tag{4.18}$$

$$C_j \geqslant 0, \quad \forall j \in \{0, 1, ..., n^N + 1\} \tag{4.19}$$

$$D_j, H_j \geqslant 0, \quad \forall j \in N \tag{4.20}$$

Lastly, Constraints (4.16 - 4.20) define the variable domains. Note that the variables D_j, C_j, and H_j for jobs $j \in N$ do not need to be defined as integers. Introducing integrity constraints for these variables is also not required, as subsequently shown by Lemma 4.3.1 by Bachtenkirch and Bock (2022).

Lemma 4.3.1. *The completion time variables* C_j*, delivery time variables* D_j*, and holding time variables* H_j *for all jobs* $j \in N$ *are integers in an optimal solution of the MILP formulation of SFDDHT.*

Proof. This proof of Lemma 4.3.1 considers the integrality of the delivery time, completion time, and holding time variables, one type after the other.

- The delivery time variables are integers due to the equality constraint of (4.10). According to this constraint, the delivery time D_j of a job j that is assigned to the l-th delivery date of customer g takes the integer value $t_{g,l}$ in any feasible solution.
- The integrality of completion time values in an optimal solution is proven by assuming that an optimal solution S^* exists with at least one fractional C_j value and shows that such a solution S^* does not exist. Consider the largest fractional C_j value in a solution: Let this completion time be known as $C_{j'}$ of job j'.

  At this point, one can distinguish between two cases: Either j' has at least one successor job in the solution, or it has none.

  – In the latter case, it must hold that $C_{j'} \leqslant D_{j'}$ due to Constraint (4.11). If $C_{j'} = D_{j'}$ holds, then $C_{j'}$ must be integral because $D_{j'}$ is integral, as proven initially. This contradicts the assumption that $C_{j'}$ is fractional and cannot be.

    Otherwise, if $C_{j'} < D_{j'}$ holds, then the solution S^* is not optimal because the setting $C_{j'} = D_{j'}$ reduces the objective value. This is true, since the holding time of job j' is determined by the inequality $H_{j'} \geqslant D_{j'} - C_{j'}$ – see Constraint (4.15)) – that is minimized by setting $H_{j'} = D_{j'} - C_{j'}$ and is *zero* for the setting $C_{j'} = D_{j'}$. The holding cost contribution of job j' is consequently also *zero* in the Objective Function (4.3). Therefore, the optimality assumption of S^* does not hold in this case.

  – In the former case, j' has an immediate successor k with integral completion time C_k and integral start time $S_k = C_k - p_k$. Since $C_{j'}$ is fractional, $C_{j'} < S_k$ holds with idle time on the machine between the production of jobs j' and k. At this point, one can distinguish between two cases: Either the delivery time of job j' is later than the start time of successor k, or it is earlier.

    In the first case $D_{j'} \geqslant S_k$, setting $C_{j'} = S_k$ reduces holding time $H_{j'}$ and hence the holding costs of job j' and the objective function value. However, this violates the optimality assumption of S^*.

    In the other case of $D_{j'} < S_k$, consider the following three cases: It holds that either $C_{j'} > D_{j'}$, $C_{j'} = D_{j'}$, or $C_{j'} < D_{j'}$:

    * Case $C_{j'} > D_{j'}$: This case violates Constraint (4.11) and cannot be.

- Case $C_{j'} = D_{j'}$: Since $D_{j'}$ is integral, this contradicts the assumption that $C_{j'}$ is fractional, and it thus cannot be.
- Case $C_{j'} < D_{j'}$: By setting $C_{j'} = D_{j'}$, the objective value can be reduced, as argued before. Hence, this case also cannot be.

Therefore, no case exists for $D_{j'} < S_k$ that is optimal with $C_{j'}$ being fractional.

In conclusion, there is no case in which $C_{j'}$ is fractional and S^* optimal. By extension, this proves that no completion time C_j with $j \in N \cup \{0\}$ is fractional in an optimal solution.

- Lastly, the holding times H_j must be integers. As already stated during this proof, the constraint $H_{j'} \geqslant D_{j'} - C_{j'}$ (4.15) is minimized by setting $H_{j'} = D_{j'} - C_{j'}$, and since the completion times $C_{j'}$ and delivery times $D_{j'}$ must be integral, the holding times H_j must be integral as well in an optimal solution. □

This proof completes the MILP formulation of the problem.

4.4 Computational Complexity

This section determines the complexity status of the SFDDHT by a polynomial reduction from a **strongly** **$\mathcal{NP}$-hard** problem. The following proof as stated in Bachtenkirch and Bock (2022) shows that the SFDDHT generalizes the **strongly** **$\mathcal{NP}$-hard** problem investigated in Yang (2000).

Lemma 4.4.1. *SFDDHT is **strongly** $\mathcal{NP}$**-hard**.*

Proof. The scheduling problem proposed by Yang (2000), named scheduling problem with generalized batch delivery dates and earliness penalties, is described by the three-field-notation of Graham et al. (1979) as $1|GBDD, B = m|\sum w_i E_i$, where E_i is equivalently defined to H_i, that is $E_i = \max\{D_i - C_i, 0\}$. The acronym $GBDD$ indicates so-called generalized batch delivery dates, that model a number of arbitrary (hence, generalized) fixed delivery dates. Furthermore, the expression $B = m$ denotes the number of considered delivery dates. Consequently, an instance of the SFDDHT with a single customer, no release dates, equivalent deadlines (which match the latest delivery date) and no transportation costs, defines a corresponding instance of $1|GBDD, B = m|\sum w_i E_i$. As Yang (2000) shows that $1|GBDD, B = m|\sum w_i E_i$ is **strongly** **$\mathcal{NP}$-hard**, it follows that SFDDHT is also **strongly** **$\mathcal{NP}$-hard**. Additionally, it holds that the task of finding a feasible schedule to the SFDDHT is strongly $\mathcal{NP}$-hard, since the feasibility problem $1|r_j, \bar{d}_j|-$ is **strongly** **$\mathcal{NP}$-hard** (Garey and Johnson, 1979). □

4.5 Conclusion

This chapter introduced the novel scheduling problem, namely, the SFDDHT, which combines several practically relevant problem assumptions into one model. Section 4.1

explained the reasoning and use cases for the formulated model. Section 4.2 and Section 4.3 described a mathematical, unambiguous problem formulation and the notation of the problem. The established MILP formulation enables solving problem instances of the SFDDHT by standard integer programming optimization methods. Since readily available solvers exist for these types of problems, this chapter's contents provide sufficient information to solve the problem instances of the proposed problem in practice. However, since integer programming is $\mathcal{NP}$**-complete** (c.f. Papadimitriou and Steiglitz, 1982, p. 316), such solution methods require an excessive amount of time to solve larger problem instances. Therefore, to solve larger instances in a reasonable amount of time, other solution approaches must be developed for the considered problem. As the SFDDHT is **strongly** $\mathcal{NP}$**-hard** (Section 4.4), no polynomial-time algorithm (unless $\mathcal{P}$ equals $\mathcal{NP}$) exists that efficiently solves the problem. This finding justifies the research of elaborate OR methods for the SFDDHT that compute (optimal or high-quality heuristic) solutions to **strongly** $\mathcal{NP}$**-hard** problems.

Chapter 5

The Developed Branch and Bound Algorithm

The contents presented in this chapter also appear in the work of Bachtenkirch and Bock (2022), titled Finding efficient make-to-order production schedules, which is published in the European Journal of Operational Research. To avoid impairing the reading flow, citations do not reference this article continuously throughout this chapter. However, when introducing a mathematical statement, such as a lemma, proposition, or proof, from the stated paper, the text includes a reference. Moreover, tables, figures, and algorithms that are taken from this paper are referenced as per usual.

This chapter describes the proposed B&B algorithm for the SFDDHT. As already pointed out at the end of the previous chapter, **strongly $\mathcal{NP}$-hard** optimization problems, such as the SFDDHT, can not be solved by a polynomial-time algorithm unless $\mathcal{P} = \mathcal{NP}$ holds. Fortunately, OR offers several methods to tackle such problems. One of these methods is the B&B method, which draws its effectiveness from utilizing problem-specific knowledge. This chapter explores the structure of the SFDDHT that makes applying the B&B method to the problem a reasonable choice. Papadimitriou and Steiglitz (1982, p. 433) describe B&B as a method of "intelligently enumerating all the feasible points of an optimization problem." Methodologically, the approach tries to prove that a solution is optimal by repeatedly partitioning the solution space. In essence, B&B algorithms search the complete solution space for at least one optimal solution. Since an exhaustive enumeration of all feasible solutions in a reasonable time-frame is impossible for larger instances, B&B algorithms apply various methods to reduce the explicitly searched solution space based on mathematical proofs.

This chapter is structured as follows: Section 5.1 describes the B&B concept and its components, to establish a common understanding of the description of the problem-specific components in the subsequent sections. Section 5.2 explains how the proposed algorithm generally partitions the solution space and verifies its applicability in search of an optimal solution for a problem instance. The section also establishes that an optimal schedule must have specific structural properties, limiting the search for an optimal solution to this type of schedule. Thereafter, Section 5.3 proposes several dominance relations and procedures that further reduce the searched solution space. Sections 5.4 to 5.6 then propose methods to limit the partitioning of the solution space to only promising partitions. Section 5.4 establishes results solely based on the initial instance data, while Section 5.5 executes on information gathered from the currently processed subset of the solution space. Section 5.6 combines the previous two sections' results and describes further reductions based on feasibility and optimality properties. Furthermore, Section 5.7 describes *three* lower bounds that estimate the cost of completing the scheduling of jobs of already partially determined schedules. The B&B algorithm for the SFDDHT (SFDDHT-B&B) uses the resulting bound values to guide the search towards promising solution space regions and to cut off the partitioning process for non-optimal regions. In Section 5.8, the interaction of the described components is explained. Lastly, Section 5.9 provides a summary of this chapter.

5.1 Principles of Branch-and-Bound Algorithms

This section briefly explains the principles and core components of B&B algorithms. The explanations of the SFDDHT-B&B build on the contents introduced in this section. The following explanations are based on various works that provide frameworks or generalized descriptions of B&B procedures. Namely, Lawler and Wood (1966), Mitten (1970), Morin and Marsten (1976), Kohler and Steiglitz (1974), Ibaraki (1976), Ibaraki (1977), Ibaraki (1978), Kumar and Kanal (1983), Nau et al. (1984), Crainic (1993), and Morrison et al. (2016) provide generalized technical descriptions of the procedure and its components. This section also summarizes important concepts that appear useful for describing the SFDDHT-B&B.

According to Mitten (1970) and Morrison et al. (2016), an optimization problem can be described as $\mathcal{P} = (\mathcal{X}, f)$ where $\mathcal{X}$ denotes the set of *feasible solutions* (the *search space*), and $f\colon \mathcal{X} \to \mathbb{R}$ is the objective function. An optimal feasible solution to an optimization problem has function value $\sup_{x\in\mathcal{X}} f(x) = f^*(\mathcal{X})$. The objective of an optimization problem is to find the set of *optimal feasible solutions*

$$\mathcal{X}^* = \{x^* \mid x^* \in \mathcal{X} \text{ and } f(x^*) = f^*(\mathcal{X})\}.$$

The descriptions are subsequently restricted to minimization problems such that

$$\min_{x\in\mathcal{X}} f(x) = f^*(\mathcal{X})$$

holds.

A B&B algorithm decomposes the original set of feasible solutions $\mathcal{X}$ into sets of decreasing size (Kumar and Kanal, 1983). The decomposition is realized by iteratively building a search tree[1] $\mathcal{T} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a set of nodes, and $\mathcal{E}$ is a set of arcs. Each node in this tree represents a subset of the search space $\mathcal{X}$, where each subset $\mathcal{X}_i \subseteq \mathcal{X}$ is called a *subproblem* of $\mathcal{P}$. The *root node* $\mathcal{N}_0$ of the search tree $\mathcal{T}$ corresponds to the

[1] A formal description of the tree data structure is, for example, given in Knuth (1997) in sections 2.3 (pp. 308–316) and 2.3.4 (pp 362–364).

original problem $\mathcal{P}$ with search space $\mathcal{X}_0 = \mathcal{X}$. All other nodes in the tree correspond to subproblems $\mathcal{X}_i$ with $i = 1, 2, \ldots, |\mathcal{N}| - 1$. By definition of a tree, each node, except the root node, has a single *parent node*. In the context of subproblem decomposition, a node is *terminal* if the corresponding subproblem is a singleton (i.e., a set with only *one* element). All nodes, except the terminal nodes, of a tree have *one* to *many child nodes*. These connections are represented by a set of arcs $\mathcal{E}$, where $\mathcal{X}_j$ is a child of $\mathcal{X}_i$ if the arc $(\mathcal{X}_i, \mathcal{X}_j)$ is an element of $\mathcal{E}$. Given the arc $(\mathcal{X}_i, \mathcal{X}_j) \in \mathcal{E}$, subproblem $\mathcal{X}_j$ is generated from $\mathcal{X}_i$ by decomposition. The level of a subproblem $l(\mathcal{X}_i)$ is the length of the path from $\mathcal{X}_0$ to $\mathcal{X}_i$, with $l(\mathcal{X}_0) = 0$. Each terminal node $\mathcal{X}_i$ in the tree is a *trivially* solved subproblem (i.e., $\mathcal{X}_i$ is a singleton). The solution to such a terminal node is a feasible solution $x \in \mathcal{X}$ for problem $\mathcal{P}$ (Kumar and Kanal, 1983; Morrison et al., 2016).

The B&B procedure starts by constructing search tree $\mathcal{T}$ with the *unexplored* subproblem $\mathcal{X}_0$ and an initial, incumbent solution $\hat{x} \in \mathcal{X}$. This incumbent solution can be generated by some heuristic beforehand. Alternatively, the procedure may start without an incumbent solution, by initially choosing $\hat{x}$ arbitrarily and setting $f(\hat{x}) = \infty$. Afterward, in each iteration, the algorithm selects and removes a subproblem $\mathcal{X}_i \subseteq \mathcal{X}$ from a list $\mathcal{L}$ of unexplored subproblems. If $\mathcal{X}_i$ is terminal (i.e., $|\mathcal{X}_i| = 1$ with $x \in \mathcal{X}$) and x is a better solution than the current incumbent solution $\hat{x}$ – that is, $f(x) < f(\hat{x})$ – then the investigated solution x becomes the new incumbent solution. Otherwise, if it can be proven that no solution $x \in \mathcal{X}_i$ is better than the current incumbent solution – that is, $\forall x \in \mathcal{X}_i \colon f(x) \geqslant f(\hat{x})$ – then the subproblem $\mathcal{X}_i$ is pruned. If such a proof cannot be conducted, then the subproblem $\mathcal{X}_i$ is decomposed into smaller subproblems $\mathcal{X}_j$ with $j = 1, \ldots, r$, which are inserted into the search tree T as nodes with edges $(\mathcal{X}_i, \mathcal{L}_j)$ $\forall j = 1, \ldots, r$ and into list $\mathcal{L}$. Once the list $\mathcal{L}$ is empty (i.e., no unexplored subproblem remains) the incumbent solution $\hat{x}$ is returned as an optimal solution of P (Morrison et al., 2016). A generic B&B framework from Morrison et al. (2016) is given in Algorithm 5.1. Note that the above description does not specify any of the core components of a B&B, which are: the *Search Strategy*, the *Branching Strategy*, and the *Pruning Rules*. These components are often highly problem-specific and must be carefully chosen for each optimization problem. The following descriptions subsequently only provide a general understanding of these components and present a few options that

Algorithm 5.1 A Branch-and-Bound Algorithm for a Minimization Problem

```
procedure BRANCH-AND-BOUND($\mathcal{X}$,f)
    Set $\mathcal{N} = \{\mathcal{X}\}, \mathcal{E} = \varnothing, L = \{\mathcal{X}\}$ and initialize $\hat{x}$
    while $\mathcal{L} \neq \varnothing$ do
        Select and remove a subproblem $\mathcal{X}_i$ from $\mathcal{L}$
        if $|\mathcal{X}_i| = 1$ (terminal) and $f(x) < f(\hat{x})$ set $\hat{x} = x$ then
        else if $\mathcal{X}_i$ can not be pruned then
            Decompose $\mathcal{X}_i$ into subproblems $\mathcal{X}_j$ with $j = 1, 2, \ldots, r$
            Insert each $\mathcal{X}_j$ into $\mathcal{N}$ and $\mathcal{L}$ and insert edges $(\mathcal{X}_i, \mathcal{X}_j)$ into $\mathcal{E}$
    return $\hat{x}$
```

Note. Adapted from Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102. Copyright 2016 by Elsevier.

are applied in the literature.

5.1.1 Search Strategy

The search strategy defines the order in which nodes are selected from the list of unexplored subproblems $\mathcal{L}$ throughout the procedure. Ibaraki (1976) describes the decision of selecting the next subproblem by considering the result of a heuristic function

$$\mathrm{h}\colon \mathcal{X} \rightarrow \mathbb{R}$$

with $\mathrm{h}(\mathcal{X}_i) \neq \mathrm{h}(\mathcal{X}_j)$ for $\mathcal{X}_i \neq \mathcal{X}_j$ to prevent ties. Different search strategies may significantly impact the required computational time of a B&B procedure. Additionally, the memory necessary to search varies substantially between strategies (Morrison et al., 2016). Sections 5.1.1.1 to 5.1.1.3 describe some prominently applied search strategies and their characteristics. Further information about different variants and hybrids of the strategies can be found in the in-depth discussion by Morrison et al. (2016).

5.1.1.1 Depth-first Search

The depth-first search (DFS) strategy explores the solution space by choosing, in each iteration, the most recently generated subproblem in list $\mathcal{L}$. Synonymous labels for this strategy are *branch from newest active bounding problem* (Lawler and Wood, 1966) or the *last-in-first-out (LIFO) rule* (Kohler and Steiglitz, 1974). This approach can be conducted by implementing the list $\mathcal{L}$ as a stack (Morrison et al., 2016). A stack is a linear list, where insertion, deletion, and access are made at the same end of the list (Knuth, 1997, p. 239). An advantage of this branching strategy is that a minimum amount of computer memory is required to perform the search, since the size of the stack is proportional to the depth of the tree (Lawler and Wood, 1966). An even more memory-efficient way to perform a DFS is to store the path from the root node to the currently evaluated subproblem (Morrison et al., 2016). A downside to this approach is that simple implementations do not utilize bounds and information about the structure of the problem (Morrison et al., 2016). Variants of the DFS approach that attempt to overcome the downsides of the standard variant are iterative, deepening DFSs (Korf, 1985), interleaved DFSs (Meseguer, 1997), or DFSs with complete branching (Scholl and Klein, 1999).

5.1.1.2 Breadth-First Search

In contrast to the DFS strategy, the breadth-first search (BFRS) strategy explores the solution space by choosing, in each iteration, the subproblem $\mathcal{X}_i$ that was added earliest to list $\mathcal{L}$ (Morrison et al., 2016). Kohler and Steiglitz (1974) describe this strategy as the *first-in-first-out (FIFO) rule*. Similar to a DFS, the BFRS strategy can be implemented with the help of a simple data structure, namely, a queue. A queue is a linear list (i.e., a sequence of elements with relative positions defined between elements) where insertion is performed at one end of the list, and access and deletion are performed at the other end of the list (Knuth, 1997, p. 239). This strategy builds the search tree level by level. The approach (in pure form) consequently does not utilize information on complete solutions to prune subproblems, while other pruning rules may be applied (Morrison et al., 2016).

5.1.1.3 Best-First Search

The best-first search (BFS) strategy explores the solution space by choosing, in each iteration, the *best* subproblem $\mathcal{X}_i$. Subproblems in this approach are evaluated by a heuristic function where the subproblem with the minimal value $h(\mathcal{X}_i)$ is selected. The approach explores promising regions of the solution space first, to prune less promising regions afterward. A heap, which is a tree data structure that stores elements and some key, can implement the BFS strategy. In a *min-heap*, the element at the top of the tree is the one with the minimal key value of all stored elements. By reordering elements, the heap property holds after each insertion or extraction. Accordingly, the data structure allows for efficient retrieval of elements in non-decreasing order of their keys (Morrison et al., 2016). Morrison et al. (2016) state that the BFS strategy offers significant advantages over a DFS strategy. A BFS is not tied to branches of a tree and often finds good, complete solutions earlier in the search process.

However, a drawback to the BFS approach is that it may spend a significant amount of time in the middle regions of the search tree. A hybridization of a DFS and a BFS, called *cyclic* or *distributed* BFS, presented by Kao et al. (2009), tries to overcome this problem by storing subproblems in heaps (called contours) organized by tree levels. In each iteration, the locally "best" subproblem from a specified contour is selected for branching. A natural approach to choose a contour in an iteration is to iterate a cyclic list of contours ordered by tree level.

5.1.2 Branching Strategy

Branching refers to the decomposition of a subproblem into smaller subproblems. Branching strategies can be categorized as either binary or non-binary (wide). The following discussion does not consider branching strategies applied to integer programming problems.

5.1.2.1 Binary Branching

Binary branching approaches decompose each subproblem into exactly two mutually exclusive smaller subproblems. These approaches appear to be most useful for optimization problems where elements are selected or rejected as part of a solution (Morrison et al., 2016). A prominent application of binary branching is the B&B approach for the *binary knapsack problem* presented by Kolesar (1967). This problem seeks a value-maximizing selection of indivisible items to fit into a capacitated bin. The presented algorithm branches on a subproblem $\mathcal{X}_i$ by constructing two subproblems, where an item is included in the knapsack in one smaller subproblem and excluded in the other smaller subproblem.

5.1.2.2 Non-binary Branching

Non-binary branching (or wide branching) may deconstruct a subproblem into more than two smaller subproblems (Morrison et al., 2016). For instance, wide branching strategies are appropriate for problems with solutions definable by a permutation of elements. Kohler and Steiglitz (1974) explicitly deals with B&B algorithms for such *permutation problems*. An example of a B&B applied to such a permutation problem is the approach described by Ignall and Schrage (1965) for flow-shop scheduling problems.A flow-shop problem covers the scheduling of jobs on multiple machines, with each job requiring processing only once on each machine. For some special cases, it is sufficient to describe a solution as a permutation of the job indexes. Ignall and Schrage use a branching strategy that decomposes a subproblem by extending the partial permutation by an unused job index to form a partial schedule.

5.1.3 Pruning Rules

Pruning rules exclude regions of the search space from exploration. Without the application of pruning rules, the B&B procedure explores *all* subproblems, regardless of the applied search or branching strategy. Pruning rules are consequently essential to reduce the computational time necessary and lower memory requirements in order to provide an optimal solution. The implementation of pruning rules is highly problem-specific. Two types of pruning rules can generally be distinguished for a minimization problem: lower bounds and dominance relations.

5.1.3.1 Lower Bounds

Most prominently, B&B algorithms for minimization problems use lower bounds to prune subproblems. A lower bound $\mathrm{LB}\colon \mathcal{X}_i \to \mathbb{R}$ computes a lower bound on the objective function value that holds for each solution of a subproblem; that is,

$$\mathrm{LB}(\mathcal{X}_i) \leqslant f(x), \; \forall x \in \mathcal{X}_i.$$

If the lower bound $\mathrm{LB}(\mathcal{X}_i)$ of the current subproblem is not strictly better than the currently known best solution value $f(\hat{x})$ in the search process, then the subproblem $\mathcal{X}_i$ can be pruned. That is, $\mathcal{X}_i$ does not contain a feasible solution that has a lower objective function value than $\hat{x}$ and therefore must not be investigated.

Additionally, the following two important conditions, stated in Ibaraki (1977), generally hold for lower bounds of subproblems in a B&B algorithm:

(a) $\mathrm{LB}(\mathcal{X}_i) \leqslant \mathrm{LB}(\mathcal{X}_j)$, for all $\mathcal{X}_j \subseteq \mathcal{X}_i$, and

(b) $\mathrm{LB}(\mathcal{X}_j) = f(x)$ holds if $\mathcal{X}_j = \{x\}$.

Condition (a) states that decomposed subproblems do not reduce the lower bound value. Since the decomposed subproblem $\mathcal{X}_j$ comprises a subset of solutions of $\mathcal{X}_i$, it immediately follows that $\mathrm{LB}(\mathcal{X}_i)$ is a valid bound for $\mathcal{X}_j$. Therefore, $\mathrm{LB}(\mathcal{X}_i) \leqslant \mathrm{LB}(\mathcal{X}_j)$ must hold. Condition (b) enforces tightness of the lower bound values for singleton subproblems.

Since such a subproblem comprises a single solution, the lower bound $\mathrm{LB}(\mathcal{X}_j)$ can be set to the objective function value of the solution x.

A common approach to construct a lower bound for a subproblem is to apply polynomial optimal solution approaches to relaxed versions of a subproblem (Morrison et al., 2016).

5.1.3.2 Dominance Relations

Given two subproblems $\mathcal{X}_1$ and $\mathcal{X}_2$, a dominance relation $\mathcal{X}_1 D \mathcal{X}_2$ allows $\mathcal{X}_2$ to be pruned. That is, the subproblem $\mathcal{X}_1$ contains a solution $x \in \mathcal{X}$ that is at least as good as all solutions x' of subproblem $\mathcal{X}_2$. In other words,

$$\mathcal{X}_1 D \mathcal{X}_2 \implies \text{there exists a } x \in \mathcal{X}_1 \text{ with } f(x) \leqslant f(x') \text{ for all } x' \in \mathcal{X}_2.$$

If it is proven that an explored subproblem is dominated, then this subproblem can be pruned during the exploration of the search space. Dominance relations can be classified as either memory-based or non-memory-based. Memory-based relations compare subproblems to already explored and stored subproblems, while non-memory-based rules have no such requirement. Instead, non-memory-based rules dynamically generate potentially dominating subproblems to test whether dominance can be established (Morrison et al., 2016).

5.2 Search Space and Solution Properties of the SFDDHT

This section starts with a description of the developed SFDDHT-B&B by defining the explored search space. A solution to a problem instance of the SFDDHT is defined by an allocation of jobs to specific machine times for processing and an allocation of jobs to delivery departure times. Let $\mathcal{X}^o$ be the *search space* (all feasible solutions) of a problem instance of the SFDDHT as defined by the constraints of the MILP formulation in Section 4.3. Note that the feasibility of an allocation of jobs to machine times and delivery departure times is computationally easy to test by, for example, verifying the validity of the constraints provided by the MILP formulation. However, a prior specification of the search space $\mathcal{X}^o$ is not possible without enumerating all *reasonable* solutions. A *reasonable* solution can be thought of as a solution generated by a procedure that does not violate the problem constraints in an obvious manner. Such apparent violations might be the preemptive production of jobs, overlaps in the production of jobs, or the scheduling of jobs outside of the given time windows. No formal definition of what a *reasonable* solution constitutes is given, since it is not crucial for the following discussion.

An important aspect to keep in mind is that the B&B needs to identify infeasible branching decisions as early as possible to skip the explicit evaluation of infeasible regions of the search space. Furthermore, the set of all feasible settings of the completion time variables C_j and delivery departure time variables D_j is large, even for small instances. An approach that decomposes the search space with both variable types consequently does not seem to be viable. Recall from Section 2.2.1 that a schedule defines the allocation of jobs to specific machine times, while a sequence only defines the processing order of jobs on a machine. Many scheduling problems found in the literature offer an inherent structure that is utilizable to search for optimal solutions. For example, for many scheduling problems, optimal schedules are shown to be *non-delay* schedules; that is, jobs are scheduled as early as possible without forcing idle times on the machine(s) (Pinedo, 2016, p. 22). Therefore, in many cases, a sequence can be mapped to a single, best-attainable schedule that comprises this sequence. The SFDDHT offers a similar structure. The following subsections demonstrate that it is sufficient to define solutions in terms of a production sequence on the machine and an allocation of the scheduled jobs to delivery departure times. This branching strategy eliminates the requirement to explore all feasible completion time settings in the SFDDHT-B&B.

5.2.1 Schedule Representation

A representation scheme that equates solutions of the SFDDHT with schedules is subsequently shown. The described schedule definition covers complete and partial schedules. A *partial schedule* is a schedule defined for a proper subset of jobs of set N. Hereafter, a (partial) schedule for an instance of the SFDDHT is denoted by symbol ζ. To distinguish partial and complete schedules, let $J(\zeta) \subseteq N$ be the *scheduled jobs* in a (partial) schedule ζ that comprises $n^J(\zeta)$ jobs, and let $\bar{J}(\zeta) = N \setminus J$ be the *remaining (unscheduled) jobs* of schedule ζ that comprises the remaining $n^{\bar{J}}(\zeta) = n^N - n^J$ jobs. The *processing sequence* of jobs

$$\Pi(\zeta) = (\pi(k), \pi(k+1), ..., \pi(n^N))$$

is a permutation (i.e., an ordered arrangement) of job-indexes $j \in J$, which defines the processing order of jobs in a (partial) schedule ζ. An element $\pi(i)$ of the sequence $\Pi(\zeta)$ defines the index of the job at the i-th position in schedule ζ. For the purposes of the SFDDHT-B&B, it is sufficient to define partial schedules that sequence jobs at the back positions of the schedule. Symbol $k(\zeta)$ consequently denotes the *first determined position* in schedule ζ. Additionally, the *scheduled position* of a job $j \in J$ is denoted by $\pi^{-1}(j)$. The transportation decisions in a (partial) schedule ζ are described by a sequence of *transportation assignments*

$$P(\zeta) = (\rho(k), \rho(k+1), ..., \rho(n^N)).$$

An element $\rho(i) \in P(\zeta)$ defines the delivery departure time ($D_{\pi(i)} = \rho(i)$) for the job scheduled at the i-th position. It holds that $\rho(i) \in T_{g^N(\pi(i))}$. In addition to the sequences Π and P, a specification of the completion times C_j for all jobs $j \in J$ is necessary to fully define a (partial) schedule ζ (Section 5.2.2 discusses the computation of completion times). The holding cost of a (partial) schedule ζ amount to

$$c^H(\zeta) = \sum_{j \in J(\zeta)} (D_j - C_j) \cdot c^H_j.$$

The transportation costs of a (partial) schedule are computed analogously to Equation (4.2). For notational ease, let $J_g(\zeta) = \{j \mid j \in J(\zeta), g^N((j)) = g\}$ be the set of *scheduled jobs of* *customer* $g \in G$ in a partial solution ζ. The transportation costs are as follows:

$$c^{T}(\zeta) = \sum_{g \in G} c_{g}^{T} \cdot \left|\left\{D_{j} \mid j \in J_{g}(\zeta)\right\}\right|.$$

The total *objective function value of a (partial) schedule* ζ is expressed by

$$f(\zeta) = z(\zeta) = c^{H}(\zeta) + c^{T}(\zeta).$$

Note that the forthcoming sections omit the supplement (ζ) from the defined symbols for readability purposes when no ambiguity exists.

5.2.2 Canonical Schedules

This subsection shows that the explicit specification of completion times is unnecessary in the search for optimal solutions. Specifically, it proves that cost-optimal completion times for a set of jobs $J \subseteq N$ can be derived with only a given production sequence Π and transportation assignments P. Consider the following notational extension: The *start time* of a job j in a (partial) schedule ζ is denoted as $S_j = C_j - p_j$ for all jobs $j \in J$. Furthermore, the *start time* of the job at the first determined position k in a schedule ζ is abbreviated as $S^1 = S_{\pi(k)}$. If the schedule ζ is empty, then S^1 is set to the maximal deadline ($\max\{\bar{d}_j \mid j \in N\}$). Procedure 5.2.2.1 by Bachtenkirch and Bock (2022) calculates cost-optimal completion times given production sequence Π and transportation assignments P.

Procedure 5.2.2.1. The completion time $C_{\pi(i)}$ of the job that is scheduled at the i-th position and is assigned to delivery departure time $D_{\pi(i)} \leqslant \bar{d}_{\pi(i)}$ in the (partial) schedule ζ is defined by recursively applying the following expression:

$$C_{\pi(i)} = \min\left\{D_{\pi(i)}, S_{\pi(i+1)}\right\}. \tag{5.1}$$

The computation begins with the last scheduled job with the setting $C_{\pi(n^N)} = D_{\pi(n^N)}$ and continues the computation by applying Equation (5.1) for all $i = \pi(n^N - 1), \pi(n^N - 2), \ldots, \pi(k)$ in order of decreasing positions.

Lemma 5.2.2.1 by Bachtenkirch and Bock (2022) proves the cost optimality of the generated completion times when the resulting schedule is feasible, that is, release date restrictions are not violated.

Lemma 5.2.2.1. *Given jobs* $J \subseteq N$*, processing sequence* Π*, and transportation assignments* P *Procedure 5.2.2.1 computes cost-optimal completion times for all jobs* $j \in J$ *if the setting* (Π, P) *enables a feasible schedule.*

Proof. The validity of Lemma 5.2.2.1 stems from the independence of the transportation costs of a (partial) schedule ζ from the jobs' production sequence and completion times. In particular, a (partial) schedule fixes the delivery departure times and the production sequence of jobs, and it hence defines the resulting transportation costs by the transportation assignment P. The remaining cost contribution of each scheduled job $j \in J$ thus solely depends on the holding costs $(D_j - C_j) \cdot c_j^H$ with $D_j \geqslant C_j$. Procedure 5.2.2.1 sets the completion time C_j to its maximal value $C_j \leqslant D_j \leqslant \bar{d}_j$ that complies with (a) the delivery departure time defined by P and (b) the production sequence Π.

The procedure determines the completion time values C_j in reversed production order, starting with the job positioned last $j = \pi(n^N)$. This job has no successor in the production schedule. The procedure consequently sets the completion time C_j to its maximal value D_j, which results in *zero* holding time and costs for job j in partial schedule ζ. Each earlier scheduled job $j \in J$ with a position $i < n^N$ has an immediate successor $\pi(i+1)$ that upper bounds the feasible completion time of C_j by the start time of this successor ($S_{\pi(i+1)}$). Job j is therefore scheduled with a minimum holding time such that it completes before the immediate successor starts its production and before or at its delivery departure time D_j. □

The purpose of Procedure 5.2.2.1 is to compute cost-optimal completion times for the scheduled jobs J given the information provided by the two sequences Π and P. The procedure does not guarantee the construction of a feasible (partial) schedule, nor does it guarantee that a feasible partial schedule is extendable to a feasible complete schedule.

Verifying the feasibility of the constructed schedule is assessable in each step of the computation: Lemma 5.2.2.1 already enforces feasible delivery time assignments to departure times by requiring $D_j \leqslant \bar{d}_j$ for each job $j \in J$, and a test for deadline violations is hence unnecessary during application of Procedure 5.2.2.1. Therefore, the infeasibility of a schedule may only arise in the form of release date violations. These violations can be checked in each step of the procedure by testing whether $S_{\pi(i)} \geqslant r_{\pi(i)}$ holds for the job scheduled at position i. The feasibility status of a (partial) schedule generated by Procedure 5.2.2.1 is consequently determined alongside the generation of completion times. Definition 5.2.2.2 describes schedules that are generated by Procedure 5.2.2.1.

Definition 5.2.2.2. A (partial) schedule constructed according to Procedure 5.2.2.1 is called a *canonical schedule*. Furthermore, the completion times of jobs in a canonical schedule are referred to as *canonical completion times*.

5.2.3 Canonical Search Space

This subsection demonstrates that due to the application of Lemma 5.2.2.1, the search for an optimal solution can be restricted to a search over all feasible canonical schedules. Let the *canonical search space* $\mathcal{X}^c$ comprise all *feasible* canonical schedules for an instance of the SFDDHT. To prove that it is sufficient to explore only canonical schedules, a procedure is introduced that transforms a feasible schedule into a feasible canonical schedule. Procedure 5.2.3.1 reschedules the jobs of a solution $x^o \in \mathcal{X}^o$ by modifying the completion times of the scheduled jobs.

Procedure 5.2.3.1. Given a feasible solution $x^o \in \mathcal{X}^o$ with production sequence Π^o and transportation assignments P^o, the transformation function $f^{oc}\colon \mathcal{X}^o \to \mathcal{X}^c$ constructs solution $x^c \in \mathcal{X}^c$ from the production sequence and transportation assignment of solution $x^o \in \mathcal{X}^o$. Specifically, the transformation is carried out by first setting $\Pi^c = \Pi^o$ and

$P^c = P^o$. The completion times for all jobs $j \in J^c = J^o$ from position n^N to position k are subsequently computed by Procedure 5.2.2.1, and the resulting schedule is a canonical schedule.

Lemma 5.2.3.1. *Procedure 5.2.3.1 transforms each feasible solution $x^o \in \mathcal{X}^o$ into a* corresponding *feasible canonical solution $x^c \in \mathcal{X}^c$ with equal or lower objective function value; that is,* $f(x^c) \leqslant f(x^o)$.

Proof. The solutions x^o and x^c only differ by the completion times of jobs. Since a canonical schedule (Lemma 5.2.2.1) is cost optimal in terms of completion times, it holds that $f(x^c) \leqslant f(x^o)$. □

Accordingly, each feasible solution $x^o \in \mathcal{X}^o$ is transformable into a corresponding feasible solution $x^c \in \mathcal{X}^c$ with equal or lower cost. Lemma 5.2.3.2 formalizes the relation between the search spaces $\mathcal{X}^o$ and $\mathcal{X}^c$.

Lemma 5.2.3.2. *The canonical search space $\mathcal{X}^c$ is a subset of the search space of all feasible solutions $\mathcal{X}^o$.*

Proof. The search space $\mathcal{X}^o$ comprises all feasible combinations of Π, P, and C_j values for $j \in N$. The search space $\mathcal{X}^c$ comprises all feasible combinations of Π, P with one specific feasible setting of the C_j values for $j \in N$. Hence, $\mathcal{X}^c$ is a subset of $\mathcal{X}^o$. □

In conclusion, it is sufficient to explore the search space $\mathcal{X}^c$ instead of the search space $\mathcal{X}^o$. Therefore, the proposed B&B algorithm explores search space $\mathcal{X}^c$. Hereafter, if not stated differently, the term "schedule" implicitly refers to a *canonical schedule* (simply denoted by symbol ζ) in the context of the SFDDHT-B&B.

5.2.4 Decomposition of the Canonical Search Space

The developed B&B algorithm uses a non binary (wide) branching scheme (see Section 5.1.2) that decomposes a subproblem, which implicitly defines a canonical schedule ζ, by extending this schedule ζ in each subsequent subproblem. The extension is carried out by additionally scheduling a job $j \in \bar{J}$. Specifically, a *backward-oriented branching scheme* is used, which builds partial schedules from the last scheduling position to the first.

The initial subproblem $\mathcal{X}^c_0$ is equivalent to the original problem with an empty schedule ζ; that is, $J = \varnothing$ and $\bar{J} = N$. Each subsequent smaller subproblem defines a nonempty schedule with $J \neq \varnothing$ and $\bar{J} = N \setminus J$. The scheduled jobs $j \in J$ occupy positions k to n^N, while leaving positions 1 to $k-1$ vacant. Each selected, non-dominated subproblem $\mathcal{X}^c_h$, with first defined position k_h, decomposes into subproblems $\mathcal{X}^c_i$ with $i = h+r, h+r+1, \dots, h+r+s-1$, where $h+r$ is the *smallest unused index for a search tree node*, and s is the *number of smaller subproblems* that are generated. The subproblem $\mathcal{X}^c_h$ is decomposed by

(a) assigning a job $j \in \bar{J}$ to the highest indexed, vacant position $(k_h - 1)$ and

(b) assigning this job j to a delivery departure time $t_l \in T_{g^N(j)}$.

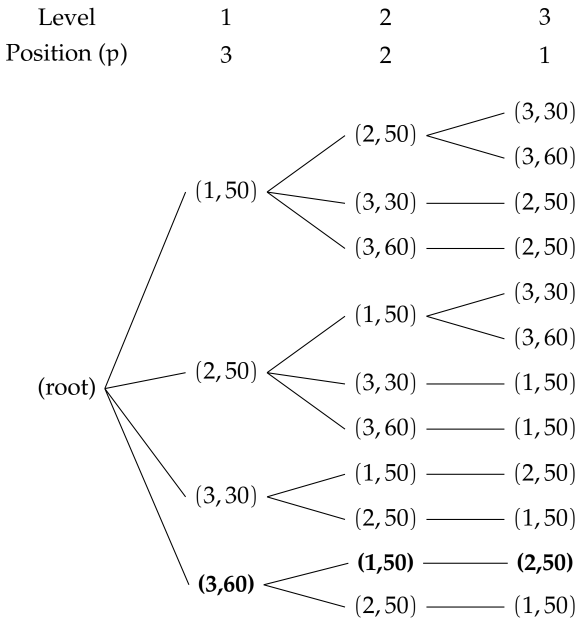
In the implementation of the approach, a node $\mathcal{N}_i^c$, which represents subproblem $\mathcal{X}_i^c$, is stored with the following information:

- A job index $j \in N$,
- the customer-specific delivery departure time index $l \in \left\{1, 2, \ldots, n^T_{g^N(j)}\right\}$, and
- the *memory location* of its parent node $\mathcal{X}_h$. This information defines arc $(\mathcal{X}_h^c, \mathcal{X}_i^c) \in \mathcal{E}$ in search tree $\mathcal{T}$.

The level of the node in the search tree defines the position in the schedule to which job j is assigned; that is, $\pi^{-1}(j) = n^N - l(\mathcal{X}_i^c) + 1$. A corresponding schedule ζ_i^c for a subproblem $\mathcal{X}_i^c$ is constructed by gathering the node information on the path from the search node to the root node of the tree. The visited nodes on this path define the jobs scheduled on positions k to n^N and their assignments to delivery departure times. The sequences Π and P are hence iteratively constructed by following the path in the search tree. To fully define the considered schedule ζ_i^c, Procedure 5.2.2.1 computes the completion times of the specified canonical schedule. Figure 5.1 illustrates the implemented branching scheme.

Figure 5.1

Decomposition of the Search Space for the SFDDHT Problem



Note. Each terminal node at tree level n^N specifies a complete feasible schedule. The displayed tree is generated for an instance with the following data: $N = \{1, 2, 3\}$, $G = \{1, 2\}$, $N_1 = \{1, 2\}$, $N_2 = \{3\}$, $T_1 = \{50\}$, and $T_2 = \{30, 60\}$. Additionally, all schedules are assumed to be feasible. Each node (except for the root node) in the illustration is displayed as a decision pair$(\pi(p), \rho(p))$ that defines – together with all nodes on the path to the root node – a subproblem. For example, the emphasized nodes in the illustration lead to a complete schedule with production sequence $\Pi = (2, 1, 3)$ and transportation assignments $P = (50, 50, 60)$.

5.2.5 Structural Analysis of Canonical Schedules

The overarching goal of the B&B method is to reduce the explicitly searched solution space to find an optimal solution. This subsection analyzes the structure of canonical schedules to distinguish potentially optimal from suboptimal decisions. Since each additional identification of the optimal structure of an optimal solution to the SFDDHT (and its application during the implemented B&B procedure) potentially reduces the number of created search nodes through avoided branches and, therefore, the time spent evaluating these nodes. This section commences by first describing some essential properties of canonical schedules. Afterward, it presents additional constructs to structure canonical schedules.

Property 5.2.5.1 by Bachtenkirch and Bock (2022) explicitly states the feasibility requirement for each feasible canonical schedule.

Property 5.2.5.1. A feasible schedule ζ processes the scheduled jobs $j \in J$ such that $r_j + p_j \leqslant C_j \leqslant D_j \leqslant \bar{d}_j$ holds.

Proof. Property 5.2.5.1 directly follows from the problem definition. □

The following two properties (Properties 5.2.5.2 and 5.2.5.3) by Bachtenkirch and Bock (2022) identify canonical schedules with suboptimal transportation decisions. Both properties are formulated as dominance relations between two schedules ζ and ζ'. Symbol $(')$ denotes variables associated with schedule ζ'.

Property 5.2.5.2 identifies suboptimal transportation decisions for a scheduled job $j \in J$, under consideration of the transportation decisions of jobs for the same customer as job j scheduled at succeeding positions. Specifically, for each job j with successor k, with $g^N(j) = g^N(k)$, it holds that $D_j \leqslant D_k$ because the opposite decision, $D_j > D_k$, causes unnecessary holding costs for job j.

Property 5.2.5.2. Let the jobs j and k, sequenced in this order, with identical customer ($g^N(j) = g^N(k)$), be feasibly processed in a (partial) schedule ζ. It consequently holds that $C_j \leqslant D_j \leqslant \bar{d}_j$ and $C_k \leqslant D_k \leqslant \bar{d}_k$. If the completion times and delivery departure times for both jobs are chosen such that $C_j < C_k$ and $D_j > D_k$ simultaneously apply, the schedule ζ is dominated by a schedule ζ' that coincides with ζ except for setting $D'_j = D_k$.

Proof. Property 5.2.5.2 proposes setting the delivery departure time of job j to time D_k instead of time D_j. This alternative is a feasible choice due to the assumption that job j is feasibly processed in schedule ζ and that $\bar{d}_j \geqslant D_j > D_k$ therefore holds. Schedule ζ' is thus feasible. The choice of delivering job j at time D_k instead of time D_j positively influences the objective function value by reducing the holding cost contribution of job j. Specifically, schedule ζ' attains the objective value

$$z(\zeta') \leqslant z(\zeta) - c^H_j \cdot (D_j - D'_j). \tag{5.2}$$

Equation (5.2) is specified as an inequality because setting $D'_j = D_k$ potentially reduces transportation costs if job j is the sole job transported at delivery departure time D_j in schedule ζ. Since schedules ζ and ζ' process identical jobs ($J = J'$) schedule ζ' dominates schedule ζ. □

Property 5.2.5.3 identifies suboptimal transportation assignments for a scheduled job $j \in J$. Some feasible schedules assign jobs to needlessly distant delivery departure times, even though transports closer to the completion time of the job exist in a solution. An assignment to an earlier delivery departure time consequently reduces the holding costs.

Property 5.2.5.3. Let job j be a job assigned to delivery departure time D_j in a (partial) schedule ζ. Additionally, assume that there exists a delivery departure time $t_\tau \in T_{g^N(j)}$, which is currently not in use by any scheduled job that lies between the completion time of job j and the currently assigned pickup time D_j. That is, it holds for delivery departure time t_τ that $C_j \leqslant t_\tau < D_j$. If $c^H_j \cdot (D_j - t_\tau) > c^T_{g^N(j)}$ holds, then schedule ζ is dominated by a schedule ζ' that is identical to schedule ζ, except that job j is scheduled with delivery departure time $D'_j = t_\tau$ instead of D_j.

Proof. Modifying the assignment of job j from delivery departure time D_j to $D'_j = t_\tau$ leads to a reduction in cost by amount $c^H_j \cdot (D_j - t_\tau) - c^T_{g^N(j)}$. Hence, the decision to use delivery departure time t_τ instead of time D_j is favorable and without any negative effects on other parts of the solution. □

The results of Properties 5.2.5.1 to 5.2.5.3 supplement the definition of canonical schedules. A (partial) schedule for which these propositions do not hold can be discarded in the search for an optimal schedule.

By using the above results, two constructs are introduced that provide additional structure for the scheduled jobs. The following definitions (Definitions 5.2.5.4 and 5.2.5.5) by Bachtenkirch and Bock (2022) introduce *delivery batches* Δ and *production blocks* Γ that each define a collection of subsets of the scheduled jobs J of a (partial) schedule ζ.

Definition 5.2.5.4. Let $\Delta = \{\Delta_1, \Delta_2, ..., \Delta_{n^T}\}$ denote the set of (possibly empty) **delivery batches** in a (partial) schedule ζ. Therefore, for each delivery departure date $l \in I^T$, there is an associated delivery batch Δ_l that defines the set of jobs of customer $g^T(l)$ that are jointly transported with the l-th transport at delivery departure time t_l; in other words,

$$\forall l \in I^T \colon \Delta_l = \left\{ j \mid j \in J_{g^T(l)}, D_j = t_l \right\}.$$

Each delivery batch Δ_l comprises jobs $\delta_l(j)$ with $j = 1, 2, ..., n^\Delta_l$ that are indexed in order of their scheduled position such that $\pi^{-1}(\delta_l(1)) < \pi^{-1}(\delta_l(2)) < ... < \pi^{-1}(\delta_l(n^\Delta_l))$ holds. All jobs $j \in \Delta_l$ in the l-th delivery batch have an identical delivery departure time $D_j = t_l$.

With Property 5.2.5.2 in mind Definition 5.2.5.4 leads to Observation 1:

Observation 1. Consider a (partial) schedule ζ. For two jobs j and k of the same customer $g^N(j) = g^N(k)$, with j scheduled before k and delivery batches $j \in \Delta_l$ and $k \in \Delta_m$, it holds that $l \leqslant m$. That is, a job with a lower indexed position in the schedule is always part of a lower or an equally indexed delivery batch.

Definition 5.2.5.4 preserves the ordering of jobs in the schedule. This enables a job $j \in J$ to be defined as either a *terminal* job in a delivery batch – that is, the job with the highest position (i.e., the latest scheduled job) in a delivery batch – or a *non-terminal* job in a delivery batch. The *set of all terminal jobs* in a schedule ζ is

$$J^t = \left\{ j \mid j \in J \wedge \exists l \in I^T : j = \delta_l(n_l^\Delta) \right\}. \tag{5.3}$$

Let $J^n = J \setminus J^t$ denote the *set of all non-terminally scheduled jobs*. Additionally, j_j^t denotes the *earliest terminal successor of a non-terminal job* $j \in J^n$; that is,

$$\forall j \in J^n : j_j^t = \min \left\{ k \mid k \in J^t : \pi^{-1}(k) > \pi^{-1}(j) \right\}. \tag{5.4}$$

Note that the term j_j^t is properly defined, since the last scheduled job is always terminal. Equations (5.3) and (5.4) enable production blocks to be defined:

Definition 5.2.5.5. Let $\Gamma = \{\Gamma(1), \Gamma(2), ..., \Gamma(n^T)\}$ denote the set of (possibly empty) **production blocks** in a schedule ζ. A production block $\Gamma_l \in \Gamma$ defines the consecutively sequenced set of jobs that comprises the terminal job $\delta_l(n_l^\Delta)$ and the immediate *chain* of non-terminal predecessors in the schedule ζ. Specifically, this chain comprises all non-terminal predecessors $i \in J^n$ for which $j_i^t = \delta_l(n_l^\Delta)$ holds. Production blocks are formally defined as

$$\forall l \in I^T : \Gamma_l = \begin{cases} \{\delta_l(n_l^\Delta)\} \cup \{j \mid j \in J^n, j_j^t = \delta_l(n_l^\Delta)\} & \text{if } \Delta_l \neq \varnothing \\ \varnothing & \text{otherwise.} \end{cases} \tag{5.5}$$

Similar to the definition of a delivery batch by Definition 5.2.5.4, in a production block, jobs are indexed by their scheduled position: Γ_l comprises jobs $\gamma_l(j)$ with $j = 1, 2, ..., n_l^\Gamma$ and $\pi^{-1}(\gamma_l(1)) < \pi^{-1}(\gamma_l(2)) < ... < \pi^{-1}(\gamma_l(n_l^\Gamma))$.

Lemma 5.2.5.1 by Bachtenkirch and Bock (2022) describes an important property of production blocks regarding the delivery departure times of jobs within the same production block:

Lemma 5.2.5.1. *For each production block $\Gamma_l \in \Gamma$, as defined by Definition 5.2.5.5, the following property is valid in a (partial) schedule ζ: Each job $j \in \Gamma_l$ has a delivery departure time $D_j \geqslant C_{\gamma_l(n_l^\Gamma)}$. That is, all jobs of a production block Γ_l have a delivery departure time that is not earlier than the completion time of all jobs of the production block Γ_l.*

Proof. Assume that there exists a job $j = \gamma_l(m)$ of production block Γ_l with delivery departure time $D_j < C_{\gamma_l(n_l^\Gamma)}$. Job j is consequently assigned to an earlier delivery departure time than the last job of production block Γ_l. Therefore, job j is not the last scheduled job of this production block and is thus a non-terminal job. Hence, there exists another job k on one of the positions $m + 1, m + 2, ..., n_l^\Gamma - 1$ of production block Γ_l with delivery departure time $D_k = D_j$ that must be a terminal job. However, this scenario contradicts Definition 5.2.5.5 that defines a production block as a set of non-terminal jobs and only one terminal job that is scheduled at position n_l^Γ of the l-th production block. Therefore, one concludes that $D_j \geqslant C_{\gamma_l(n_l^\Gamma)}$. □

An essential property of nonempty production blocks is that jobs in the same production block are produced consecutively, without idle time between the processing of jobs. This property can be inferred from Lemma 5.2.5.1. Each job in a production block Π_l is delivered at some point after the completion of the last job $\gamma_l(n_l^\Gamma)$ in this production block. Therefore, the holding cost $H_{\gamma_l(i)} = (D_{\gamma_l(i)} - C_{\gamma_l(i)})) \cdot c^H_{\gamma_l(i)}$ is minimized by setting $C_{\gamma_l(i)}$ to its maximal value. This is guaranteed by the procedure of Lemma 5.2.2.1. Thus, idle time in a canonical schedule may only exist between two production blocks or at the beginning of the schedule.

To illustrate the process of building canonical schedules and the structuring of such schedules, an extensive example is presented below. Example 5.2.5.6 is also mentioned by Bachtenkirch and Bock (2022) in a condensed format; the below presented version comprises detailed calculations and an illustration of the instance.

Example 5.2.5.6. There are $n^N = 6$ jobs with $n^G = 2$ customers. Jobs $1 - 3$ belong to customer 1, while jobs 4-6 belong to customer 2. The complete instance data for this example is stated in the following table.

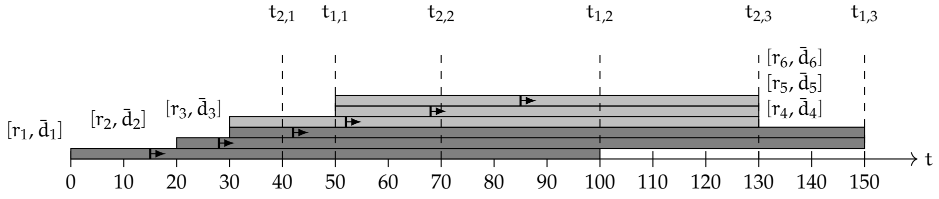
Instance data of the SFDDHT

Customer (g)	Jobs ($g^N(j) = g$)					Transportation cost (c^T_g)	Delivery departure times (T_g)
	j	p_j	r_j	$\bar{d}_j$	c^H_j		
1	1	15	0	100	2	100	50, 100, 150
	2	8	20	150	6		
	3	12	30	150	5		
	j	p_j	r_j	$\bar{d}_j$	c^H_j		
2	4	22	30	130	9	80	40, 70, 130
	5	18	50	130	7		
	6	35	50	130	5		

The time windows for the jobs are illustrated in Figure 5.2.

Figure 5.2

Graphical Instance Representation for Example 5.2.5.6



Note. Each horizontal bar represents the time window of a job j with inner markings to indicate earliest completion times.

The example schedule ζ is described by the two sequences: the production sequence $\Pi = \{1, 4, 5, 2, 3, 6\}$ and the transportation assignments $P = \{100, 70, 70, 100, 100, 130\}$. Therefore, it immediately holds that $D_1 = D_2 = D_3 = 100$, $D_4 = D_5 = 70$, and $D_6 = 130$. The completion times are derived by Procedure 5.2.2.1. To clarify the computations below, the formulas of this procedure are reprinted:

$$C_{\pi(n^N)} = D_{\pi(n^N)} \text{ and}$$

$$C_{\pi(i)} = \min\{D_{\pi(i)}, S_{\pi(i+1)}\} \text{ for each } i = \pi(n^N - 1), \pi(n^N - 2), ..., \pi(k).$$

The computation is carried out as follows:

Step 1: $i = n^N = 6, \pi(i) = \pi(6) = 6$.
$C_6 = D_6 \Leftrightarrow C_6 = 130 \Rightarrow S_6 = C_6 - p_6 = 130 - 35 = 95$.

Step 2: $i = n^N = 5, \pi(i) = \pi(5) = 3$.
$C_3 = \min\{D_3, S_6\} = \min\{100, 95\} = 95 \Rightarrow S_3 = C_3 - p_3 = 95 - 12 = 83$.

Step 3: $i = n^N = 4, \pi(i) = \pi(4) = 2$.
$C_2 = \min\{D_2, S_3\} = \min\{100, 83\} = 83 \Rightarrow S_2 = C_2 - p_2 = 83 - 8 = 75$.

Step 4: $i = n^N = 3, \pi(i) = \pi(3) = 5$.
$C_5 = \min\{D_5, S_2\} = \min\{70, 75\} = 70 \Rightarrow S_5 = C_5 - p_5 = 70 - 18 = 52$.

Step 5: $i = n^N = 2, \pi(i) = \pi(2) = 4$.
$C_4 = \min\{D_4, S_5\} = \min\{70, 52\} = 52 \Rightarrow S_4 = C_4 - p_4 = 52 - 22 = 30$.

Step 6: $i = n^N = 1, \pi(i) = \pi(1) = 1$.
$C_1 = \min\{D_1, S_4\} = \min\{100, 30\} = 30 \Rightarrow S_1 = C_1 - p_1 = 30 - 15 = 15$.

In conclusion, the completion times amount to

$$C_1 = 30, C_2 = 83, C_3 = 95, C_4 = 58, C_5 = 70, \text{ and } C_6 = 130.$$

The available delivery departure times t_l with $l \in I^T$ are indexed by non-decreasing time value, such that: $t_1 = 40$, $t_2 = 50$, $t_3 = 70$, $t_4 = 100$, $t_5 = 130$, and $t_6 = 150$. Since jobs 1-3 are assigned to transport 4, jobs 4 and 5 are assigned to transport 3, and job 6 is assigned to transport 5. The set of delivery batches is

$$\Delta = \{\varnothing, \varnothing, \{4, 5\}, \{1, 2, 3\}, \{6\}, \varnothing\}.$$

The terminal jobs are $J^t = \{5, 3, 6\}$, and the terminal successors of the non-terminal jobs are $j_1^t = j_4^t = 5$ and $j_2^t = 3$. Therefore, the resulting production blocks are

$$\Gamma = \{\varnothing, \varnothing, \{1, 4, 5\}, \{2, 3\}, \{6\}, \varnothing\}.$$

The schedule is also visualized in Figure 5.3. In total, there are *three* transports, *one* for customer 1 at time $t_{1,2} = 100$ and *two* for customer 2 at times $t_{2,2} = 70$ and $t_{2,3} = 130$. Hence, the transportation costs amount to

$$z^T(\zeta) = 1 \cdot c_1^T + 2 \cdot c_2^T = 1 \cdot 100 + 2 \cdot 80 = 260.$$

Jobs 5 and 6 complete at their respective departure times and do not contribute holding costs to the total objective value. Job 1 is stored for $H_1 = D_1 - C_1 = 100 - 30 = 70$ time unitss (TU) and contributes holding cost $H_1 \cdot c_1^T = 70 \cdot 2 = 140$. Job 2 is held for

$H_2 = D_2 - C_2 = 100 - 83 = 17$ TUs and contributes holding cost $H_2 \cdot c_2^T = 17 \cdot 6 = 102$. Next, job 3 is held for $H_3 = D_3 - C_3 = 100 - 95 = 5$ TUs and contributes holding cost $H_3 \cdot c_3^T = 5 \cdot 5 = 25$. Finally, job 4 is held for $H_4 = D_4 - C_4 = 70 - 52 = 18$ TUs and contributes holding cost $H_4 \cdot c_4^T = 18 \cdot 9 = 162$. The total holding cost amounts to

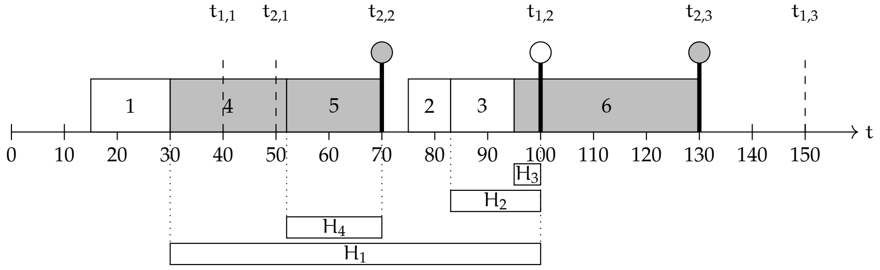
$$z^H(\zeta) = 140 + 102 + 25 + 162 = 429.$$

This results in total cost

$$z(\zeta) = z^T(\zeta) + z^H(\zeta) = 260 + 429 = 689.$$

Figure 5.3

Graphical Representation of the Exemplary Schedule of Example 5.2.5.6.



Note. Deliveries to customer 1 are marked by ○ and deliveries to customer 2 are marked by ●. Additionally, inventory holding times H_j are given below the axis. Adapted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

5.3 Dominance Relations and Procedures

This section describes dominance relations (see Section 5.1), which are the basis for the implementation of *dominance procedures* that try to reduce the number of explicitly explored subproblems during the enumeration process. The proposed dominance relations allow to fathom (finish) a considered (partial) schedule ζ (and all schedules constructible from ζ), if the existence of a dominating (partial) schedule ζ' is proven.

This section describes *four* dominance procedures. The first *three* are non-memory-based dominance procedures. Hence, the dominating schedule ζ' does not need to be found during the search but is constructed specifically to prove dominance over schedule ζ. As a prerequisite for the description of the non-memory-based dominance procedures, Sections 5.3.1 and 5.3.2 describe dominating sequence permutations for the SFDDHT and their efficient evaluation. These results play a major role in the application of the subsequently described dominance procedures, namely, *permutation dominance* (Section 5.3.3), *production block dominance* (Section 5.3.4), and *delivery batch dominance* (Section 5.3.5).

Figure 5.4

Flow Diagram for each Non-Memory-Based Dominance Procedure

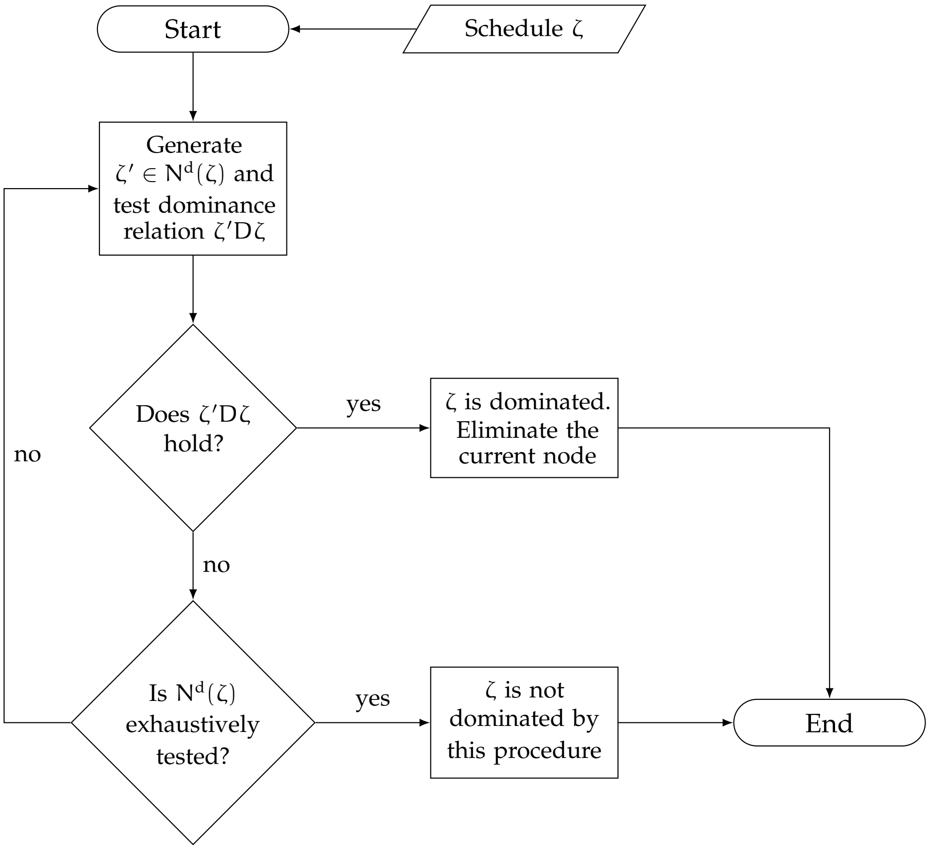

Start
Schedule ζ
Generate $\zeta' \in N^d(\zeta)$ and test dominance relation $\zeta' D \zeta$
Does $\zeta' D \zeta$ hold?
yes
ζ is dominated. Eliminate the current node
no
no
Is $N^d(\zeta)$ exhaustively tested?
yes
ζ is not dominated by this procedure
End

Figure 5.4 illustrates the program flow for the non-memory-based dominance procedures. Each procedure receives a schedule ζ as its input and tests a set of alternative schedules (or neighborhood) $N^d(\zeta)$, where $N^d(\zeta)$ is defined differently by each of the dominance procedures. The schedule alternatives depend on the characteristics of the input schedule ζ, and in each step, the iterative process tests whether one of the alternative schedules dominates the currently explored schedule ζ. Each dominance procedure terminates when establishing dominance over ζ or when no alternative schedule is left to test: Either, one alternative schedule $\zeta' \in N^d(\zeta)$ dominates ζ, or no alternative schedule is constructible, that dominates ζ. In the first case, the B&B discards the current search node that defines schedule ζ. In the second case, the remaining dominance procedures have the opportunity to discard the explored node.

The *fourth* dominance procedure (Section 5.3.6) is a memory-based procedure, implemented with the help of a dominance table that stores the required information of schedules found during the search process to carry out dominance relations testing.

5.3.1 Dominating Permutations

This subsection postulates the existence of optimal production sequences given a fixed transportation assignment of jobs. As a prerequisite, the following paragraphs provide an analysis of the situation for the remaining jobs $\bar{J}$ of a particular schedule ζ. In a (partial) schedule ζ, the processing of the jobs J is determined by Procedure 5.2.2.1, and the decisions are consequently fixed for schedule ζ and all schedules derived from ζ. The start time of the first scheduled job at position k is, as previously introduced, denoted as S^1. Therefore, the processing of jobs $\bar{J}$ is restricted by this time point. That is, all remaining jobs $\bar{J}$ must not complete later than S^1 to feasibly extend schedule ζ towards a complete schedule. Definition 5.3.1.1 additionally considers the deadlines for the remaining jobs:

Definition 5.3.1.1. The *maximal completion time of any unscheduled job* $j \in \bar{J}$, given partial schedule ζ, is

$$\bar{C}^{\max} = \min \left\{ \max_{j \in \bar{J}} \left\{ \bar{d}_j \right\}, S^1 \right\}.$$

The value $\bar{C}^{\max}$ can be seen as an indicator of the flexibility to schedule the remaining jobs $\bar{J}$ given schedule ζ. It follows that any change to the processing of the already scheduled jobs does not negatively impact the flexibility of the remaining jobs, unless $\bar{C}^{\max}$ decreases by such modification. In contrast, given a schedule ζ, a different permutation for the scheduled jobs J may exist, which results in a higher value S^1 and therefore in a potentially higher value for $\bar{C}^{\max}$ (i.e., an increased *flexibility* to schedule the remaining jobs $\bar{J}$). This idea is formalized by Bachtenkirch and Bock (2022) by Lemma 5.3.1.1, which states the existence of locally optimal production sequences. Informally, this lemma compares two feasible schedules with an identical set of jobs scheduled and identical transportation decisions for the scheduled jobs. Both schedules differ only in the sequencing of jobs and hence in the derived canonical completion times. A feasible schedule is dominated if another feasible schedule exists that schedules the jobs in a different order that attains a lower holding cost and is not less flexible than the dominated schedule.

Lemma 5.3.1.1. *Consider two feasible (partial) schedules ζ and ζ' with an identical set of scheduled jobs $J = J'$ and identical delivery departure times $D_j = D'_j$ for each job $j \in J (= J')$ but differing* *production sequences $\Pi \neq \Pi'$. If the alternative schedule ζ' is not less flexible $\bar{C}'^{\max} \geqslant \bar{C}^{\max}$ and attains a lower total cost $z(\zeta') < z(\zeta)$, then schedule ζ is dominated by the alternative schedule ζ'.*

Proof. Lemma 5.3.1.1 compares two non-identical schedules ζ and ζ' with identical scheduled jobs $J = J'$ and identical delivery departure times. As the scheduled jobs are identical, the same holds true for the unscheduled jobs; that is, $\bar{J} = \bar{J}'$. As $\bar{C}'^{\max} \geqslant \bar{C}^{\max}$ holds any extension of schedule ζ towards a complete schedule ζ_c (by scheduling the jobs $\bar{J}'$) can also be executed identically for schedule ζ' towards a complete schedule ζ'_c. That is, the setting of the completion times $C_j = C'_j$ and delivery departure time settings $D_j = D'_j$ for the remaining jobs $\bar{J} = \bar{J}'$ is feasible for the alternative schedule ζ'_c if these decisions are feasible for ζ_c.

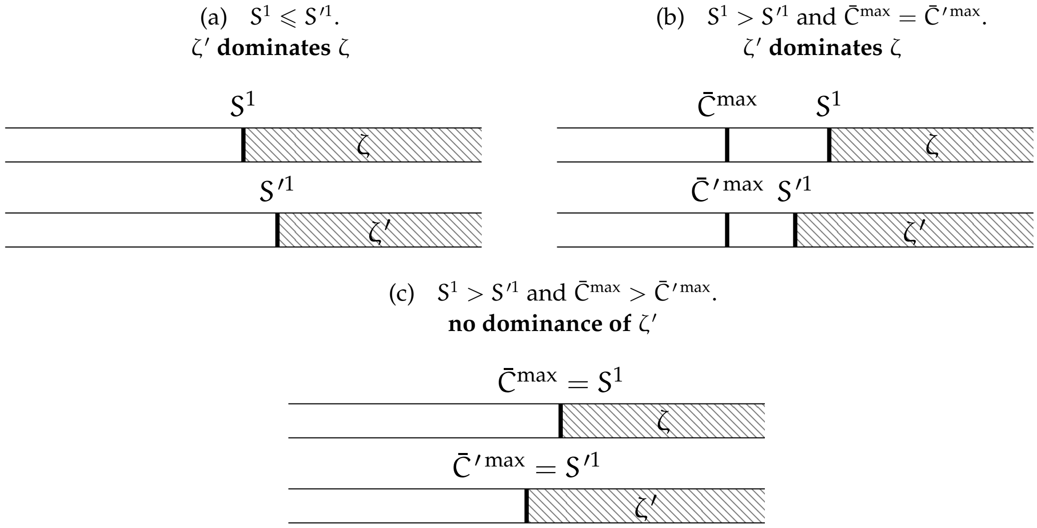
Moreover, Lemma 5.3.1.1 requires $z(\zeta') < z(\zeta)$. The extended schedules ζ_c and ζ'_c have identical decisions concerning the completion and delivery times of the remaining jobs $\bar{J} = \bar{J}'$ of the original schedules ζ and ζ'. Therefore, the transportation cost and holding cost contributions of these jobs are identical in both schedules. That is, the holding costs $\sum_{j \in \bar{J}} (D_j - C_j) \cdot c^H_j$ equal the holding costs $\sum_{j \in \bar{J}'} (D'_j - C'_j) \cdot c^H_j$, and the transportation costs $\sum_{g \in G} c^T_g \cdot \left|\{D_j \mid j \in \bar{J}_g\}\right|$ equal the transportation costs $\sum_{g \in G} c^T_g \cdot \left|\{D_j \mid j \in \bar{J}'_g\}\right|$. The total costs of both extended schedules are consequently increased by the same amount compared to the original schedules ζ and ζ'. However, due to $z(\zeta') < z(\zeta)$, it holds that $z(\zeta'_c) < z(\zeta_c)$, and therefore schedule ζ is dominated by schedule ζ' and can be excluded from the ongoing search process. □

Figure 5.5 illustrates the comparison of Lemma 5.3.1.1 and depicts the alternative outcomes of comparing two schedules ζ and ζ', under the conditions laid out by Lemma 5.3.1.1. All three cases imply that $z(\zeta) < z(\zeta')$ holds, because otherwise, dominance can not be established. The two cases (a) and (b) show the outcomes with both schedules providing identical flexibility for the remaining jobs $j \in \bar{J}$. In the case of alternative (c), the schedule ζ' provides a better sequence in terms of cost but restricts the processing capabilities for the remaining jobs. Hence, case (c) establishes no dominance.

Lemma 5.3.1.1 provides the basis to identify any (partial) production schedule as either *locally optimal* or *locally suboptimal*. That is, either the currently investigated schedule ζ minimizes the holding costs and preserves the flexibility for the unscheduled jobs, or, a different production sequence exists that is strictly better in terms of cost and not worse in terms of flexibility. As previously explained, the feasibility problem of scheduling jobs with hard time windows is **strongly $\mathcal{NP}$-hard** and can not be solved efficiently. Therefore, there is little hope (unless $\mathcal{P} = \mathcal{NP}$) of providing a polynomial algorithm that finds a locally optimal schedule. In fact, for a schedule with n^J jobs, there are $n^J! - 1$ alternative schedules, and an exhaustive test of all permutation sequences must consequently be ruled out due to computational time constraints. Instead, a small subset of alternative permutations is built heuristically to test the dominance conditions of Lemma 5.3.1.1 (see Section 5.3.3).

Figure 5.5

Possible Outcomes of Applying Lemma 5.3.1.1



(a) $S^1 \leqslant S'^1$. ζ' **dominates** ζ

(b) $S^1 > S'^1$ and $\bar{C}^{\max} = \bar{C}'^{\max}$. ζ' **dominates** ζ

(c) $S^1 > S'^1$ and $\bar{C}^{\max} > \bar{C}'^{\max}$. **no dominance of** ζ'

Note. Cases (a) to (c) compare schedules ζ and ζ' with $z(\zeta') < z(\zeta)$, as stated in Lemma 5.3.1.1.

5.3.2 Efficient Comparison of Permutation Schedules

Prior to the description of the heuristics that test permutation dominances, this subsection describes an efficient procedure to compare a feasible schedule ζ to a newly constructed schedule ζ', which only differs from the original schedule in terms of the production sequence (but not the delivery decisions) of the scheduled jobs. The procedure starts with a feasible schedule ζ comprising scheduled jobs J, delivery departure times D_j for all jobs $j \in J$ (inferred from transportation assignment P), and canonical completion times C_j for all jobs $j \in J$ (inferred from production sequence Π). This schedule is compared with an alternative schedule ζ' that differs by permutation $\Pi' \neq \Pi$ and delivery schedule P' with for all $i \in \{k, k+1 \dots, n^N\}$ it holds that $\rho'(i) = \rho(\pi^{-1}(\pi'(i)))$. That is, the transportation assignment P' is constructed such that delivery departure times are identical in both schedules; that is, $D_j = D'_j$ for each $j \in J (= J')$.

Each of the production sequences of both schedules can be described as a sequence of two subsequences:

$$\Pi = (\Pi_1, \Pi_2) \text{ and } \Pi' = (\Pi'_1, \Pi_2).$$

In other words, both permutations differ only in the subsequences Π_1 and Π'_1, while the second (possibly empty) subsequence Π_2 is identical for both schedules. As a consequence of the backward computation of canonical completion times by Procedure 5.2.2.1, the completion times (and therefore holding times and costs) of the jobs of subsequence Π_2 are identical in both schedules. Hence, only canonical completion times for subsequence Π'_1 must be computed to finalize the definition of schedule ζ'. The nonidentical subsequences are

$$\Pi_1 = (\pi(k), \pi(k+1), \dots, \pi(l))$$

and

$$\Pi'_1 = (\pi'(k), \pi'(k+1), \dots, \pi'(l))$$

with l denoting the position of the last job in subsequences Π_1 and Π'_1. Algorithm 5.2 computes the total cost $z(\zeta')$ the start time S'^1 and the feasibility status of ζ'.

Algorithm 5.2 Comparison of Permutation Schedules

```
1:  Requires: Schedules $\zeta$ and $\zeta'$ with $J = J'$, $\Pi = (\Pi_1, \Pi_2)$, $\Pi' = (\Pi'_1, \Pi_2)$, and $D_j = D'_j \; \forall j \in J (= J')$;
2:  $z(\zeta') \leftarrow z(\zeta)$;
3:  $t \leftarrow S_{\pi'(l+1)}$ if $\Pi_2 \neq \emptyset$, $t \leftarrow \infty$ otherwise;
4:  for $i = l$ to $k$ do
5:      $C'_{\pi'(i)} \leftarrow \min\{D'_{\pi'(i)}, t\}$;
6:      $t \leftarrow C'_{\pi'(i)} - p_{\pi(i)}$;
7:      if $t < r_{\pi(i)}$ then
8:          return infeasible;
9:      $z(\zeta') \leftarrow z(\zeta') + (D'_{\pi'(i)} - C'_{\pi'(i)}) \cdot c^H_{\pi'(i)} - (D_{\pi(i)} - C_{\pi(i)}) \cdot c^H_{\pi(i)}$;
10: $S'^1 \leftarrow t$;
11: return feasible;
```

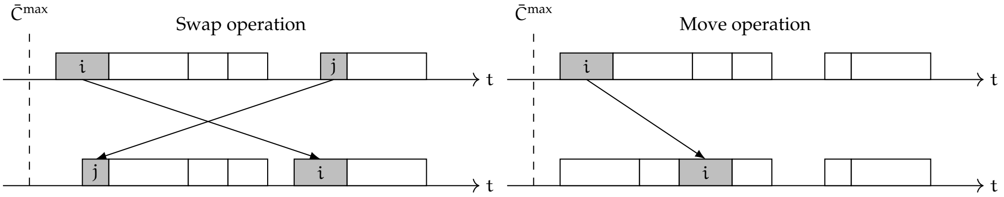
The algorithm derives the canonical completion times for the jobs of subsequence Π'_1. In each step of the procedure, the current schedule's feasibility is checked (Line 7). The algorithm terminates early if the permutation leads to an infeasible schedule. The total cost of schedule ζ' is initially set to $z(\zeta)$. In Line 9, the total cost is updated by reducing the total value by the holding cost contribution of the job at position i in schedule ζ and by adding the holding cost contribution of the job at position i in schedule ζ'. This cost update is sufficient, since the transportation decisions are identical in both schedules. The earliest start time of schedule ζ' is assigned in Line 10. Afterward, the maximum completion time $\bar{C}'^{\max}$ is computable with this value by applying the equation in Definition 5.3.1.1. Algorithm 5.2 executes on a subsequence of the production sequence in linear time and only needs to update the relevant data from the base schedule ζ. Therefore, it is more efficient than a full computation of the alternative schedule. The asymptotic runtime complexity is $\mathcal{O}(|\Pi_1|)$ with $|\Pi_1| \leqslant n^J$, as the procedure requires $|\Pi_1|$ steps.

5.3.3 Permutation Dominance Procedure

This subsection details the permutation dominance procedure by Bachtenkirch and Bock (2022) that generates a subset of alternative permutations based on Lemma 5.3.1.1, which is evaluated by Algorithm 5.2. The procedure tests two types of alternative schedules: The first type is an alternative schedule generated by exchanging the positions of the first scheduled job at $\pi(k)$ and another job in schedule ζ to form a new permutation Π', and the second type is an alternative schedule constructed by moving the job at the k-th position to a different position. Figure 5.6 illustrates both operations. The generation and testing of both types of alternative schedules are explained in Sections 5.3.3.1 and 5.3.3.2.

Figure 5.6

Illustration of the Swap and Move Operators



Note. Reprinted from “Finding efficient make-to-order production schedules” by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier.

5.3.3.1 Swap Dominance

Swap dominance describes the method of constructing alternative permutations from production sequence Π by exchanging the job located at the first determined position $(\pi(k))$ of schedule ζ with a later scheduled job in production sequence Π. In particular, the dominance procedure tests the following set of alternative permutations:

$$Z^S(\Pi) = \left\{\Pi_i^S \mid i = k+1, ..., n^N\right\}.$$

For each alternative permutation Π_i^S, it holds that job $\pi(k)$ occupies position i such that $\pi_i^S(i) = \pi(k)$, and job $\pi(i)$ occupies position k such that $\pi_i^S(k) = \pi(i)$, while all other job positions between positions k and i and after position i are identical for both sequences. That is, for a permutation $\Pi = (\pi(k), \pi(k+1), \pi(k+2), \dots, \pi(n^N))$, it constructs the following alternative permutations:

$$\begin{aligned} Z^S(\Pi) = \{ & \left(\pi(k+1), \pi(k), \pi(k+2), \dots, \pi(n^N)\right), \\ & \left(\pi(k+2), \pi(k+1), \pi(k), \dots, \pi(n^N)\right), \\ & \dots, \\ & \left(\pi(n^N), \pi(k+1), \pi(k+2), \dots, \pi(k)\right)\}. \end{aligned}$$

Each alternative permutation $\Pi_i^S \in N^S(\Pi)$ leads to a new alternative schedule ζ_i^S. The applied dominance procedure tests the permutations one by one by application of Algorithm 5.2, which computes the necessary information to test whether ζ is dominated by ζ_i^S according to Lemma 5.3.1.1. The swap dominance procedure follows the program flow illustrated in Figure 5.4 – each sequence permutation defines a new neighbor $N^d(\zeta)$ – and computes at most $n^J - 1$ exchanges for the positions $k+1, k+2, \dots, n^N$, which are evaluated by the application of Algorithm 5.2 in $O(n^J)$ time. Hence, the overall asymptotic time complexity of the swap dominance procedure is in $O(n^{J2})$ for a schedule ζ with n^J jobs. Notice that a generated alternative permutation does not necessarily constitute a feasible schedule. The computation provided by Algorithm 5.2 applies a feasibility test in each iteration that aborts the evaluation of a permutation early if infeasibility is detected.

In some cases, feasibility checks are unnecessary. Consider again the exchanged positions k and l of jobs i and j in sequence Π and production sequence

$$\Pi_l^S = (j, \pi_l^S(k+1), \dots, \pi_l^S(l-1), i, \pi_l^S(l+1), \dots, \pi_l^S(n^N)).$$

Algorithm 5.2 first determines the completion time of job i in schedule $\zeta' = \zeta_i^S$, and as job i starts later in the modified schedule (i.e., $S_i' > S_i$), feasibility for this job is guaranteed. One can distinguish between two cases concerning the jobs scheduled at positions $k+1, \dots, l-1$. If $S_i' \geqslant S_j$, then the completion times of the jobs between i and j are not processed earlier in schedule ζ'. Therefore, the feasibility test for these jobs is unnecessary, and only the feasibility of the decision to place job j at position k needs to be checked. Otherwise, if $S_i' < S_j$ holds, then the completion times for the jobs at positions $k+1, \dots, l-1$ may decrease, and release date constraints may therefore be violated. The feasibility check must thus be carried out in this case. If, in some iteration, an increased start time $S_h' \geqslant S_h$ for a job $h \in \{\pi'(l-1), ..., \pi'(k+1)\}$ is detected, then the feasibility test can be omitted for subsequent iterations of jobs at lower positions. Example 5.3.3.1 illustrates the swap operator.

Example 5.3.3.1. This example considers an instances with $n^N = 6$ jobs. Swap dominances are tested for a partial schedule ζ with $J = \{1, 2, 3, 4\}$ and production sequence $\Pi = (\pi(3), \pi(4), \pi(5), \pi(6))$ with $\pi(3) = 1, \pi(4) = 2, \pi(5) = 3$, and $\pi(6) = 4$. In this example, the swap permutation $\Pi_5^S = (3, 2, 1, 4)$ is considered, which exchanges the positions of job 1 (position 3) and job 3 (position 5). Additionally, the characteristics of the remaining jobs $\bar{J} = \{5, 6\}$ lead to $\bar{C}^{\max} = 30$. Transportation costs are omitted. The rest of this example is showcased below:

Comparison of schedules ζ and $\zeta' = \zeta_5^S$

Schedule ζ								Schedule ζ' with $\Pi' = \Pi_5^S$							
Pos (i)	Job j	r_j	S_j	p_j	C_j	D_j	c_j^H	Pos (i)	Job j	r_j	S_j'	p_j	C_j'	D_j'	c_j^H
3	1	20	37	10	47	70	2	3	3	30	35	12	47	70	1
4	2	45	47	8	55	55	3	4	2	45	47	8	55	55	3
5	3	30	58	12	70	70	1	5	1	20	60	10	70	70	2
6	4	40	70	7	77	77	4	6	4	40	70	7	77	77	4

After generating the alternative schedule, the necessary conditions for schedule ζ' to dominate schedule ζ, as stated by Lemma 5.3.1.1, must be checked. The cost of ζ totals $z(\zeta) = 46$ due to the holding costs of job 1. The swap of jobs 1 and 3 reduces the holding cost of job 1 to *zero*, while job 3 now contributes a *non-zero* holding cost value. In total, schedule ζ' has total cost value $z(\zeta) = 23$. Therefore, the first necessary condition $z(\zeta') < z(\zeta)$ is fulfilled. Furthermore, the start time $S^1 = S_1$ in schedule ζ is $S^1 = 37$ with, as initially stated, $\bar{C}^{\max} = 30$. The alternative schedule ζ' has an earlier start time $S^1 = S_3 = 35$, which does not decrease flexibility for the remaining jobs (i.e., $\bar{C}'^{\max} = \bar{C}^{\max}$). Hence, this example is of type case (b), as illustrated in Figure 5.5. Schedule ζ' **dominates** schedule ζ.

5.3.3.2 Move Dominance

Instead of exchanging the positions of *two* jobs, another approach to find schedules that potentially dominate a considered schedule ζ is to move the job at the first position to another position in the production sequence π. Analogous to swap dominance, the move dominance procedure constructs the following alternative permutations:

$$Z^M(\Pi) = \{\Pi_i^M \mid i = k+1, ..., n^N\}$$

where for each permutation Π_i^M, it holds that $\pi_i^M(h) = \pi(h+1)$ for each $h \in \{k, ..., i-1\}$, and $\pi_i^M(i) = \pi(k)$ and $\pi_i^M(h) = \pi(h)$ for each $h \in \{i+1, \ldots, n^N\}$. Hence, the set $Z^M(\Pi)$ comprises the following sequences:

$$\begin{aligned} Z^M(\Pi) = \big\{ &\big(\pi(k+1), \pi(k), \pi(k+2), \ldots, \pi(n^N)\big)\,, \\ &\big(\pi(k+1), \pi(k+2), \pi(k), \ldots, \pi(n^N)\big)\,, \\ &\ldots, \\ &\big(\pi(k+1), \pi(k+2), \ldots, \pi(n^N), \pi(k)\big)\big\}\,. \end{aligned}$$

Each alternative permutation $\Pi_i^M \in Z_i^M(\Pi)$ defines a new alternative schedule $\zeta' = \zeta_i^M$ that is evaluated by Algorithm 5.2. In this procedure, the feasibility test can be omitted if, in some iteration, $S'_h \geqslant S_h$ for a job $h \in \{\pi'(l-1), ..., \pi'(k)\}$ applies, where l is the insertion position of job $\pi(k)$. The asymptotic time complexity of the move dominance procedure is equivalent to the swap dominance procedure and therefore in $O(n^{J2})$. Example 5.3.3.2 demonstrates the application of the move dominance procedure.

Example 5.3.3.2. Consider the same instance and schedule ζ as stated in Example 5.3.3.1 and the move permutation $\Pi_4^M = (2, 1, 3, 4)$, which moves job 1 from position 3 to position 4, thereby forcing job 2 to position 3 of the partial schedule ζ_4^M. The execution of this move is depicted below:

Comparison of schedules ζ and ζ'

Schedule ζ								Schedule $\zeta' = \zeta_4^R$ with $\Pi' = \Pi_4^m$							
Pos (i)	Job j	r_j	S_j	p_j	C_j	D_j	c_j^H	Pos (i)	Job j	r_j	S'_j	p_j	C'_j	$D'_j =$	c_j^H
3	1	20	37	10	47	70	2	3	2	**45**	**40**	8	48	55	3
4	2	45	47	8	55	55	3	4	1	20	48	10	58	70	2
5	3	30	58	12	70	70	1	5	3	30	58	12	70	70	1
6	4	40	70	7	77	77	4	6	4	40	70	7	77	77	4

Again, the necessary conditions must be checked for schedule ζ' to dominate schedule ζ, as stated in Lemma 5.3.1.1. The reinsertion of job 1 from position 3 to position 4 results in a cost decrease, as the total cost amounts to $z(\zeta') = 45$. As highlighted in the above table, this schedule is not feasible, since $S_2 < r_2$ violates release date constraints for job 2. Therefore, schedule ζ **is not dominated** by schedule ζ'.

5.3.3.3 Dominance Test

The SFDDHT-B&B implementation applies Lemma 5.3.1.1 by testing dominance first with the swap permutations and then with the move permutations. That is, for each $i \in \{k+1, \ldots, n^N\}$, it performs the computation of Algorithm 5.2 with the permuted sequence $\Pi_i^S \in Z^S(\Pi)$. If a dominating schedule ζ' is found, then the computation ends, and the current node, which defines schedule ζ, is discarded from the enumeration process. Otherwise, for each $i \in \{k+2, \ldots, n^N\}$, it performs the computation of Algorithm 5.2 with the permuted sequence $\Pi_i^M \in Z^M(\Pi)$ and again discards the current node, which defines schedule ζ if a dominating schedule ζ' was constructed. Note that Π_{k+1}^S and Π_{k+1}^M are identical and therefore only tested once.

5.3.4 Production Block Dominance Procedure

The second dominance procedure targets the sequencing of jobs within a production block. Specifically, the procedure establishes precedence constraints for two currently scheduled jobs h and j of the same production block. Any identified precedence violation by the procedure proves that the current schedule ζ is locally suboptimal and can be dismissed from further exploration. The dominance relation by Bachtenkirch and Bock (2022) that establishes such precedence constraints is stated in Proposition 5.3.4.1.

Proposition 5.3.4.1. *Let h and j be two jobs processed in the stated order with identical production block Γ_l for the l-th transport of set I^T for a given schedule ζ. Additionally, let α be the set of jobs of production block Γ_l that are processed between jobs h and j, and let the total processing time of these in-between jobs be $P_\alpha = \sum_{i \in \alpha} p_i$. If*

$$c_h^H \cdot (P_\alpha + p_j) + (p_j - p_h) \cdot \sum_{i \in \alpha} c_i^H > c_j^H \cdot (P_\alpha + p_h) \tag{5.6}$$

and

$$S_i + (p_j - p_h) \geqslant r_i \quad \forall i \in \alpha \cup \{j\} \tag{5.7}$$

apply, then an optimal schedule ζ' must exist in which job h is not processed before job j in production block Γ_l.

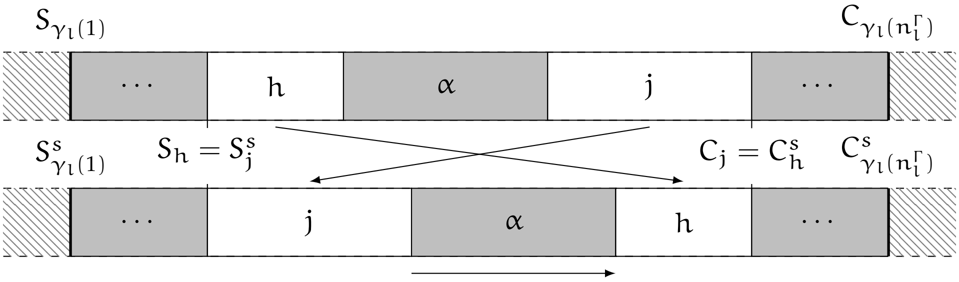
Proof. Proposition 5.3.4.1 considers an exchange of positions for two jobs h and j in a production block Γ_l. Examine the modified schedule ζ^s with exchanged positions for jobs h and j, as considered by Proposition 5.3.4.1. In this modified schedule ζ^s, job j is scheduled before jobs α, and job h is scheduled after jobs α. Lemma 5.2.5.1 states that all jobs that are part of the same production block are assigned to delivery departure times that are not before the latest completion time of a job within this block. As the jobs are scheduled canonically and therefore consecutively, no idle time exists between jobs of the same production block (see Lemma 5.2.2.1).

Equation (5.6) assumes that job h completes at time $C_h^s = C_j$ in schedule ζ^s. This choice for job h is feasible due to $D_h \geqslant C_j$ in schedule ζ. Scheduling job h at C_j, jobs α canonically before job h, and lastly job j before all jobs of α results in start time $S_j^s = S_h$ for job j. Therefore, the earliest start and latest completion time of the processing of the jobs $\{h\} \cup \alpha \cup \{j\}$ are left unmodified after the exchange. No job scheduled before job j and after job h is consequently affected in terms of its processing on the machine in modified schedule ζ^s.

The exchange results in job h's increased completion time by $(P_\alpha + p_j)$ TUs whereas job j completes $(P_\alpha + p_h)$ TUs earlier. The completion times of the jobs scheduled in-between decrease by amount $(p_h - p_j)$ if $p_h > p_j$, and they increase $(p_h < p_j)$ or remain unchanged $(p_h = p_j)$ otherwise. Inequality (5.6) states a decrease in the total holding costs by exchanging jobs h and j. Furthermore, Inequality (5.7) ensures that no release date violations occur for jobs $\alpha \cup \{j\}$. Job h starts later than before and is therefore feasibly scheduled. □

Figure 5.7

Illustration of Proposition 5.3.4.1



Note. In above schedule, production block Γ_l comprises the jobs h, j, and α for which it holds that $p_h < p_j$ and α is nonempty. Adapted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

The dominance relation of Proposition 5.3.4.1 establishes precedence relations between jobs in the same production block. Figure 5.7 illustrates the dominance relation for two jobs h and j with $p_h < p_j$ and a non-empty set of jobs α for a production block Γ_l. Jobs scheduled before j and h are not affected by the exchange, while for $p_h < p_j$, the jobs of set α have increased completion times after the exchange.

The computation proposed by Proposition 5.3.4.1 considers the holding costs of all affected jobs in an exchange (Inequality [5.6]) as well as the feasibility requirements (Inequality [5.7]). In some cases, this computation can be simplified. The following *three* cases stated in Proposition 5.3.4.2 originate from Bachtenkirch and Bock (2022).

Proposition 5.3.4.2. *Schedule ζ is dominated if a production block $\Gamma_l \in \Gamma$ exists with job $h \in \Gamma_l$ scheduled before job $j \in \Gamma_l$, with $r_j \leqslant \max\{r_h, S_h\}$ and **one** of the following cases applies:*

Case 1. $\alpha = \varnothing$ *and* $c_j^H/p_j < c_h^H/p_h$.

Case 2. $p_h \leqslant p_j$ *and* $c_j^H \cdot (P_\alpha + p_h) < c_h^H \cdot (P_\alpha + p_j)$

Case 3. $p_h \leqslant p_j$ *and* $c_j^H \cdot (P_\alpha + p_h) + (p_h - p_j) \cdot \sum_{i \in \alpha} c_i^H < c_h^H \cdot (P_\alpha + p_j)$.

Proof. Let ζ^s be the schedule derived from schedule ζ that exchanges the positions of jobs h and j. All three cases assume that $r_j \leqslant \max\{r_h, S_h\}$ holds. An exchange of jobs h and j leads to $S_j^s = S_h$. Hence, an exchange is guaranteed to be feasible for j.

In *Case 1*, $\alpha = \varnothing$ holds. Therefore, jobs h and j are scheduled next to each other in production block Γ_l, and the cost and feasibility considerations of Proposition 5.3.4.1 for jobs scheduled between h and j can hence be omitted. The term $c_h^H \cdot p_j < c_j^H \cdot p_h$ can be written as $c_j^H/p_j < c_h^H/p_h$.

In *Cases 2* and *3*, $p_h \leqslant p_j$ holds. Therefore, the completion times of the jobs of set α do not decrease. Holding costs consequently do not increase for these jobs, and release date violations do not occur. *Case 2* considers solely the increase in holding costs of job j and the decrease in holding costs of job h. *Case 3* additionally accounts for the potential holding cost decrease for the jobs scheduled between j and h.

As a result, all cases are simplifications of Proposition 5.3.4.1 that apply to special cases. □

5.3.4.1 Dominance Test

Algorithm 5.3 Dominance Test for Two Jobs h and j Scheduled in the Same Production Block Γ^1 that Applies Propositions 5.3.4.1 and 5.3.4.2

Input: A partial schedule ζ. **Output: true** if ζ is dominated, **false** otherwise.

Step 1: Set $h = \pi(k)$, $P(\alpha) = 0$, $H(\alpha) = 0$ and $i = 2$.

Step 2: Let $j = \gamma^1(i)$. Define the holding costs for jobs h and j at position k as $h(h) = c_h^H \cdot (P(\alpha) + p_j)$ and $h(j) = c_j^H \cdot (P(\alpha) + p_h)$. The exchange is *feasible* if $r_j \leqslant max\{r_h, S_h\}$. The cases from Proposition 5.3.4.2 can be applied if the exchange is feasible and $p_h \leqslant p_j$ holds. If both requirements and additionally $h(j) < h(h)$ hold true, then **return true** (ζ is dominated), otherwise goto **Step 3**.

Step 3: The holding costs for the jobs in α are $h(\alpha) = (p_h - p_j) \cdot H(\alpha)$. If $p_h \leqslant p_j$ and $(h(j) + h(\alpha)) < h(h)$, then the exchange improves cost; **return true** (ζ is dominated), otherwise goto **Step 4**.

Step 4: If $(h(j) + h(\alpha)) < h(h)$, calculate the start times for all jobs $i \in \alpha$ and test whether $S_i > r_i$ holds. If the exchange is feasible, then **return true** (ζ is dominated), otherwise goto **Step 5**.

Step 5: Update the set α, i.e. set $P(\alpha)$ to $P(\alpha) + p_j$ and $H(\alpha)$ to $H(\alpha) + c_j^H$. Increment i to $i + 1$. If $i > |\Gamma^1|$, then **return false** (ζ is not dominated). Else, goto **Step 2**.

The second dominance test only considers the first non-empty production block – in the following denoted as $\Gamma^1 = \{\gamma^1(1), \cdots, \gamma^1(|\Gamma^1|)\}$ – of the current schedule ζ with $h = \pi(k) = \gamma^1(1)$. The dominance relation proposed by Proposition 5.3.4.1 is tested for job h and each successor in production block Γ^1; that is, for $j \in \{\gamma^1(2), \cdots, \gamma^1(|\Gamma^1|)\}$ in this order. For each combination of h and j, the procedure uses the computationally least expensive calculations first, while interrupting the computation if dominance over ζ can be proven. The dominance procedure stores intermediate results such that the subsequent tested cases access previously computed values. Algorithm 5.3 employs memorization to speed up the computation of the testing procedure. The algorithm runs for $|\Gamma^1 - 1|$ iterations at most; that is, $|\Gamma^1 - 1|$ exchanges are tested by application of Propositions 5.3.4.1 and 5.3.4.2. All operations performed within each iteration are basic and executed in $\mathcal{O}(1)$. Algorithm 5.3 consequently has asymptotic time complexity $\mathcal{O}(|\Gamma^1|)$ when applied to a schedule ζ with production block Γ^1.

5.3.5 Delivery Batch Dominance Procedure

The third dominance procedure identifies suboptimal delivery batches in a schedule ζ. The proposed dominance relation identifies cases in which the merger of two delivery batches for the same customer into a single delivery batch leads to a schedule ζ' that dominates the original schedule ζ. That is, the split delivery of the two sets of jobs is consolidated into a single delivery. To properly define the proposed dominance relation, the procedure distinguishes between *open* and *closed* delivery batches (Bachtenkirch and Bock, 2022). A delivery batch is closed if it cannot be extended by an additional job in any schedule derived from the current schedule ζ, otherwise it is considered open. Next, the conditions leading to the open or closed status of a delivery batch are defined. Definition 5.3.5.1 first introduces the earliest delivery departure time of a customer in a schedule:

Definition 5.3.5.1. Let $t_g^{\min}$ denote the *earliest used delivery departure time of customer* $g \in G$ in a schedule ζ. Additionally, let $t_g^{\min} = \infty$ for schedules without scheduled jobs for customer g.

Furthermore, Proposition 5.3.5.1 introduces $\bar{J}(g,l)$ as the *set of remaining jobs for customer* g *that can still be assigned to the* l*-th delivery departure time of customer* g.

Proposition 5.3.5.1. *For each customer* $g \in G$ *and transport* $l \in \{1, 2, \dots, n_g^T\}$*, the following holds:*

$$\bar{J}(g,l) = \left\{ j \mid j \in \bar{J}_g, r_j + p_j \leqslant t_{g,l} \leqslant \min\left\{\bar{d}_j, t_g^{\min}\right\}\right\}.$$

Proof. From the problem definition it immediately follows that a job j ordered by customer g is only assignable to the l-th transport to customer g if job j

a) can be completed before the departure time (i.e., $r_j + p_j \leqslant t_{g,l}$ holds) *and*
b) does not have to be delivered earlier (i.e., $\bar{d}_j \geqslant t_{g,l}$ holds).

Additionally, from Property 5.2.5.2, it follows that no remaining job $j \in \bar{J}_g$ can be optimally assigned to a departure time $t > t_g^{\min}$, since assignments to an earlier, already used transport is more cost efficient. □

As a consequence of Proposition 5.3.5.1, *four* conditions exist for a set $\bar{J}(g,l)$ to be an empty set, as stated by Observation 2.

Observation 2. One of the four sufficient conditions must be met to claim that $\bar{J}_{g,l}$ is empty:

(*i*) $\bar{J}(g,l) = \varnothing$ if $t_{g,l} > t_g^{\min}$,

(*ii*) $\bar{J}(g,l) = \varnothing$ if $t_{g,l} > \max\limits_{j \in \bar{J}_g}\{\bar{d}_j\}$,

(*iii*) $\bar{J}(g,l) = \varnothing$ if $t_{g,l} < \min\limits_{j \in \bar{J}_g}\{r_j + p_j\}$, or

(*iv*) $\bar{J}(g,l) = \varnothing$ if $\bar{J}_g = \varnothing$.

Let $\Delta(g)$ denote the *set of delivery batches for customer* g; that is, $\Delta(g) = \{\Delta_l \mid \Delta_l \in \Delta, l \in I^T_g\}$ for each customer $g \in G$. Additionally, the individual delivery batches for customer g are indexed by increasing departure time – that is, $\Delta(g) = \{\Delta_{g,1}, \Delta_{g,2}, \cdots, \Delta_{g,n^T_g}\}$ – such that $t(\Delta_{g,l}) = t_{g,l}$. The set of *closed delivery batches* $\Delta^c(g)$ can now be stated for each customer $g \in G$ as follows:

$$\Delta^c(g) = \left\{\Delta_{g,l} \in \Delta(g) \mid \bar{J}(g,l) = \varnothing\right\}.$$

Evidently, a delivery batch is closed if there is a preceding (lower indexed) non-empty delivery batch for the same customer g in schedule ζ; the deadlines for all remaining jobs of customer g are earlier than the delivery departure time for Δ_l; or all jobs of customer g are already scheduled, and no job therefore can be appended to the current delivery batch. For completeness, the *set of open delivery batches* for customer g is

$$\Delta^o(g) = \Delta(g) \setminus \Delta^c(g).$$

An important property of any nonempty closed delivery batch $\Delta \in \Delta^c(g)$ for some customer g is that the reassignment of all jobs of this closed delivery batch to a later delivery time does not affect the remaining jobs $\bar{J}_g$ in the choice of their delivery time. This is due to the fact, that no remaining job $j \in \bar{J}_g$ can be added to this batch Δ, as it is closed. The following dominance relation by Bachtenkirch and Bock (2022) identifies suboptimal closed delivery batches. Informally, a closed delivery batch in a schedule ζ is identified as suboptimal if the jobs of this delivery batch can be reassigned to a later delivery time and if this reassignment reduces the total cost of a schedule. The dominance relation is formalized by Proposition 5.3.5.2.

Proposition 5.3.5.2. *Consider a schedule ζ with a nonempty closed delivery batch $\Delta_{g,l} \in \Delta^c(g)$ and a second nonempty closed delivery batch $\Delta_{g,o} \in \Delta^c(g)$ of identical customer $g \in G$, with $o > l$. In this setting, the o-th delivery batch of customer g is defined for a later delivery departure time compared to the l-th batch of customer g.*

The schedule ζ is dominated by a feasible schedule ζ' with an identical set of jobs $J = J'$ and identical sequence $\Pi = \Pi'$ that joins the jobs of both delivery batches $\Delta_{g,l}$ and $\Delta_{g,o}$ to define the delivery batch $\Delta'_{g,o}$ and leaves $\Delta'_{g,l}$ empty. That is, $\Delta'_{g,o} = \Delta_{g,o} \cup \Delta_{g,l}$ with $D'_j = t_{g,o}$ for each $j \in \Delta'_{g,o}$ and $\Delta'_{g,l} = \varnothing$.

If the schedule ζ' is feasible, which is guaranteed if $D'_j \leqslant \bar{d}_j$ for each $j \in \Delta'_{g,o}$ holds, and has a lower objective value – that is, $z(\zeta') < z(\zeta)$ holds – then schedule ζ is dominated by schedule ζ'.

Proof. The modification of the delivery departure times D_j for jobs $j \in \Delta_{g,o}$ to the new delivery departures time $D'_j = t_{g,o}$ may increase (but never reduce) the completion times of the jobs $\Delta_{g,l}$ as well as all jobs $\{\pi(k), \pi(k+1), \ldots, \delta_{g,o}(n^\Delta_{g,o})\}$. This property follows from the rescheduling of jobs by Procedure 5.2.2.1, which leads to a canonical schedule. The schedules flexibility is consequently preserved; that is, $\bar{C}'^{\max} \geqslant \bar{C}^{\max}$ is guaranteed. As the delivery batch $\Delta_{g,l}$ is closed, the delivery departure time $t_{g,l}$ is unreachable for any remaining job $j \in \bar{J}_g$ of customer g. Therefore, closing this batch does not negatively affect the transport assignments of the remaining jobs. Proposition 5.3.5.2 assumes that

ζ' is feasible (i.e., $\bar{d}_j \geqslant t_{g,o}$ for each job $j \in \Delta_{g,l}$), and has a lower cost than schedule ζ. Therefore, schedule ζ is dominated by schedule ζ'. □

Note that the modification proposed by Proposition 5.3.5.2 eliminates a transport to customer g and reduces the total cost value by fixed transportation cost value c^T_g. The holding costs for the jobs in delivery batch $j \in \Delta_{g,l}$ are guaranteed to increase due to the assignment to a later delivery departure time by at most $\sum_{j \in \Delta_{g,o}} c^H_j \cdot (t_{g,o} - t_{g,l})$ holding costs. The actual holding cost value might be lower, since the completion times of the jobs potentially increase by canonical rescheduling. Since the completion times of other jobs also potentially increase, additional holding cost savings might be realized by this modification.

5.3.5.1 Dominance Test

The B&B algorithm tests the proposed dominance relation of Proposition 5.3.5.2 only for a schedule ζ if the addition of the job $\pi(k)$ leads to the closure of a previously open delivery batch. Therefore, the procedure avoids redundant testing on later sections of a (partial) schedule, as these were already tested during evaluation of ancestor nodes. To determine the tests to execute, the dominance procedure first identifies the *newly closed delivery batches*, closed by adding job $\pi(k)$, and proceeds only if such a closed delivery batch and a suitable delivery batch exist such that a merge, as described by Proposition 5.3.5.2, is executable.

Let $\Delta^\bullet(g) \subseteq \Delta(g)$ denote the *set of nonempty delivery batches of customer* g *in schedule* ζ; that is, $\Delta^\bullet(g) = \{\Delta_{g,l} \in \Delta(g) \mid \Delta_{g,l} \neq \varnothing\}$. In the following, let $g^N(\pi(k)) = g$. If $|\Delta^\bullet(g)| = 1$ holds, then Proposition 5.3.5.2 is not applicable, as it proposes merging of two nonempty delivery batches. Therefore, let $\Delta^\bullet_1(g)$ and $\Delta^\bullet_2(g)$, denote the first two (in order of delivery departure time) nonempty delivery batches transported to customer g for a schedule ζ with $|\Delta^\bullet(g)| > 1$, and additionally, let $\Delta^\bullet_3(g)$ denote the third nonempty delivery batch transported to customer g for a schedule ζ with $|\Delta^\bullet(g)| > 2$. The dominance test detects whether the first two delivery batches are *newly closed* as follows:

- Job $\pi(k)$ is added to $\Delta^\bullet_1(g)$, which is the first nonempty delivery batch. If this batch is also closed, verified by one of the conditions of Observation 2, then the batch was open prior to the addition of $\pi(k)$ and is therefore *newly closed*.
- $\Delta^\bullet_2(g)$ was previously open if $\bar{d}_{\pi(k)} \geqslant t(\Delta^\bullet_2(g))$. That is, job $\pi(k)$ was assignable to this delivery batch prior to its assignment to delivery batch $\Delta^\bullet_1(g)$ and is therefore newly closed.

The procedure computes, at most, *two* dominance tests for a schedule ζ, one for each of the following two cases:

Test 1. $\Delta^\bullet_1(g)$ is newly closed: Set $\Delta_{g,l}$ of Proposition 5.3.5.2 to $\Delta^\bullet_1(g)$, and set $\Delta_{g,o}$ to $\Delta^\bullet_2(g)$.

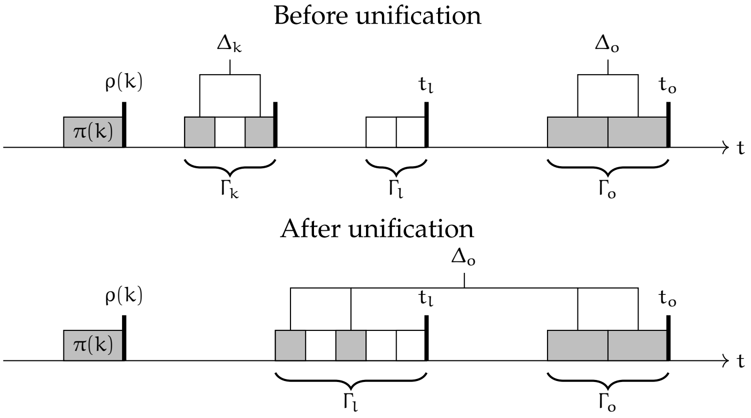
Test 2. $\Delta^\bullet_2(g)$ is newly closed: Set $\Delta_{g,l}$ of Proposition 5.3.5.2 to $\Delta^\bullet_2(g)$, and set $\Delta_{g,o}$ to $\Delta^\bullet_3(g)$.

Note that both tests may apply for the same schedule ζ such that the procedure carries out two tests in the stated order. If the two delivery batches required for Proposition 5.3.5.2

can be successfully built for either of the two tests, then the procedure executes the modifications proposed by that proposition. Furthermore, if schedule ζ' fulfills all necessary conditions to establish dominance over ζ, then the current schedule is dismissed, since it is suboptimal. Figure 5.8 illustrates the second test by use of the general (customer agnostic) indexing of delivery batches. The two delivery batches Δ_l and Δ_o are merged to constitute schedule ζ'. The modified schedule does not transport any job at time t_l and instead delays transportation of the jobs Δ_l to the transport at time t_o. In this example, dissolving delivery batch Δ_l additionally leads to the merging of production blocks Γ_l and Γ_l due to the increased delivery departure time of the last job γ_{l,n_l^Γ}.

Figure 5.8

Illustration of the Merge of Delivery Batches



Note. The illustration depicts the unification of production blocks Γ_k and Γ_l due to joint delivery of jobs Δ_k and Δ_o at time t_o. Reprinted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier.

The computation of the applicability of Proposition 5.3.5.2 is carried out in linear time. The implementation makes a single pass through the current schedule ζ to check whether the nonempty delivery batches $\Delta_1^\bullet(g)$, $\Delta_2^\bullet(g)$, and $\Delta_3^\bullet(g)$ exist and whether delivery batches $\Delta_1^\bullet(g)$ and $\Delta_2^\bullet(g)$ are newly closed. Additionally, alongside this check, the required job information is stored to construct the alternative schedules. The delivery batches for customer g are found by iterating the production sequence Π and grouping the jobs of customer g into disjoint sets, distinguished by their delivery departure times, that define the first nonempty delivery batches for customer g. Once the first nonempty delivery batch is found, the check for closedness of $\Delta_1^\bullet(g)$ is performed by testing the sufficient conditions of Observation 2, for which the second condition *(ii)* (which is the most time-consuming condition) requires a check for whether the deadlines of the remaining jobs of customer g make further assignments to delivery departure time $t(\Delta_1^\bullet(g))$ impossible. The closedness of $\Delta_2^\bullet(g)$ immediately follows from the existence of $t(\Delta_1^\bullet(g))$. The same argument applies for the second and third nonempty delivery batches. Upon verification that $\Delta_1^\bullet(g)$ is newly closed and $\Delta_2^\bullet(g)$ exists, the alternative canonical schedule ζ' is built and evaluated backwards, starting with the last job of delivery batch $\Delta_{g,l}$. If Proposition 5.3.5.2 applies, then the procedure is terminated, otherwise the last steps are repeated for delivery batches $\Delta_2^\bullet(g)$ and $\Delta_3^\bullet(g)$. The evaluation is carried out in linear time. In the worst case, the dominance test requires the scheduled jobs J and the unscheduled jobs of customer g to be analyzed once and requires two evaluations for schedule ζ. The asymptotic run time complexity is consequently in $\mathcal{O}(n^{\bar{J}}_g + n^J + 2 \cdot n^J)$, which is $\mathcal{O}(n^{\bar{J}}_g + n^J)$.

5.3.6 Dominance Table

The final proposed dominance procedure is memory-based. The procedure compares the current schedule under investigation ζ to already processed schedules by comparing ζ with schedules stored in a *dominance table*. Proposition 5.3.6.1 by Bachtenkirch and Bock (2022) establishes a dominance relation for two schedules:

Proposition 5.3.6.1. *Let ζ and ζ' denote two schedules for which all of the following statements hold:*

1. $J = J'$, *(identical set of scheduled jobs)*
2. $(\Pi, P) \neq (\Pi', P')$; *(non-identical decisions)*
3. $z(\zeta') < z(\zeta)$; *(lower total cost for schedule ζ')*
4. $\bar{C}'^{\max} \geqslant \bar{C}^{\max}$; *and* *(schedule ζ' is not less flexible)*
5. $\forall g \in G$, *with* $J_g \neq \emptyset$ *and* $\bar{J}(g, t^{\min}_g) \neq \emptyset$, *it holds that* $t^{\min}_g \geqslant t'^{\min}_g$.
   That is, the earliest already used transports of schedule ζ' to a customer g depart not later, compared to the transports of schedule ζ, if there are remaining jobs left in both schedules that can still be assigned to this earliest transport.

If all five statements hold, then schedule ζ is dominated by schedule ζ' and can be discarded.

Proof. Proposition 5.3.6.1 extends Lemma 5.3.1.1 by considering different permutations for an identical set of jobs and different transportation decisions. The correctness of the dominance property of statements 1 – 4 for schedules with identical transportation decisions are already addressed by the proof of Lemma 5.3.1.1. That is, any feasible decision for the remaining jobs conducted for schedule ζ can also be conducted for schedule ζ'. Therefore, the less costly (and not less flexible) schedule ζ' can be extended towards complete schedules with a lower cost compared to schedule ζ.

Hence, it only needs to be shown that the validity of the *fifth* statement establishes dominance between two schedules ζ and ζ' with additionally unequal transportation decisions. As stated by Proposition 5.3.6.1, the requirement of $t^{\min}_g \geqslant t'^{\min}_g$ only applies to customers who

(a) have remaining jobs left ($\bar{J}_g \neq \emptyset$) that
(b) can be assigned to delivery departure time $t^{\min}_g$ ($\bar{J}(g, t^{\min}_g) \neq \emptyset$),

because identical transportation assignments of the remaining jobs to delivery departure times earlier than $t^{\min}_g$ result in equal ($t'^{\min}_g = t^{\min}_g$) or potentially lower ($t'^{\min}_g < t^{\min}_g$) transportation costs for schedule ζ' compared to schedule ζ.

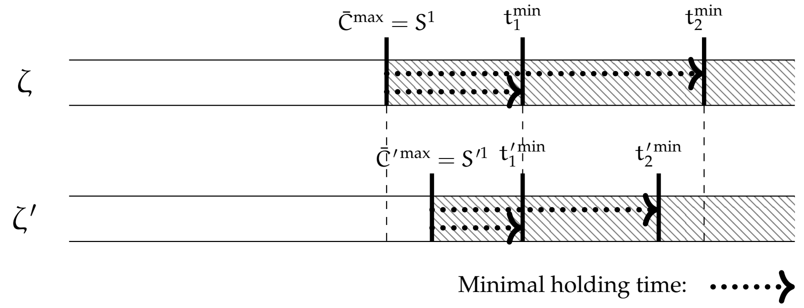
The customers for which the requirement applies have at least one job that is assignable to the first already used transportation that departs at time $t^{\min}_g / t'^{\min}_g$. The minimal holding time of assigning a job to this transport is $t^{\min}_g - \bar{C}^{\max}$ for schedule ζ, and $t'^{\min}_g -$

$\bar{C}'^{\max}$ for schedule ζ'. Due to $t'^{\min}_g \leqslant t^{\min}_g$ and $\bar{C}'^{\max} \geqslant \bar{C}^{\max}$, one concludes that $t'^{\min}_g - \bar{C}'^{\max} \leqslant t^{\min}_g - \bar{C}^{\max}$, and the holding cost of assigning a job to $t'^{\min}_g$ is therefore not larger compared to $t^{\min}_g$. Schedule ζ' consequently dominates schedule ζ. □

Figure 5.9 illustrates the two schedules ζ and ζ' of Proposition 5.3.6.1. In this example, $\bar{C}'^{\max}$ is later than $\bar{C}^{\max}$. Moreover, the earliest used delivery times for the first customer are identical, and the earliest used delivery times for the second customer are earlier for schedule ζ'.

Figure 5.9

Illustration of Proposition 5.3.6.1



Note. Adapted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

5.3.6.1 Dominance Test

The SFDDHT-B&B tests the dominance relation of Proposition 5.3.6.1 for the currently considered schedule ζ and a suitable set of stored schedules in a dominance table. Apart from the stated comparisons, the dominance test requires no additional computations. To check the proposed conditions, the required information is stored for each already processed schedule in memory. To guarantee fast access to comparable schedules with identical jobs scheduled ($J = J'$), the following binary encoding scheme is used:

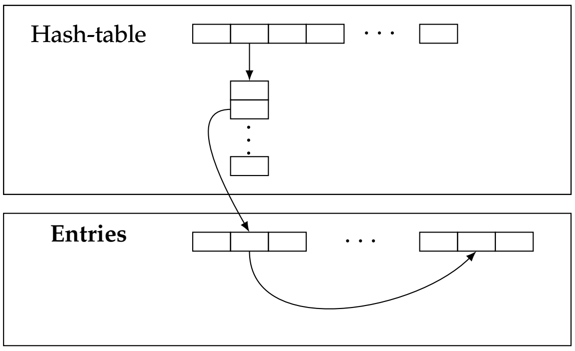
Let $B^J(J) = (b^J_1, b^J_2, \ldots, b^J_{n^N})$ denote a sequence of bits of length n^N, indicating whether a job is scheduled in ζ. Hence, for each job $j \in N$, the entry

$$b^J_j = \begin{cases} 1 & \text{if } j \in J, \\ 0 & \text{otherwise.} \end{cases}$$

The data structure for the implemented dominance table is illustrated in Figure 5.10. For each encountered set of jobs J defined by the schedule ζ, the binary encoding $B^J(J)$ is computed. By application of a hash function, a hash table stores the index of the first table entry (stored in a continuous array) with the set of scheduled jobs J. Each entry of this array points to the subsequent entry with an identical set of scheduled jobs such that all entries with identical jobs are accessible. Note that the number of schedules that are inserted into the dominance table is unknown, as the number of schedules that will be evaluated during the application of the B&B algorithm is determined dynamically. The implementation hence pre-allocates a large amount of memory to store the dominance table entries. The hash table also uses dynamic allocation. It comprises an array of a limited size of buckets, where each bucket stores values with identical hash values; therefore, collisions of different objects with identical hash values are resolved by a linear search over the bucket entries. Once the hash table stores a certain number of elements, it is allowed to grow by enlarging the capacity of the array and rehashing all stored elements. For the hash table[2] and the hash function,[3] the built-in implementation of the C++ standard library implementation of the GNU GCC compiler (version 10.2) is used.

Figure 5.10

Memory Layout of the Implemented Dominance Table



The computational complexity of the memory-based dominance test is not easily derivable, as the computational time it requires is impacted by the size of the hash-table, the computational time necessary to grow the hash table, and the number of already stored schedules. Therefore, further analysis of the computational complexity is omitted. The dominance table stores schedules in the form of a representation that comprises the necessary information to test Proposition 5.3.6.1. The representation of a schedule ζ comprises the following information:

- The B&B search node index for ζ;
- The index of the next entry with identical job set J in the dominance table, or a default value indicating that this entry is the last of the incidental list of entries with identical jobs;
- The earliest departure time values $t_1^{\min}, t_2^{\min}, \ldots, t_{n^G}^{\min}$;
- The latest completion time for the remaining jobs $\bar{C}^{\max}$;
- The total cost $z(\zeta)$.

The dominance table processes a new schedule ζ by testing whether it dominates schedules already inserted into the table with identical jobs or whether it is dominated by an already inserted schedule. If schedule ζ itself is not dominated by any other schedule, then its representation is added to the table. Otherwise, if ζ is dominated by another schedule, then the process terminates, and schedule ζ is discarded by the B&B approach. All schedules

[2] The hash table uses the GNU GCC compiler implementation of the std::unordered_map data structure.

[3] The hash-function uses an implementation of the murmurhash2 algorithm. See https://sites.google.com/site/murmurhash/ (Last visited on 15.02.2021).

that are dominated by a newly processed schedule are removed from the table and labeled as dominated by the B&B approach. Upon encountering a child node of this node during further branching (which is detected when recursively building Π and P to define the canonical schedule), the B&B approach immediately deletes the child node.

5.3.7 Application Order of the Dominance Procedures

The B&B approach applies the proposed dominance tests in order of increasing computational complexity. As such, it starts with the dominance test of Section 5.3.4 specified for the first production block that applies dominance relation Proposition 5.3.4.1. Next, the two heuristic sequence permutation tests, based on Lemma 5.3.1.1, are performed as described in Sections 5.3.3.1 and 5.3.3.2. Afterward, the delivery batch merging is tested as described in Section 5.3.5. Finally, Proposition 5.3.5.2 is checked with the help of the described dominance table from Section 5.3.6. If domination of schedule ζ is proven at any point, then the application of dominance tests ends and schedule ζ is immediately dismissed from the branching process. That is, the current node containing the information for schedule ζ is deleted.

5.4 Preprocessing Procedures

This section describes preprocessing procedures that compute rules that eliminate branching decisions before the tree enumeration starts. The procedures thus only use data from the original problem instance. This section proposes two types of precomputed constraints. The first type establishes *precedence constraints* between two jobs: Scheduling job i before job j is required in all feasible schedules. Hence, the B&B algorithm discards any branching decision that schedules job j before i during the enumeration process. The second type establishes *position constraints* that allow the scheduling of a job j only on a subset of positions $\{1, 2, \dots, n^N\}$ in any feasible schedule. Therefore, the B&B discards any branching decision suggesting the scheduling of job j on a forbidden position or leading to job j left unscheduled without a remaining feasible position. Both preprocessing procedures only operate on the data concerning the processing of jobs on the machine.

5.4.1 Precedence Constraints

This section establishes precedence constraints between two jobs i and j of N based on their processing time windows. The SFDDHT requires the processing of all jobs $j \in N$ in the time interval $[r_j, \bar{d}_j]$ with processing duration p_j. Hence, *the earliest feasible processing interval* for a job $j \in N$ is

$$[S^e_j, C^e_j]$$

with $S^e_j = r_j$ and $C^e_j = r_j + p_j$. Analogously, the *latest processing interval* is given by

$$[S^l_j, C^l_j]$$

with $S^l_j = \bar{d}_j - p_j$ and $C^l_j = \bar{d}_j$. Property 5.4.1.1 by Bachtenkirch and Bock (2022) formulates the precedence constraints between two jobs i and j.

Property 5.4.1.1. Given two jobs i and j, with $i \neq j$ and $i, j \in N$, if $C_j^e > S_i^l$ holds, then job j is scheduled after job i in any feasible schedule.

Proof. Property 5.4.1.1 states that job j must be processed after job i (hence, job i must be processed before job j) in any feasible schedule if job j cannot finish processing before the latest processing start time for job i occurs. According to this property,

$$C_j^e > S_i^l \Leftrightarrow r_j + p_j > \bar{d}_i - p_i \Leftrightarrow r_j + p_j + p_i > \bar{d}_i$$

holds. Assume that there exists a feasible schedule ζ' with job j processed before job i: The earliest completion time of job j is $C_j^e = r_j + p_j$; therefore, the earliest attainable completion time for job i processed after job j is $C_i' = r_j + p_j + p_i$. For this schedule, it must hold that $r_j + p_j + p_i \leqslant \bar{d}_i$, but the initial assumption was that $r_j + p_j + p_i > \bar{d}_i$ holds. This contradicts the initial assumption, thereby proving that job j must be scheduled after job i in a feasible schedule. □

The SFDDHT-B&B implementation computes the comparison stated by Property 5.4.1.1 for each pair of non-identical jobs before executing the branching process and stores the comparison results in a symmetric precedence matrix M^{prec} with n^N rows and n^N columns. For each row $i \in \{1, 2, \ldots, n^N\}$ and column $j \in \{1, 2, \ldots, n^N\}$, the matrix stores a binary value $m_{i,j}^{prec}$. The value of $m_{i,j}^{prec}$ is set to 1 if job i needs to be scheduled before job j, according to Property 5.4.1.1, and it is set to 0 otherwise. Afterward, the branching process of the B&B approach determines the extensions of a partial schedule ζ by a remaining job $j \in \bar{J}$. With the help of the precomputed precedence matrix M^{prec}, the approach tests whether job j is a suitable candidate to occupy position $k - 1$ in schedule ζ by testing whether $m_{j,i}^{prec}$ is set to a value 1 or 0 for each job $i \in \bar{J} \backslash \{j\}$. If $m_{j,i}^{prec} = 1$, then the currently considered job j cannot be placed at position $k - 1$, since it must be scheduled before job i. Note that checking whether some already scheduled job violates any precedence constraint is not necessary, since each scheduled job $j \in J$ passed this test previously; therefore, it holds that $m_{\pi(i),h}^{prec} = 0$ for any $h \in \bar{J} \cup \{\pi(k), \ldots, \pi(i-1)\}$.

5.4.2 Position Constraints

The second type of proposed constraint forbids placing a job j at a position p in any feasible schedule. In the SFDDHT, a feasible schedule processes each job in its time window as stated in Section 5.4.1. Therefore, if a job j is placed at position p, the remaining jobs $N - \{j\}$ must be scheduled at the other positions $1, \ldots, p - 1$ and $p + 1, \ldots, n^N$ while adhering to the time-window constraints. The general idea of the following approach is to derive a schedule of jobs at the earlier positions $1, \ldots, p - 1$ and the later positions $p + 1, \ldots, n^N$, and subsequently to test the feasibility of scheduling job j in-between both partial sub-schedules. Unfortunately, even the generation of a feasible schedule for a single machine problem with release and deadlines ($1|r_j, \bar{d}_j|-$) is an $\mathcal{NP}$**-complete** problem, according to Garey and Johnson (1979) (this result was already highlighted in Section 4.4 but is restated for the sake of the argument). The existence of an efficient algorithm that builds a feasible schedule with job j at the p-th position or is able to prove that no such schedule exists is thus unlikely. Instead, the following procedure utilizes Property 5.4.2.1 by Bachtenkirch and Bock (2022):

Property 5.4.2.1. Given an instance of the SFDDHT, the following two properties hold regarding scheduling a job $j \in N$ at a position $p \in N$ in a feasible schedule:

(i) A job $j \in N$ can only be scheduled at position p if a subset of $p-1$ jobs of the remaining jobs $N_j^* = N \setminus \{j\}$ have been assigned to **preceding** positions $1, \ldots, p-1$ such that job j can be scheduled at position p with $r_j + p_j \leqslant C_j \leqslant \bar{d}_j$ in this partial schedule.

(ii) A job $j \in N$ can only be scheduled at position p if a subset of $n^N - p$ remaining jobs $N_j^* = N \setminus \{j\}$ have been assigned to **succeeding** positions $p+1, \ldots, n^N$ such that job j can be scheduled at position p with $r_j + p_j \leqslant C_j \leqslant \bar{d}_j$ in this partial schedule.

Proof. The feasibility condition $r_j + p_j \leqslant C_j \leqslant \bar{d}_j$ follows directly from the problem definition. The first property (i) suggests selecting a subset of size $p-1$ of jobs of $N \setminus \{j\}$ scheduled at positions $1, \ldots, p-1$ and scheduling job j subsequently. If such a feasible partial schedule does not exist, then even the most favorable subset of the remaining jobs (with a minimal makespan) is not able to let job j occupy position p, as job j must be processed beyond its deadline. Hence, a negative result excludes positioning job j at position p; job j must be placed at a lower indexed position. The same argument applies for property (ii), but in this instance, a subset of jobs is placed at the last $n^N - p$ positions. A negative result indicates that job j must start processing before its release date to avoid conflicting with the jobs processed at positions $p+1, \ldots, n^N$. Therefore, job j must be placed at a position with a higher index than p. □

To demonstrate that p is a valid position, let

$$\Pi^f(j,p) = (\pi^f(1), \pi^f(2), \cdots, \pi^f(p-1), j)$$

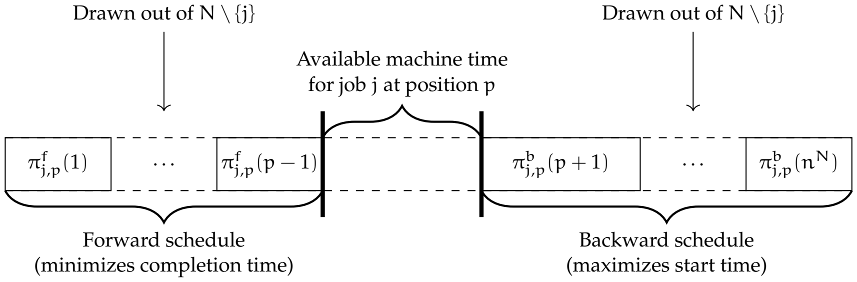
be the *front production sequence*, and

$$\Pi^b(j,p) = (j, \pi^b(p+1), \pi^b(p+2), \cdots, \pi^b(n^N))$$

the *back production sequence*, with job j scheduled at position p such that feasible partial schedules exist for both sequences. In contrast, if one of the two sequences does not produce a valid schedule, then the assignment of job j to position p is invalid. Figure 5.11 illustrates the general concept of the following approach. In the case of the front production sequence $\Pi^f(j,p)$, the feasibility of scheduling job j depends on the completion time $C_{\pi^f(p-1)}$ of the immediate predecessor of job j. As already explained, finding a feasible schedule is an $\mathcal{NP}$**-complete** problem. Hence, to establish positional constraints, the developed procedure computes a lower bound for the completion time of job $\pi^f(p-1)$ and, with this information, a lower bound for the completion time of job j with $p-1$ jobs scheduled before job j. In the case that this bound value defines an infeasible completion time for job j, the position assignment can be excluded on this account. Similarly, the start time of the immediate successor of job j, which is $S_{\pi^f(p+1)}$, determines whether job j can

Figure 5.11

Precomputation of Eligible Positions



Note. A job $j \in N$ at position p must be scheduled between the forward and backward schedules in any feasible schedule ζ. Reprinted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier.

be processed inside its time window, or if the release date constraint for job j is violated. In this case, the upper bound to the completion time of j decides on the feasibility of the assignment to position p. To generate the bound values, the time window constraints are relaxed in both cases. Next, Section 5.4.2.1 describes the bound computation for a schedule at positions $1, \ldots, p-1$. Thereafter, Section 5.4.2.2 discusses the case for the succeeding positions $p+1, \ldots, n^N$.

5.4.2.1 Computation of the Front Schedule

The approach that schedules jobs N^*_j at the first $p-1$ positions relaxes the deadline constraints of the jobs. The objective is thus to find a schedule for a subset of jobs of size $p-1$ that minimizes the maximal completion time (makespan). In general, this scheduling problem can be denoted as $1 \mid r_j, n' \leqslant n \mid C_{\max}$, (i.e., the scheduling problem of finding a selection of $n' \leqslant n$ jobs that minimizes the makespan $C_{\max}$ subject to release date constraints).

The classic scheduling problem, without selecting a subset of jobs $1 \mid r_j \mid C_{\max}$, is polynomially solvable by applying the earliest release date (ERD) rule, as shown by Lawler (1973). The ERD rule defines a sequence of jobs ordered by non-decreasing release date. This rule is ambiguous, since the order of two jobs i and j with identical release date $r_i = r_j$ is left undefined. To resolve this ambiguity, Definition 5.4.2.2 defines the precedence relation $i \prec_{\text{ERD}} j$, which provides second and third ordering criteria to establish a single ERD sequence for a given set of jobs.

Definition 5.4.2.2. In an ERD schedule, the precedence relation $i \prec_{\text{ERD}} j$ holds for each job i scheduled before job j :

$$i \prec_{\mathrm{ERD}} j \text{ holds if and only if : } \begin{cases} r_i < r_j & \vee \\ r_i = r_j,\ p_i < p_j & \vee \\ r_i = r_j,\ p_i = p_j,\ i < j. \end{cases} \tag{5.8}$$

Algorithm 5.4 presents a *list scheduling algorithm* that generates a schedule in ERD order, as established by Definition 5.4.2.2. A list scheduling algorithm assigns jobs to a machine in order of an initially generated list. The order does not change, because of the information provided by an intermediate schedule. Due to the required minimization of $C_{\max}$, jobs are scheduled as early as possible without generating unnecessary idle time on the machine. The algorithm performs in $\mathcal{O}(n \log(n))$ time due to the initial sorting operation.

Algorithm 5.4 ERD list

```
Require: A list of indexes of jobs $I = [i_1, i_2, \cdots i_n]$ in ERD order, such that
  $i_1 \prec_{ERD} i_2 \prec_{ERD} \ldots \prec_{ERD} i_n$;
  procedure ERD-LIST(I)
      $C_{i_1} := r_{i_1} + p_{i_1}$;
      for $i_j \in \{i_2, i_3, \cdots, i_n\}$ do
          $C_{i_j} := \max\{C_{i_{j-1}}, r_{i_j}\} + p_{i_j}$.
      return $C_{i_n}$;
```

An application of Algorithm 5.4 does not necessarily result in an optimal schedule for the selection problem $1|r_j, n' \leqslant n|C_{\max}$, as proven by Example 5.4.2.3.

Example 5.4.2.3. Consider an instance with $n = 3$ jobs with the following data:

Job (j)	1	2	3
Release date (r_j)	0	2	4
Processing time (p_j)	2	4	1

The application of the ERD rule results in optimal sequence $(1, 2, 3)$ for problem $1|r_j|C_{max}$ with the following completion times:

Job (j)	1	2	3
Completion time (C_j)	2	**6**	7

Consider the problem $1|r_j, n' \leqslant n|C_{max}$ with $n' = 2$. The ERD rule yields $C_{\max} = C_2 = 6$, as shown by the above schedule. However, the sequence $(1 - 3)$ results in completion times $C_1 = 2$ and $C_{\max} = C_3 = 5$. This proves that the ERD rule does not optimally solve $1|r_j, n' \leqslant n|C_{\max}$.

Instead, we now describe a specifically developed DP algorithm that solves $1|r_j, n' \leqslant n|C_{\max}$. The approach is based on Lemma 5.4.2.1 by Bachtenkirch and Bock (2022):

Lemma 5.4.2.1. *An optimal schedule exists for problem* $1|r_j, n' \leqslant n|C_{max}$ *where the* n' *selected jobs are scheduled in ERD order.*

Proof. The ERD rule solves problem $1|r_j|C_{max}$ optimally (Lawler, 1973). The problem of scheduling a selection of n' jobs of the original jobs $1, 2, \ldots, n$ reduces to an instance of $1|r_j, |C_{max}$ with the n' selected jobs. □

The following description assumes, without loss of generality, that jobs are indexed in the unambiguous ERD order defined by Definition 5.4.2.2. That is,

$$\forall i, j \in N\colon i \prec_{\text{ERD}} j \iff i < j$$

holds. Consider a procedure that decides on selecting (or rejecting) a job $j \in N$ for integration into the schedule in ERD order. If a job j is selected for integration, then it appears at the intermediate schedule's last position, since all jobs i with $i < j$ are already selected or rejected. The DP algorithm that solves $1|r_j, n' \leqslant n|C_{\max}$ utilizes the fixed relative order of jobs in an ERD schedule. Each state during the execution of the algorithm is characterized by its function value $f(j, l)$ which is the minimal makespan of scheduling l jobs of the set $\{1, 2, \dots, j\}$. Informally, a state's function value results by selecting one of two policies: Either job j is not scheduled in a schedule with $l \leqslant n'$ jobs, or job j extends the best schedule with $l - 1 < n'$ jobs at the last position, which increases the makespan by the earliest completion time that job j attains with the $l - 1$ other jobs scheduled before job j. Algorithm 5.5 by Bachtenkirch and Bock (2022) describes the DP formulation. Furthermore, Lemma 5.4.2.2 by Bachtenkirch and Bock (2022) claims that the above algorithm optimally solves the selection problem and states the DPs asymptotic time complexity.

Algorithm 5.5 Dynamic Programming Algorithm for $1|r_j, n' \leqslant n|C_{max}$

Dynamic programming algorithm for $1|r_j, n' \leqslant n|C_{max}$

Initial conditions:

$$f(l, 0) = 0 \quad \forall l = 0, 1, ...n,$$

$$f(1, 1) = r_1 + p_1,$$

$$f(j, l) = \infty \quad \forall j < l.$$

Recursive computation:

$$f(j, l) = \min\{f(j-1, l), \max\{f(j-1, l-1), r_j\} + p_j\}$$

$$\text{with } j \in N \text{ and } 1 \leqslant l \leqslant j.$$

Optimal value (i.e., output value):

$$f(n, n').$$

Lemma 5.4.2.2. *Algorithm 5.5 optimally solves the scheduling problem* $1|r_j, n' \leqslant n|C_{max}$ *in time* $\mathcal{O}(\max\{\log(n), n'\} \cdot n)$.

Proof. First, consider the correctness of the initial conditions and the recursive computation. The return value (=makespan) of $f(l, 0)$ for each $l \in \{0, 1, \dots, n\}$ is set to *zero* because no job is scheduled for $l = 0$. The return value of $f(1, 1)$ is set to $r_1 + p_1$, since the state $(1, 1)$ describes the situation where only job 1 is scheduled on the machine. States (j, l) with $j < l$ are infeasible and set to ∞, since no schedule of size l exists with less than l jobs scheduled.

Second, consider the value function $f(j, l)$ with $j \in N$ and $1 \leqslant l \leqslant j$ of state (j, l). Such a state decides whether the cost optimal decision of scheduling l jobs from set $\{1, 2, \cdots, j\}$

is attainable by selecting or rejecting job j. Due to the indexing of jobs in ERD order, the selection of job j implies that it is scheduled at the last position and therefore completes at time $\max\{r_j, f(j-1, l-1)\} + p_j$. In a case where the rejection of job j is cost optimal at state (j, l), then the makespan is identical to the one provided by $f(j-1, l)$. Since $f(j-1, l-1)$ and $f(j-1, l)$ are optimal, either selecting or rejecting job j leads to an optimal schedule for state (j, l). To execute the DP algorithm, n jobs are sorted first, with asymptotic time complexity $\mathcal{O}(\log(n) \cdot n)$. The DP algorithm requires $n \cdot n'$ states at most; hence, the total asymptotic time complexity of the algorithm is $\mathcal{O}(\max\{\log(n), n'\} \cdot n)$. □

With Algorithm 5.5 defined, the discussion returns to the description of the preprocessing procedure for the SFDDHT. For each job $j \in N$ and position $p \in N$, the algorithm computes the minimal makespan of assigning $p-1$ jobs of set N_j^* to the first $1, \dots, p-1$ positions, with $C_{N_j^*,p}^{f,min}$ denoting the resulting makespan value. Note that a single call of Algorithm 5.5 with job set N_j^* and $n = n' = n^N - 1$ computes the minimal makespan values for each $n' \in \{0, 1, \cdots, n\}$. It thus suffices to call the algorithm n^N times, once for each job $j \in N$. Equation (5.9) estimates the lower bound value for the front schedule with job $j \in N$ at the p-th position as follows:

$$C_{j,p}^{min} = \max\{C^{f,min}(N_j^*, p), r_j\} + p_j \quad \forall j, p \in N. \tag{5.9}$$

5.4.2.2 Computation of the Back Schedule

Similarly to the computation of the front schedule, Algorithm 5.5 computes the back schedule at positions $p+1, \dots, n^N$. Instead of scheduling jobs as early as possible, the jobs must be processed as late as possible at the back positions. Therefore, rather than relaxing the deadlines of jobs, the approach relaxes the release dates of jobs such that each job is solely defined by its deadline and processing time. To apply Algorithm 5.5, an initial procedure transforms the relaxed instances with deadlines and processing times into an equivalent instance with release dates and processing times. Specifically, given jobs N of the SFDDHT instance, the deadlines are transformed into *equivalent release dates* r_j^b as follows:

$$\forall j \in N\colon r_j^b = \bar{d}^{max} - \bar{d}_j$$

where $\bar{d}^{max} = \max_{j \in N}\{\bar{d}_j\}$ is the maximal deadline.

Following this transformation, Algorithm 5.5 is called once with jobs N_j^*, processing times p_j, and equivalent release dates r_j^b and $n' = n^N$ for each job $j \in N$, with $C_{N_j^*,p}^{b,min}$ denoting the resulting makespan values. To state an upper bound value for the latest completion time of a job j scheduled at position p with an assignment of a selection of the jobs N_j^* to positions $p+1, \dots, n^N$, the approach corrects values $C^{b,min}(N_j^*, p)$ by subtracting the values from $\bar{d}^{max}$. This operation yields the latest start time of a back schedule that comprises a selection of jobs N_j^*. Equation (5.10) is consequently an upper bound for the completion times of a job j scheduled at position p:

$$C_{j,p}^{max} = \min\{\bar{d}^{max} - C^{b,min}(N_j^*, p), \bar{d}_j\} \quad \forall j, p \in N. \tag{5.10}$$

5.4.2.3 Preprocessing and Application in the Branching Process

The preprocessing procedure computes the completion time bounds $C_{j,p}^{\min}$ and $C_{j,p}^{\max}$ for each pair $j, p \in N$. Similar to the implementation of the precedence matrix M^{prec}, the procedure constructs a position binary matrix M^{pos} with $n^N \times n^N$ rows and columns. The entry $m_{i,j}^{pos}$ of the i-th row and j-th column is set to 1 if position j is allowed for job i, and it is set to 0 otherwise. Equation (5.11) decides the binary value for each entry:

$$m_{i,j}^{pos} = \begin{cases} 1 & \text{if } C_{i,j}^{\min} \leqslant \bar{d}_j \wedge C_{i,j}^{\max} \geqslant r_j + p_j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.11}$$

The branching procedure eliminates each potential branch that schedules a job j at an excluded position p. That is, given schedule ζ, the job $j \in \bar{J}$ does not extend schedule ζ at position $k-1$ if $m_{j,k-1}^{pos} = 0$.

5.5 Dynamic Feasibility Testing

As explained, the preprocessing procedures construct the preprocessing tables M^{prec} and M^{pos} before starting the enumeration of the B&B algorithm, and during the enumeration, the algorithm simply reads the values of the stored matrices. This technique allows for fast computable decisions for allowing or excluding branching decisions. However, the computation uses no knowledge gained by considering a currently active schedule ζ to exclude branching decisions. Therefore, this section proposes a procedure that utilizes dynamic knowledge at the expense of additional computational effort to further eliminate branching decisions that were not excluded by the preprocessing results. The dynamic procedure utilizes knowledge about the currently explored schedule ζ that already fixed the decisions for jobs J scheduled at positions $k, \dots, n^N$ and the scheduling of the remaining jobs $\bar{J}$.

Suppose that the B&B considers a non-dominated node with evaluated schedule ζ for branching. The branching procedure determines whether a job $j \in \bar{J}$ can be scheduled directly before the already scheduled jobs J at position $k-1$. Hence, for a candidate $j \in \bar{J}$ to be placed at position $k-1$, the *other remaining jobs* $\bar{J}_j^* = \bar{J} \setminus \{j\}$ must be scheduled at positions $1, \dots, k-2$ if the set $\bar{J}_j^*$ is nonempty. The start time of the currently first scheduled job in schedule ζ is $S^1 = S_{\pi(k)}$. Therefore, the latest time point at which a candidate job j can be feasibly scheduled in any schedule that extends ζ is

$$\forall j \in \bar{J}\colon \bar{C}_j^{\max}(\zeta) = \min\{\bar{d}_j, S^1\}.$$

Similar to the methods applied in the preprocessing phase, the subsequently discussed procedure determines whether placing job j at position $k-1$ can be identified as an infeasible extension of schedule ζ. Again, the applied procedure computes only a lower bound on the makespan of the remaining jobs at positions $1, \dots, k-2$ to allow or disallow job j at position $k-1$. Fortunately, in the dynamic case, the set of scheduled jobs before j is fixed. Therefore, a more restrictive relaxation can be utilized, compared to the one applied in Section 5.4.2 to compute a lower bound value. Additionally, the underlying scheduling procedure enables the detection of infeasible scheduling of the jobs $\bar{J}_j^*$.

The applied procedure relaxes the feasibility problem of scheduling jobs by allowing preemptive processing. More precisely, it solves the scheduling problem $1|r_j, pmnt|T_{\max}$ that pursues the minimization of maximum tardiness for a given set of jobs with release dates and due dates (i.e., soft deadlines), while enabling the preemption of jobs. Preemption allows for the jobs to be partially processed and for their processing to be resumed at a later time on the machine. This preemptive scheduling problem is polynomially solvable (in contrast to its non-preemptive counterpart) through the approach described by Baker and Su (1974) that schedules the ready job with $r_j \geqslant t$ at each time-point t with the minimal due date. As the machine does not delay the processing of ready jobs, the makespan is minimized by this dispatching rule.

Algorithm 5.6 describes the SFDDHT-B&B implementation of the approach by Baker and Su (1974) for a set of jobs with indexes $N' = \{j_1, j_2, \ldots, j_{n'}\}$ labeled in order of non-decreasing release dates such that $r_{j_1} \leqslant r_{j_2} \leqslant \cdots \leqslant r_{j_{n'}}$ holds. Ties between jobs with identical release date values are broken (a) in order of nondecreasing deadline and (b) in order of job index. Next, the remaining processing time variables for all jobs are denoted as p'_j for each $j \in N'$ and initially set to $p'_j = p_j$.

The algorithm returns the minimal makespan if no remaining processing time for any job is left. Otherwise, if a deadline violation is detected during the execution, then the algorithm terminates early and returns ∞. The implementation requires an initial sort to obtain the described ordering of jobs by non-decreasing release dates, which is an $\mathcal{O}(n' \log(n'))$ operation. The set S initially comprises the reversely ordered job indexes such that access and deletion at the back of the index list are constant operations. The set of ready job indexes R is implemented as a heap data structure that stores the ready job with the earliest deadline at the top. In general, this data structure has $\mathcal{O}(\log(n))$ element insertion, $\mathcal{O}(1)$ top-element access, and $\mathcal{O}(\log(n))$ top-element deletion (with subsequent restoration of the heap property) performance for n elements. In the stated algorithm, $n' - 1$ preemptions (for each newly released job after the first processed one) occur at most; hence, the number of iterations that the algorithm performs is linear in terms of the number of jobs. The overall asymptotic worst time complexity of the algorithm is $\mathcal{O}(n' \ \log(n'))$.

The return value of Algorithm 5.4 (i.e., the minimal makespan) is denoted as $\bar{C}^{\min}(\bar{J}^*(\zeta))$ for a particular set of remaining jobs $\bar{J}^*(\zeta)$. The minimal makespan for job j under these conditions is then given as

$$\bar{C}_j^{\max}(\zeta) = \max\{r_j, \bar{C}^{\min}(\bar{J}^*(\zeta))\} + p_j$$

Given schedule ζ, job $j \in \bar{J}$ can be scheduled at position $k - 1$ if the following holds:

$$\bar{C}_j^{\min}(\zeta) \leqslant \bar{C}_j^{\max}(\zeta).$$

Algorithm 5.6 Implementation of the Algorithm Proposed by Baker and Su, Adapted to Solve Problem $1|r_j, \bar{d}_j|C_{\max}$

```
Require: Jobs $N' = \{j_1, j_2, \dots, j_{n'}\}$ ordered such that $r_{j_1} \leqslant r_{j_2} \leqslant \cdots \leqslant r_{j_{n'}}$.
$t \leftarrow 0$                                        ▷ Current time
$S \leftarrow \{n', n'-1, \dots, 1\}$                  ▷ List of job indexes in reverse ERD order
$R \leftarrow \varnothing$                  ▷ Heap of ready jobs indexes (top element has earliest deadline)
while $S \neq \varnothing$ and $R \neq \varnothing$ do
    if $R = \varnothing$ then
        $t = r_{j_{\mathrm{back}(S)}}$
        while $r_{j_{\mathrm{back}(S)}} == t$ do
            $\mathrm{insert}(R, j_{\mathrm{back}(S)})$
            $\mathrm{pop\_back}(S)$
    $j \leftarrow \mathrm{top}(R)$
    $c' \leftarrow t + p_{j'}$
    if $c' > \bar{d}_{j'}$ then
        return $\infty$                                ▷ Deadline violation detected
    while $r_{j_{\mathrm{back}(S)}} < c'$ do
        $k \leftarrow \mathrm{back}(S)$
        $\mathrm{insert}(R, j_k)$
        $\mathrm{pop\_back}(S)$
        if $\bar{d}_{j_k} \leqslant \bar{d}_{j'}$ then
            $c' \leftarrow r_{j_k}$
            break
    if $\mathrm{top}(R) = j'$ then
        $\mathrm{delete\_top}(R)$
    else
        $p'_{j'} \leftarrow p'_{j'} - c' + t$
    $t \leftarrow c'$
return $t$                                              ▷ Minimal makespan
```

5.6 Branching Strategy

This section describes the final branching strategy of the SFDDHT-B&B algorithm, which utilizes the findings from Section 5.4 and Section 5.5 to restrict the set of jobs that can extend a schedule ζ. Furthermore, this section proposes a reduction of transports that must be considered for feasibility and optimality reasons. The combination of both phases significantly reduces the number of subproblems that must be considered in the branching process. Given a subproblem $\mathcal{X}_i^c$ defining schedule ζ, the following explanations unambiguously define the set of smaller subproblems generated from $\mathcal{X}_i^c$ in the branching process by defining all non-dominated job-time pairs that extend schedule ζ at position $k-1$.

The following *initial branching decisions* may extend schedule ζ at position $k-1$, without applying any feasibility considerations:

$$B^0(\zeta) = \left\{(j, t) \mid j \in \bar{J}, t \in T_{g^N(j)}\right\}. \tag{5.12}$$

Therefore, in non-restrictive cases, the process considers $\sum_{g \in G} |\bar{J}_g| \times n_g^T$ different subproblems, assuming that all transports are still assignable by unscheduled jobs.

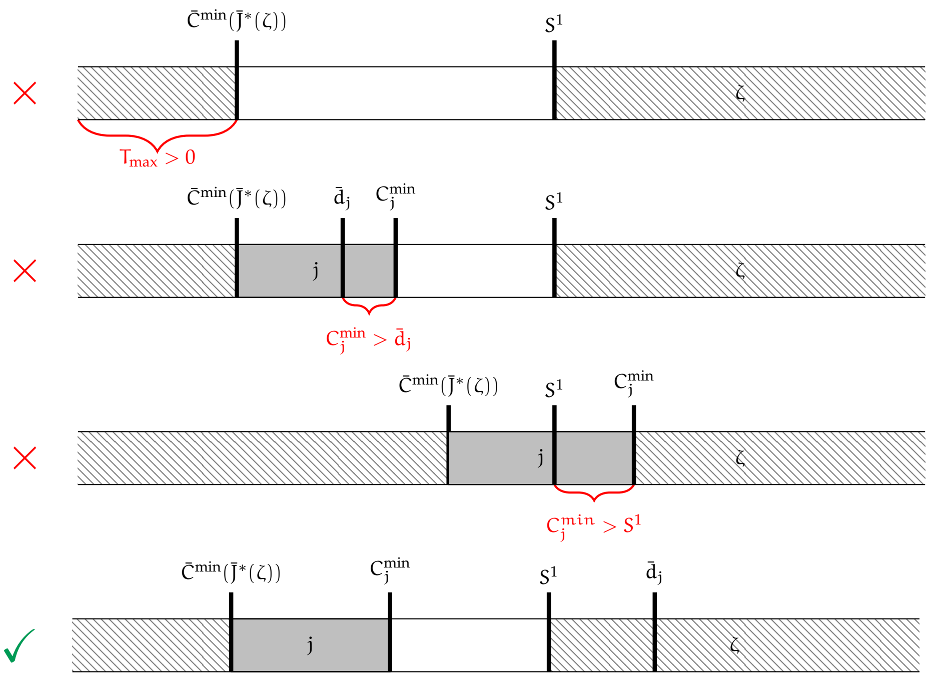
The previously described preprocessing procedures and the dynamic feasibility test reduce this number and define the *set of branching candidate jobs* $\bar{J}^b(\zeta) \subseteq \bar{J}(\zeta)$, which can not be excluded and therefore must be considered for branching. Equation (5.13) defines this set as follows:

$$\bar{J}^b(\zeta) = \left\{ j \in \bar{J}(\zeta) \;\middle|\; \begin{array}{r} \sum_{i \in \bar{J}^*_j(\zeta)} m^{prec}_{j,i} = 0, \\ m^{pos}_{j,k-1} = 1, \\ \bar{C}^{min}_j(\zeta) \leqslant \bar{C}^{max}_j(\zeta) \end{array} \right\}. \tag{5.13}$$

Figure 5.12 illustrates cases in which job j is included or excluded in the set of branching candidate jobs, based on the third condition. In the first case, the set of remaining jobs without j is not feasibly scheduled; in the second case, the addition of job j after scheduling the other remaining jobs violates the deadline constraint of job j; and in the third case, job j can not be completed before production of the already scheduled jobs begins.

Figure 5.12

Different Cases of Evaluation Jobs for Branching



Note. Symbols on the left indicate that job j is included ✓ (discarded ×) in the branching candidate set $\bar{J}^B$ for extending schedule ζ at position $k-1$.

After defining set $\bar{J}^b$, the branching procedure obtains *viable transports* for each job $j \in \bar{J}^b(\zeta)$. Let $D_{j,\tau}(\zeta) \subseteq T_{g^N(j)}$ denote the set of *non-dominated delivery departure times* assignable to job j completing at time τ given schedule ζ. The branching process sets τ to $\bar{C}^{min}_j(\zeta)$, which is a valid lower bound for the completion time C_j and also for the delivery time D_j of a job $j \in \bar{J}$. Hence, no date before time $\bar{C}_j^{\min}(\zeta)$ has to be considered to define set $D_{j,\tau}(\zeta)$.

Next, we introduce three subsets of departure times that form set $D_{j,\tau}(\zeta)$ by unification. For notational purposes, Equation (5.14) defines the set of *feasible delivery departure times of job* j solely based on the job's time window:

$$\forall j \in N : T_j^N = \left\{ t \mid t \in T_{g^N(j)}, r_j + p_j \leqslant t \leqslant \bar{d}_j \right\}. \tag{5.14}$$

The first subset $D_{j,\tau}^1(\zeta)$ – Equation (5.15) – includes all feasible delivery departure times of job j *earlier than* the starting time S^1 of schedule ζ:

$$D_{j,\tau}^1(\zeta) = \left\{ t \mid t \in T_j^N, \tau \leqslant t < S^1 \right\}. \tag{5.15}$$

The second subset $D_{j,\tau}^2(\zeta)$ – Equation (5.16) – is a singleton set that comprises the *first unused transport not earlier* than the schedule start time S^1 and is defined as follows:

$$D_{j,\tau}^2(\zeta) = \begin{cases} d_{j,\tau}^2(\zeta) = \min \left\{ t \mid t \in T_j'^N \right\} & \text{if } T_j'^N = \left\{ t \mid t \in T_j^N, S^1 \leqslant t < t_g^{\min} \right\} \neq \varnothing, \\ \varnothing & \text{otherwise.} \end{cases} \tag{5.16}$$

Lastly, Equation (5.17) considers the first used transport at time $t_g^{\min}$. If schedule ζ defines such a transport for customer $g = g^N(j)$ it is the sole element of the third singleton subset:

$$D_{j,\tau}^3(\zeta) = \begin{cases} \{t_g^{\min}\} & \text{if } t_g^{\min} \in T_j^N, \\ \varnothing & \text{otherwise.} \end{cases} \tag{5.17}$$

Equation (5.18) unifies the three subsets to form set $D_{j,\tau}(\zeta)$:

$$D_{j,\tau}(\zeta) = D_{j,\tau}^1(\zeta) \cup D_{j,\tau}^2(\zeta) \cup D_{j,\tau}^3(\zeta). \tag{5.18}$$

Lemma 5.6.1 by Bachtenkirch and Bock (2022) states that $D_{j,\tau}(\zeta)$ is exhaustive.

Lemma 5.6.1. *Given a partial schedule ζ with job $j \in \bar{J}$, no delivery departure time other than the ones in the unified set $D_{j,\tau}(\zeta) = D_{j,\tau}^1(\zeta) \cup D_{j,\tau}^2(\zeta) \cup D_{j,\tau}^3(\zeta)$ need to be considered in the branching process for job j at position $k-1$ for schedule ζ.*

Proof. The set T_j^N defines all feasible delivery departure times for a job j, which follows from the problem definition. No delivery departure time $t \in T_j^N$ with $t < \tau = \bar{C}_j^{\min}$ has to be included in set $D_{j,\tau}(\zeta)$, since $\bar{C}_j^{\min}$ is a lower bound of the completion time of j, and inequality $D_j \geqslant C_j$ holds from the problem definition. All delivery departure times $t \in T_j^N$ with $t < S^1$ are included in set $D_{j,\tau}^1(\zeta)$. Therefore, $D_{j,\tau}(\zeta)$ includes all feasible transports up to time $S^1 - 1$.

The remaining delivery departure times to consider are the transports that range from time S^1 up to the jobs deadline $\bar{d}_j$, which marks the latest feasible delivery departure time for job j. The set $D_{j,\tau}^2(\zeta)$ comprises at most a single delivery departure time that is defined for the earliest unused transport to customer $g^N(j)$. This delivery departure time minimizes the holding costs for job j subject to the maximization of the completion time of j in a canonical schedule. Departures at any other delivery departure times later than $d^2_{j,\tau}(\zeta)$ result in higher holding costs for job j, while resulting in an equal transportation cost increase of $c^T_{g^N(j)}$. The later unused transports can thus be dismissed for optimality reasons. Lastly, if the earliest already used transport at $t^{min}_{g^N(j)}$ exists, it is included, and all later, already used transports can be excluded due to Property 5.2.5.2. In conclusion, no delivery departure time that is not included in set $D_j(\zeta)$ needs to be considered. □

In the following lemma, this set may further be reduced by *one* transport. In some circumstances, the delivery departure $t^{min}_{g^N(j)}$ from set $D_{j,\tau}(\zeta)$ can be excluded if $d^2_{j,\tau}(\zeta)$ is dominant due to costs; that is, if the saved transportation costs of an assignment to $t^{min}_{g^N(j)}$ do not make up for an increase in holding cost compared to an assignment to earlier date $d^2_{j,\tau}(\zeta)$, then an extended schedule with j assigned to $t^{min}_{g^N(j)}$ does not need to be considered. Lemma 5.6.2 by Bachtenkirch and Bock (2022) formalizes this finding.

Lemma 5.6.2. *Given partial solution ζ and a job $j \in \bar{J}$ with nonempty sets $D^2_{j,\tau}$ and $D^3_{j,\tau}$, an assignment of job j to $d^2_j(\zeta)$ dominates an assignment to $t^{min}_{g^N(j)}$ if*

$$\left(t^{min}_{g^N(j)} - d^2_{j,\tau}(\zeta)\right) \cdot c^H_j \geqslant c^T_{g^N(j)}$$

holds. It follows that an extended schedule of ζ with job j assigned to departure time $d^2_{j,\tau}(\zeta)$ dominates an extended schedule of ζ with job j assigned to departure time $t^{min}_{g^N(j)}$.

Proof. The minimal cost of assigning job j to time $d^2_{j,\tau}(\zeta)$ is $c^1 = (d^2_{j,\tau}(\zeta) - S^1) \cdot c^H_1 + c^T_{g^N(j)}$, and the minimal cost of assigning job j to $t^{min}_{g^N(j)}$ is $c^2 = (t^{min}_{g^N(j)} - S^1) \cdot c^H_j$, which can be written as

$$c^2 = (t^{min}_{g^N(j)} - d^2_{j,\tau}(\zeta)) \cdot c^H_j + (d^2_{j,\tau}(\zeta) - S^1) \cdot c^H_j.$$

By assumption, $(t^{min}_{g^N(j)} - d^2_{j,\tau}(\zeta)) \cdot c^H_j \geqslant c^T_{g^N(j)}$. Therefore,

$$c^2 \geqslant c^T_{g^N(j)} + (d^2_{j,\tau}(\zeta) - S^1) \cdot c^H_j \equiv c^2 \geqslant c^1$$

holds. Additionally, other jobs of customer $g^N(j)$ assignable to the transport at time $d^2_{j,\tau}(\zeta)$ may benefit from this earlier transport. □

The construction of viable transports is illustrated by Figure 5.13. The application of Lemma 5.6.2 may exclude one element of $D_{j,\tau}(\zeta)$ in the branching process. Therefore, $D^*_{j,\tau}(\zeta)$ – Equation (5.19) – defines the viable transports for each branching candidate $j \in \bar{J}^B(\zeta)$.

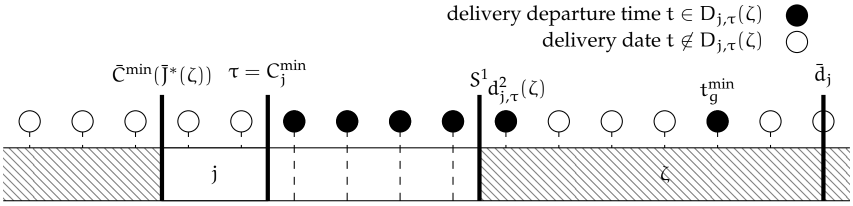
$$D^*_{j,\tau}(\zeta) = \begin{cases} D_{j,\tau}(\zeta) \setminus D^3_{j,\tau}(\zeta) & \text{if Lemma 5.6.2 holds,} \\ D_{j,\tau}(\zeta) & \text{otherwise.} \end{cases} \tag{5.19}$$

The branching carried out during the branching procedure for a node with schedule ζ is thus defined by the following set:

$$B(\zeta) = \{(j,t) \mid j \in \bar{J}^b(\zeta), t \in D^*_{j,\bar{C}^{min}_j(\zeta)}(\zeta)\}.$$

Figure 5.13

Considered Delivery Departure Times for a Candidate Job During Branching


delivery departure time $t \in D_{j,\tau}(\zeta)$ ●
delivery date $t \notin D_{j,\tau}(\zeta)$ ○
$\bar{C}^{min}(\bar{J}^*(\zeta))$
$\tau = C_j^{min}$
S^1
$d_{j,\tau}^2(\zeta)$
t_g^{min}
$\bar{d}_j$
j
ζ

Note. Each delivery departure time marked with a black circle is a decision that must be considered for branching. All delivery departure times marked with a white circle can be discarded for feasibility or optimality reasons. In the shown case six new branches must be considered.

5.7 Lower Bounds

This section describes a total of *three* alternative lower bounds for the SFDDHT that are computable for each generated partial schedule ζ during the execution of the B&B algorithm. As detailed in Section 5.1, a lower bound value for a subproblem $\mathcal{X}_i^c$ that defines a partial schedule ζ is a valid lower bound for the objective value for each derived subproblem of $\mathcal{X}_i^c$ and, in particular, each complete schedule derivable from the analyzed subproblem. The SFDDHT-B&B computes lower bounds for two reasons. First, given an upper bound of the objective value of an optimal solution to a problem instance, any subproblem can be discarded for which the computed lower bound value is not lower than the upper bound, as no better upper bound can be found following the branches from such a subproblem. Second, the lower bounds help to guide the search for promising subproblems earlier, as the SFDDHT-B&B adopts a best-first strategy.

Each lower bound for a subproblem $\mathcal{X}_i^c$ with partial schedule ζ consists of two parts: The first part is the cost value $z(\zeta)$ for the scheduled jobs J, and it is constant for all derived schedules of ζ. The second part is a cost estimate $z_i^e(\zeta, \bar{J})$ for the remaining unscheduled jobs. Note that the computation of $z_i^e(\zeta, \bar{J})$ depends on the remaining jobs $\bar{J}$ and the decisions made for the already scheduled jobs J. This cost estimate lower bounds the cost attainable by scheduling the remaining jobs $\bar{J}$. The return value for each lower bound $i \in \{1, 2, 3\}$ is given as $LB_i(\zeta) = z(\zeta) + z_i^e(\zeta, \bar{J})$, where the subscript i distinguishes between the *three* proposed lower bounds and cost estimates.

The different lower bounds adopt the common methodology of suitably relaxing problem assumptions to define instances of polynomially solvable problems (c.f. Morrison et al., 2016). An optimal solution to such an *easier* problem instance then provides a cost lower bound for scheduling the remaining jobs. In many scheduling problems, finding a suitable relaxation is straightforward. For example, the $\mathcal{NP}$**-hard** problem $1 \mid r_j \mid \sum w_j C_j$ is relaxable to the polynomially solvable problem $1 \mid\mid \sum w_j C_j$ by omitting the job release dates. An optimal solution to an equivalent instance for $1 \mid\mid \sum w_j C_j$ provides a lower bound value for the problem with release dates, since having job release dates may only increase the individual job completion times. Note that this relaxation does not necessarily provide tight bounds.

The SFDDHT comprises significantly more constraints than the abovementioned example problem. Therefore, the subsequently presented lower bounds relax multiple problem aspects and try to preserve different components of the original SFDDHT problem to generate strong lower bound values.

5.7.1 Linear Assignment Bound

The first lower bound (LB I), namely, the linear assignment bound, maps the remaining decisions of a subproblem of the SFDDHT to an instance of the well-known (balanced) linear assignment problem (LAP). The LAP seeks a cost-minimal assignment of m^{lap} elements to m^{lap} targets (Kuhn, 1955). An assignment of the i-th element to the j-th target is expressed by setting binary decision variable $X_{i,j}^{lap}$ to 1 (and a non-assignment to 0) in the following LP formulation of the LAP:

$$\min \sum_{i \in M^{lap}} \sum_{j \in M^{lap}} c_{i,j}^{lap} X_{i,j}^{lap} \tag{5.20}$$

subject to:

$$\sum_{j \in M^{lap}} X_{i,j}^{lap} = 1 \quad \forall i \in M^{lap}, \tag{5.21}$$

$$\sum_{i \in M^{lap}} X_{i,j}^{lap} = 1 \quad \forall j \in M^{lap}, \tag{5.22}$$

$$X_{i,j}^{lap} \geqslant 0 \quad \forall (i,j) \in M^{lap} \times M^{lap}. \tag{5.23}$$

The Objective Function (5.20) minimizes the total costs of all assignments, while Constraints (5.21) and (5.22) ensure that each element is assigned to one target and that each target is assigned to an element. As recognized by Pinedo (2016), requiring the $X_{i,j}^{lap}$to be integers is not necessary – see (5.23) – since the structural properties of the LP lead to integral solutions for the problem. Note that the cost of each assignment in the LAP is independent of the other assignments. The LAP is polynomially solvable in $\mathcal{O}\left({m^{lap}}^3\right)$ time for m^{lap} targets/elements.

An instance of the LAP is specified by a cost matrix with m^{lap} elements (rows) and m^{lap} targets (columns). The proposed bound links the remaining jobs $\bar{J}$ with elements of the LAP, and time slots (processing intervals on the machine) with targets of the LAP. A time slot explicitly defines the completion time on the machine for the assigned job in a schedule. The proposed approach constructs a cost matrix in *three steps*:

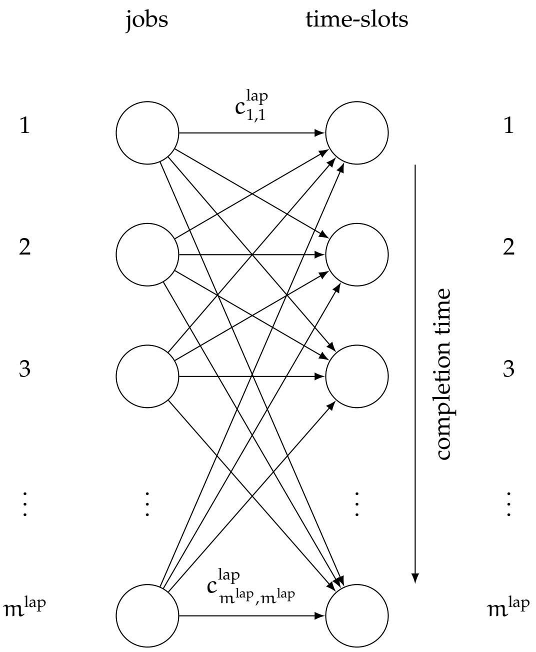
1. The approach specifies the time slots on the machine to process the remaining jobs $\bar{J}$, where each time-slot is assignable by one job at most, and each job $j \in \bar{J}$ must be assigned to a single time slot. Additionally, the introduction of additional dummy jobs balances the cost matrix of the LAP when the number of generated time slots exceeds the number of regular jobs.
2. The approach computes a cost factor for each transport that is attributed to individual jobs to account for the batch delivery transportation cost.

3. The approach calculates a lower bound for the total cost contribution for each *job to time slot assignment* that represents the minimal holding and transportation costs of completing a job j at a specific time on the machine.

The problem-specific interpretation of the LAP's cost-matrix is illustrated in Figure 5.14. Note that the problem definition requires the possibility of assigning each job to each time slot. Release date and deadline restrictions are accounted for by introducing prohibitively large cost values for infeasible assignments. An optimal LAP solution avoids such assignments if possible.

Figure 5.14

Interpretation of the LAP Cost Matrix



LAP: linear assignment problem

5.7.1.1 Construction of Time Slots

Given a partial schedule ζ, the remaining jobs $\bar{J} \neq \varnothing$ must be scheduled before the already processed jobs. The available machine time to process these jobs is denoted as *processing interval* $P = [p^e, p^l]$, which ranges from $p^e = \min\{r_j \mid r_j \in \bar{J}\}$ to $p^l = \bar{C}^{max}$. The assignable time slots are defined as processing intervals within this range. Note that for an empty schedule, $\bar{C}^{max}$ is set to the maximal deadline value of the jobs $\bar{J} = N$ (see Section 5.3.1); hence, P is properly defined.

To construct intervals that lead to a valid lower bound of the cost of scheduling jobs $\bar{J}$, a set of completion times is derived that provides a segmentation of the processing interval P. Specifically, we now define a subset of completion times that may occur in complete canonical schedules for a given partial schedule ζ. This subset is constructed by first considering the unique delivery departure times, which are the time points, in processing interval P:

$$T(P) = \{t_l \mid l \in I^T, p^e \leqslant t_l \leqslant p^l\}.$$

The delivery departure times of set $T(P)$ are denoted as $t_1^P, t_2^P, \dots, t^P_{n^{T(P)}}$ and ordered by increasing time value such that $t_1^P < t_2^P < \dots < t^P_{n^{T(P)}}$ holds. The defined time points separate the processing horizon P into disjoint time intervals, as specified by Definition 5.7.1.1.

Definition 5.7.1.1. Let $U(P) = \{u_1, u_2, \dots, u_{n^{U(P)}}\}$ with $u_i = [u_i^e, u_i^l]$ define the set of intervals defined by the planning horizon start time p^e, the unique delivery departure times $T(P)$, and the planning horizon end time p^l. The number of intervals comprising $U(P)$ is either $n^{U(P)} = n^{T(P)} + 1$ if $t^P_{n^{T(P)}} < p^l$, or $n^{U(P)} = n^{T(P)}$ otherwise. Specifically, the intervals are defined as follows:

- $u_1 = [p^e, t_1^P]$.
- $u_{n^{U(P)}} = \begin{cases} [{t^P}_{n^{T(P)}}, p^l] & \text{if } t^P_{n^{T(P)}} < p^l \\ [t^P_{n^{U(P)}-1}, t^P_{n^{U(P)}}] & \text{otherwise.} \end{cases}$
- For each $l = 2, 3, ..., n^{T(P)} - 1$ set $u_l = [t^P_{l-1}, t^P_l]$.

The set $U(P)$ defines disjoint intervals of available machine time to process the jobs $j \in \bar{J}$. The length of each interval $l(U_i) = u_i^l - u_i^e$ depends on the distance between the unique departure times on the one hand and the horizon start time p^e and end time p^l on the other. Consider the remaining jobs $\bar{J}$ and their feasible scheduling within the now separated processing interval P by intervals $U(P)$. The *set of jobs completable in an interval* $U_i \in U(P)$ is defined by

$$N(U_i) = \left\{ j \in \bar{J} \mid r_j + p_j \leqslant u_i^l \leqslant \bar{d}_j \right\}.$$

Note that this definition ensures only that each job of $N(U_i)$ can complete within this interval U_i, without considering any other scheduling decisions. Now consider the scheduling of a subset of jobs of $N(U_i)$ such that all jobs of this subset complete in interval U_i. Let $N'(U_i) \subseteq N(U_i)$ be the subset with maximal cardinality subject to

$$\sum_{j \in N'(U_i)} p_j \leqslant l(U_i),$$

that is, the jobs $N'(U_i)$ are all jointly processable within interval U_i. This set is generated by considering the jobs $N(U_i)$ in order of non-decreasing processing times and choosing the largest subset of ordered elements that does not violate the stated processing constraint. Once this subset $N'(U_i)$ is established, define

$$n(U_i) = \min \left\{ |N(U_i)|, |N'(U_i)| + 1 \right\}$$

as the *maximal number of jobs that can complete (but not necessarily start) production in interval* U_i for each $i \in \{1, \dots, n^{U(P)}\}$. Let the jobs of $N'(U_i)$ be denoted as $j^U_{i,1}, j^U_{i,2}, \dots j^U_{i,n^{U(P)}}$ and ordered by non-decreasing processing times such that $p_{j^U_{i,1}} \leqslant p_{j^U_{i,2}} \leqslant \dots \leqslant p_{j^U_{i,n^{U(P)}}}$ holds.

Finally, each element of the set of completion times $\mathrm{C}(\mathrm{U}_i) = \left\{\mathrm{C}^{\mathrm{U}}_{i,1}, \ldots, \mathrm{C}^{\mathrm{U}}_{i,n(\mathrm{U}_i)}\right\}$ for an interval $\mathrm{U}_i \in \mathrm{U}(\mathrm{P})$ is defined as

$$\mathrm{C}^{\mathrm{U}}_{i,p} = u^l_i - \sum_{j=1}^{n(\mathrm{U}_i)-p} p_{j^{\mathrm{U}}_{i,j}}.$$

Each set $\mathrm{C}(\mathrm{U}_i)$ defines *maximally chosen completion times* to schedule $n(\mathrm{U}_i)$ jobs that complete in interval U_i. That is, the completion times $\mathrm{C}(\mathrm{U}_i)$ represent the scheduling of jobs $\mathrm{N}'(\mathrm{U}_i)$ in order of non-increasing processing times such that the job with minimal processing time completes at time u^l_i. Specifically, considering the job j at position p of the i-th interval, it holds that there is no feasible later completion time for job j such that j has $p-1$ successors completing at or before time u^l_i. As a consequence, given a transport assignment for this job j at some time $t \geqslant u^l_i$, the holding time and holding costs for job j are minimal for this setting with the chosen completion time. The constructed assignment problem allows not only the assignment of job j to the considered completion time $\mathrm{C}^{\mathrm{U}}_{i,p}$, but also the assignment of all other jobs of $\mathrm{N}(\mathrm{U}_i)$ to this time point (or time slot). This relaxation allows for the scheduling of jobs with high holding cost values c^{H}_j to later positions, regardless of their actual processing time requirement. Formally, Lemma 5.7.1.1 proves the validity of the derived completion times.

Lemma 5.7.1.1. *Consider a partial schedule ζ that is extended towards a complete feasible schedule ζ^c with completion times C^c_j for each job $j \in \bar{\mathrm{J}}$ of schedule ζ. Each of the remaining jobs of schedule ζ completes within an interval $\mathrm{U}_i \in \mathrm{U}(\mathrm{P})$ and is therefore an element of a set $\mathrm{N}(\mathrm{U}_i)$.*

Consider the interval U_i of schedule ζ^c and s jobs $j_{i,1}, j_{i,2}, \ldots, j_{i,s}$ that complete within this interval indexed by their relative position from 1 to s, where $j_{i,s}$ is the last job that completes before time u^l_i.

By replacing each completion time $\mathrm{C}^c_{j_{i,p}}$, with $p \in \{1, 2, \ldots, s\}$, by the corresponding completion time $\mathrm{C}^{\mathrm{U}}_{i,p} \in \mathrm{C}(\mathrm{U}_i)$, it follows that in this modified (not necessarily feasible) schedule, no job $j_{i,1}, j_{i,2}, \ldots j_{i,s}$ completes earlier than before.

Proof. The jobs $j_{i,1}, j_{i,2}, \ldots, j_{i,s}$ were not scheduled in ζ, and because they can feasibly complete in interval U_i in schedule ζ^c, they are elements of the set $\mathrm{N}(\mathrm{U}_i)$. Therefore, subset $\mathrm{N}'(\mathrm{U}_i)$ and the set of completion times $\mathrm{C}(\mathrm{U}_i)$ are generated for schedule ζ by considering these jobs.

The attained completion times $\mathrm{C}^c_{j_{i,p}}$ with $p \in \{1, 2, \ldots, s\}$ are maximally chosen if it holds that $p_{j_{i,s}} \leqslant p_{j_{i,s-1}} \leqslant \cdots \leqslant p_{j_{i,1}}$ with job $j_{i,s}$ completing at time u^l_i. Therefore, if $j_{i,k} \in \mathrm{N}'(\mathrm{U}_i)$ for each $j_{i,k}$ with $k \in \{1, 2, \ldots, s\}$ and if the jobs have processing times that are lower or equal to the remaining jobs of set $\mathrm{N}'(\mathrm{U}_i)$, then the completion times $\mathrm{C}^c_{j_{i,p}}$ are equal to the completion times $\mathrm{C}^{\mathrm{U}}_{j_{i,p}} \in \mathrm{C}(\mathrm{U}_i)$. Otherwise, at least one completion time $\mathrm{C}^{\mathrm{U}}_{j_{i,p}}$ is larger than the corresponding completion time $\mathrm{C}^c_{j_{i,p}}$. This proves that no constellation exists in which modifying the completion times as defined by Lemma 5.7.1.1 forces a job to complete earlier. □

The collection of completion times $\mathrm{C}(\mathrm{U}(\mathrm{P}))$ defines the time slots in the LAP formulation. Therefore, an assignment of a job $j \in \bar{\mathrm{J}}$ to a time-slot with time t represents the scheduling of job j such that the job completes on time t. Keep in mind that this representation relaxes the processing of the jobs on the machine.

5.7.1.2 Computation of the Assignment Costs

While an assignment of the LAP formulation directly defines the completion time of a remaining job, the departure time is not expressed directly. Further calculations are necessary to obtain a definitive cost estimate for an assignment. For this, all feasible and non-dominated departure time decisions for a job completing at a specific time are compared, and the costs of the *best* setting are chosen as the assignment costs.

The set of feasible and non-dominated transports, given that a job j completes at a specific time-point τ, was already defined in Section 5.6 as $D^*_{j,\tau}(\zeta)$. The cost of an assignment to such a delivery departure time is split into two parts: the holding costs and the transportation costs. Since the transportation costs in the SFDDHT are fixed costs, once the transport for a single job is planned at a specific time, an additional assignment of jobs does not increase the costs. A way to assess the transportation cost contribution of individual jobs is to split the transportation costs for a delivery batch over all jobs of this batch. The assignment cost calculation adopts this cost accounting. Since the assignment cost values in the LAP formulation are predetermined and do not depend on the chosen assignments, the transportation costs can not be evaluated dynamically when computing the optimal assignment of jobs to time slots. Hence, the minimal cost contribution of a job assigned to a specific transport is estimated as the transportation cost contribution if the delivery batch is of maximal size, which minimizes the split cost share for the considered job. An upper bound for the *maximal delivery batch size* $n^{\Delta}_{g,l}(\bar{J})$ for the l-th transport to customer g is

$$n^{\Delta}_{g,l}(\bar{J}) = \left|\{j \mid j \in \bar{J}_g, r_j + p_j \leqslant t_{g,l} \leqslant \bar{d}_j\}\right|.$$

The set $n^{\Delta}_{g,l}(\bar{J})$ comprises jobs individually assignable and processable on the machine before departure time $t_{g,l}$. It follows that an assignment of job j to the l-th transport of customer g contributes at least $c^T_g/n^{\Delta}_{g,l}(\bar{J})$ transportation costs to the total cost value.

The total cost of assigning job j to the time slot with completion time $C^U_{i,p}$ is calculated for each job $j \in \bar{J}$, $C^U_{i,p} \in C(U(P))$ by

$$c_{j,i,p} = \begin{cases} \min\limits_{t \in D^*_{j,C^U_{i,p}}(\zeta)} \left\{(t - C^U_{i,p}) \cdot c^H_j + c^T_g/n^{\Delta}_{g,l}(\bar{J})\right\} & \text{if } j \in N(U_i), \\ \infty & \text{otherwise.} \end{cases} \tag{5.24}$$

Equation (5.24) chooses the minimal combined holding and transportation cost value for all feasible transport assignments. Infeasible assignments are penalized by a prohibitively large cost value, previously defined simply as ∞.

5.7.1.3 Solving the Resulting Linear Assignment Problem Instance

Once the cost values are computed, a cost matrix with $m^{\text{lap}} = \sum_{i=1}^{n^{U(P)}} n(U_i)$ elements (targets) for the LAP can be formulated. This is carried out by mapping the cost values

$c_{j,i,p}$ to the cost matrix of a LAP such that each row is a job, and each column is a completion time. Specifically, let $\forall j \in \bar{J}, \forall i \in \{1, 2, \dots, n^{U(P)}\}, \forall p \in \{1, 2, \dots, n(U_i)\}$:

$$c^{lap}_{j,\sum_{h=1}^{i-1} n(U_h)+p} = c_{j,i,p}.$$

The initial resulting LAP cost matrix is usually unbalanced, since the defined time slots outnumber the remaining jobs. To balance the matrix, dummy jobs that possess zero cost values are added for assignment to all time slots; that is, new *zero-rows* are appended to the matrix until it is balanced. The assignment of these dummy jobs is arbitrary and does not influence the LAP solution and the total assignment cost.

Variants of the Hungarian method originated by Kuhn (1955) solve the LAP to optimality. The SFDDHT-B&B implementation uses the algorithm described by Munkres (1957), which guarantees an asymptotic run-time complexity of $\mathcal{O}\left(m^{lap^3}\right)$. The attained optimal objective function value serves as the estimated cost component $z_1^e(\zeta, \bar{J})$ of LB I.

5.7.2 Bounding the Number of Additional Transports

This subsection describes the lower and upper bound procedures to derive the minimal and maximal number of additional required transports for feasibly scheduling the remaining jobs $\bar{J}$ of a partial schedule ζ. The bound values of both procedures are utilized by the second (Section 5.7.3) and third (Section 5.7.4) lower bounding procedures during execution of the B&B algorithm. The minimal and maximal number of additional transports to a customer $g \in G$ in a complete solution derived from partial schedule ζ depends on the number of remaining jobs $\bar{J}_g$ and the unassigned transports that are still assignable. First, let the assignable delivery departure times for customer g be

$$T_g^a(\zeta) = \bigcup_{j \in \bar{J}_g} D_{j,0}(\zeta). \tag{5.25}$$

Equation (5.25) unifies the assignable delivery departure times of all remaining jobs of customer g (see Section 5.6). The set of *unused assignable delivery departure times* for customer g, given schedule ζ, is then defined as

$$T_g^{u,a}(\zeta) = T_g^a(\zeta) \setminus \{t_g^{min}\}.$$

5.7.2.1 The Maximal Number of Additional Transports

A valid upper bound to the number of additional transports to a customer $g \in G$ given a partial schedule ζ is stated by Lemma 5.7.2.1 from Bachtenkirch and Bock (2022).

Lemma 5.7.2.1. *The value* $n_g^{D,max}(\zeta) = \min\left\{\left|\bar{J}_g\right|, |T_g^{u,a}(\zeta)|\right\}$ *is an upper bound of the number of additional transports for jobs* $\bar{J}_g$ *that may appear in a complete schedule derived from partial schedule* ζ*.*

Proof. The first part of the expression assumes that each job of $\bar{J}_g$ is assigned to a different delivery departure time. The second part defines the set of non-dominated unassigned delivery departure times ($T_g^{u,a}(\zeta)$). The minimum size of the two sets defines an upper bound to the number of additional deliveries. □

5.7.2.2 The Minimal Number of Additional Transports

The minimal number of additional transports is estimated by finding an assignment of the remaining jobs to transports such that the number of newly used transports is minimal. Since the already used transport at time $t_g^{\min}$ is potentially assignable by some jobs of set $\bar{J}_g$, these jobs must be excluded from the lower bound computation for the number of additional transports. Therefore, Definition 5.7.2.1 defines the set of *critical jobs* as a subset of $\bar{J}_g$.

Definition 5.7.2.1. The set of *critical jobs* of customer g in a schedule ζ is defined as follows:

$$\bar{J}_g^c = \begin{cases} \{j \in \bar{J}_g \mid \bar{d}_j < t_g^{\min}\} & \text{if } t_g^{\min} \in T_g^a(\zeta), \\ \bar{J}_g & \text{otherwise.} \end{cases}$$

Note that if $\bar{J}_g^c$ is an empty set, then no additional transport is required for the remaining jobs of customer g. Additionally, $|\bar{J}_g^c| = 1$ implies that exactly *one* additional transport must exist in a complete schedule. Therefore, assume that $|\bar{J}_g^c| > 1$ for the subsequently presented bounding procedure.

The difficulty of determining an assignment of minimal size for critical jobs $\bar{J}_g^c$ to delivery departure times $T_g^{u,a}(\zeta)$ arises from the different processing and delivery time windows of the jobs: Not all jobs $j \in \bar{J}_g^c$ are necessarily feasibly assignable to each delivery departure time of set $T_g^{u,a}(\zeta)$. Recall that the set T_j^N comprises each delivery departure time that is assignable by job $j \in N$ solely based on the release date, processing time, and deadline parameters. Hence, Definition 5.7.2.2 introduces the assignable delivery departure times for each critical job $j \in \bar{J}_g^c$.

Definition 5.7.2.2. The set of *non-dominated unassigned delivery departure times assignable by a critical* job $j \in \bar{J}_g^c$ of customer g is defined as

$$T_{g,j}^{u,a}(\zeta) = T_g^{u,a}(\zeta) \cap T_j^N.$$

A lower bound to the number of additional transports is described by Lemma 5.7.2.2 from Bachtenkirch and Bock (2022).

Lemma 5.7.2.2. *Given a partial schedule ζ and a customer $g \in G$, assigning each critical job $j \in \bar{J}_g^c$ to a delivery departure time $t \in T_{g,j}^{u,a}(\zeta)$, such that the number of assigned times of $T_g^{u,a}(\zeta)$ is minimal, provides a lower bound to the number of additional transports required for jobs $\bar{J}_g$ in a complete schedule computed from partial schedule ζ.*

Proof. The noncritical jobs $j \in \bar{J}_g \setminus \bar{J}_g^c$ are assignable to delivery departure time $t_g^{\min}$. These jobs do not require an additional transport. Therefore, only jobs $\bar{J}_g^c$ are relevant to establish a lower bound on the number of additional transports.

Each critical job must be assigned to a delivery departure time $t \in T_g^{u,a}(\zeta)$ in a complete schedule derived from ζ. More precisely, the set $T_{g,j}^{u,a}(\zeta)$ for each job $j \in \bar{J}_g^c$ comprises the assignable delivery departure times. Finding a complete assignment of jobs to delivery departure times that minimizes the number of nonidentical delivery departure times provides a lower bound to the number of additional transports required for jobs $\bar{J}_g$ in a complete schedule derived from ζ. □

Note that finding an assignment of jobs to delivery departure times does not imply that a feasible schedule exists that includes these assignments. Specifically, the developed approach does not test the scheduling of jobs with such an assignment. The use of set $T^{u,a}_{g,j}(\zeta)$ to specify whether a job is assignable to a delivery departure time, only guarantees that this job is assignable to one of these delivery departure times in isolation. An assignment of two or more jobs to an identical delivery departure time may lead to release date violations in an actual schedule.

The problem of determining an assignment of jobs to a subset of available delivery departure times with minimal size is equivalent to the (unweighted) minimal set covering problem (SCP). Therefore, the transport assignment problem is mapped to an instance of the SCP. Fortunately, the transformed instances have a special structure that allows for the application of a polynomial-time optimization procedure. As a result, the optimal solution to the SCP instance directly provides the lower bound for the number of additional transports.

In the SCP, a set of elements and a collection of subsets of these elements exist. The objective of the SCP is to find a minimal number of subsets, such that each element is contained in at least *one* of the selected subsets. A linear programming formulation of the SCP, adapted from Ruf and Schöbel (2004), uses a matrix A^{scp} with $M^{scp} = \{1, 2, \ldots, m^{scp}\}$ rows and $N^{scp} = \{1, 2, \ldots, n^{scp}\}$ columns. An entry $a^{scp}_{i,j}$ of the matrix A^{scp} is set to 1 if column (=subset) j comprises the i-th element, otherwise it is set to 0. Furthermore, a *cover* in the SCP is defined as a subset of columns $\bar{N}^{scp} \subseteq N^{scp}$ such that for each row $i \in M^{scp}$, some element $a^{scp}_{i,j}$ with $j \in \bar{N}^{scp}$ equals 1. A cover in the LP formulation is expressed by binary variables

$$X^{scp}_j = \begin{cases} 1 & \text{if } j \in \bar{N}^{scp}, \\ 0 & \text{otherwise.} \end{cases}$$

The objective function and constraints of the LP are

$$\min \sum_{j \in N^{scp}} X^{scp}_j \tag{5.26}$$

subject to

$$\sum_{j \in N^{scp}} a^{scp}_{i,j} X^{SCP}_j \geqslant 1 \quad \forall i \in M^{scp} \tag{5.27}$$

$$X^{scp}_j \in \{0, 1\} \quad \forall j \in N^{scp} \tag{5.28}$$

Objective Function (5.26) minimizes the size of the cover, and Inequality (5.27) describes m^{scp} constraints in which at least one X^{scp}_j with coefficient $a^{scp}_{i,j}$ must equal 1 for satisfaction. Lastly, the X^{scp}_j variables are defined as binary variables by Constraint (5.28).

In general, the SCP is a **strongly $\mathcal{NP}$-hard** problem (Karp, 1972). Fortunately, the structure of the described transport assignment problem defines SCP instances that belong to a class of polynomial solvable instances. Ruf and Schöbel (2004) investigates SCPs whose coefficient matrices have the *consecutive ones property*. That is, the ones of the coefficients ($a^{scp}_{i,j}$) in each row $i \in M^{scp}$ of the LP formulation appear consecutively. For each row i of M^{scp}, it holds that

$$a^{SCP}_{i,h} = 1, a^{SCP}_{i,k} = 1 \text{ with } h \leqslant k \implies a^{scp}_{i,j} = 1 \text{ for all } h \leqslant j \leqslant k.$$

Additionally, let $M^{scp}_j = \{i \in M^{scp} \colon a^{scp}_{i,j} = 1\}$ and $N^{scp}_i = \{j \in N^{scp} \colon a^{scp}_{i,j} = 1\}$ define the *one rows* of column j and *one columns* of row i, respectively. The algorithm that solves the SCP optimally uses two reduction rules first proposed by Toregas and Revelle (1973).

Lemma 5.7.2.3 (adapted from Ruf and Schöbel, 2004)**.**

1. *If $N^{scp}_{i_1} \subseteq N^{scp}_{i_2}$, then an optimal solution to the SCP is found by considering the reduced problem without row $i_2 \in M^{scp}$.*
2. *If $M^{scp}_{j_1} \subseteq M^{scp}_{j_2}$, then an optimal solution to the SCP is found by considering the reduced problem without column $j_1 \in N^{scp}$.*

The first reduction rule of Lemma 5.7.2.3 excludes the row that represents element i_2 because covering element i_1 also covers i_2. In other words, each subset that contains i_1 also contains i_2. Since at least one subset with element i_1 must be part of a cover, i_2 is part of a cover in the final solution. The second reduction rule of Lemma 5.7.2.3 excludes a column (subset) j_1 if another column (subset) j_2 exists where all elements of subset j_1 are also elements of subset j_2. Therefore, the first subset does not need to be considered in a cover, because the second set comprises all elements of the first set and potentially additional elements.

Ruf and Schöbel (2004) propose the consecutive application of the introduced reduction rules to reduce a given matrix A^{scp}.

Lemma 5.7.2.4 (adapted from Ruf and Schöbel, 2004)**.** *The matrix A^{scp} with consecutive ones property can be reduced to an identity matrix by applying a finite sequence of the rules of Lemma 5.7.2.3.*

After application of a finite sequence of the two reduction rules as proposed by Lemma 5.7.2.4, the remaining columns define a *minimal cover*, as stated by Ruf and Schöbel (2004).

To utilize the proposed procedure, the transport assignment problem for each customer must be transformed into an instance of the SCP. The transformation is described by Procedure 5.7.2.3.

Procedure 5.7.2.3. Equate the number of rows (elements) of the SCP with the number of critical jobs, and the number of columns (subsets) with the number of distinct delivery departure times:

$$m^{scp} = \left|\bar{J}^c_g\right| \text{ and}$$
$$n^{scp} = \left|T^{u,a}_g(\zeta)\right|.$$

Let $j^c_{g,1}, j^c_{g,2}, \dots, j^c_{g,m^{scp}}$ denote the jobs of set $\bar{J}^c_g$, and let $t^{u,a}_{g,1}, t^{u,a}_{g,2}, \dots, t^{u,a}_{g,m^{scp}}$ denote the delivery departure times of set $T^{u,a}_g(\zeta)$ such that $t^{u,a}_{g,1} < t^{u,a}_{g,2} < \dots < t^{u,a}_{g,m^{scp}}$. Then, the entries of the coefficient matrix A^{scp} are constructed as follows:

$$a^{scp}_{i,j} = \begin{cases} 1 & \text{if } t^{u,a}_{g,i} \in T^{u,a}_{j^c_{g,j}} \\ 0 & \text{otherwise.} \end{cases} \text{ for each } i \in M^{scp} \text{ and } j \in N^{scp}.$$

The exhaustive application of the reduction rules to the constructed matrix A^{scp} yields a reduced matrix A'^{SCP}. The number of columns n'^{SCP} is a lower bound to the number of additional deliveries. Hence, the value $n^{D,min}_g$ is set to n'^{SCP} for each customer $g \in G$. Example 5.7.2.4 illustrates the approach:

Example 5.7.2.4. Consider the five remaining jobs $\bar{J}_g$ of customer g and the eight assignable delivery departure times $T^a_g(\zeta)$:

Job	C^{min}_j	$\bar{d}_j$
1	5	30
2	12	50
3	20	50
4	35	60
5	40	100

$$T^a_g(\zeta) = \{10, 15, 20, 30, 40, 50, 60, 70\} \text{ and } t^{min}_g = 70$$

As the fifth job is assignable to t^{min}_g, this job is not considered to compute the lower bound. The following table presents the resulting SCP instance with four rows and six columns.

Departure time	10	15	30	40	50	60
Job 1	1	1	1	0	0	0
Job 2	0	1	1	1	1	0
Job 3	0	0	1	1	1	1
Job 4	0	0	0	1	1	1

The application of the rules by Lemma 5.7.2.3 first eliminates row 3, as row 4 is a subset.

Departure time	10	15	30	40	50	60
Job 1	1	1	1	0	0	0
Job 2	0	1	1	1	1	0
Job 4	0	0	0	1	1	1

Columns 1 and 3 are consequently eliminated as subsets of column 2, and columns 5 and 6 are reduced as subsets of column 4.

Departure time	15	40
Job 1	1	0
Job 2	1	1
Job 4	0	1

Lastly, the rules allow for the elimination of row 2, as it is a superset of row 1. The reduction results in the following non-reducible table with two columns.

Departure time	15	40
Job 1	1	0
Job 4	0	1

Therefore, at least two additional transports are required to deliver the remaining jobs.

5.7.3 Parallel Machine Bound

The second lower bound (LB II), namely, the parallel machine bound, utilizes similarities between parallel machine scheduling problems and the SFDDHT. Each delivery batch formed in the SFDDHT is mapped to a single machine, which processes the assigned jobs. In contrast to LB I, detailed in Section 5.7.1, LB II computes an isolated cost estimate for each customer $g \in G$. For each customer, multiple parallel machine environments are considered, where a variation in the number of machines mimics the decision to use a certain number of transports for the remaining jobs. The subsequent lower bound procedure utilizes the equivalence of relaxed instances of the SFDDHT and the parallel machine scheduling problem $Pm||\sum w_jS_j$, which minimizes the weighted start times on parallel machines, as shown in Section 4.4.

The proposed transformation found in the proof of Lemma 4.4.1 implies that an optimal solution for $Pm||\sum w_jS_j$ provides an optimal solution for $1|s|w_jE_j$ with nonrestrictive delivery departure times. That is, the delivery departure times are sufficiently spread out to avoid restricting the cost-minimal scheduling of jobs. It follows that for instances with restrictive delivery departure times (i.e., insufficient available machine time to process all jobs between consecutive transports), an optimal objective function value for $Pm||\sum w_jS_j$ is a lower bound for the SFDDHT. Unfortunately, even problem $Pm||\sum w_jC_j (= \sum w_jS_j)$ is **weakly $\mathcal{NP}$-hard**, as proven by Bruno et al. (1974) and the computation of optimal solutions is hence ruled out in a bounding procedure in reasonable time. Therefore, this section proposes a bounding procedure for the SFDDHT that uses lower bound values computed for instances of problem $Pm||\sum w_jC_j$ to estimate the total cost values for the remaining jobs of schedule ζ.

Several lower bounding procedures for $Pm||\sum w_jC_j$ exist in the literature. Experimental implementations of the SFDDHT-B&B procedure have revealed that bounding techniques proposed by Webster (1995) and Belouadah and Potts (1994) do not attain convincing results (weak bounding values for the SFDDHT). Instead, a result of Elmaghraby and Park (1974) is used to compute lower bounding values: Instances of $Pm||\sum w_jC_j$ with only *agreeable jobs* are polynomially solvable. That is, for two jobs i and j, the processing times $p_i \leqslant p_j$ imply weights $w_i \geqslant w_j$. Elmaghraby and Park (1974) demonstrate that a list scheduling algorithm that considers the jobs in weighted shortest processing time (WSPT) order provides an optimal schedule. That is, in the final schedule, jobs complete in order of non-decreasing p_j/w_j-values.

In general, list scheduling algorithms execute two phases: In the first phase, each job is assigned a priority, and in the second phase, jobs are iteratively selected and then scheduled in order of their priority. The list scheduling algorithm WSPT-List (Elmaghraby and Park, 1974) prioritizes jobs in WSPT order. Thereafter, in each iteration, the remaining job with the highest priority is scheduled on the machine that currently completes earliest. Hence, the completion time for each job is minimized under the constraint that all higher-prioritized jobs are already scheduled on the machines. The algorithm performs an initial sort in $\mathcal{O}(n \log(n))$ time for n jobs to establish the WSPT ordering. Afterward, the n jobs are scheduled on m machines in n iterations. To schedule the next job, the algorithm chooses the machine that currently completes earliest. The implementation of the dynamic machine prioritization uses a min-heap datastructure that stores all m machines, with the machine with minimal completion time at the top. As the heap operations perform with time complexity $\log(m)$, WSPT-List has the worst time complexity of $\mathcal{O}(\max\{n \log(n), n \log(m)\})$.

5.7.3.1 The Bounding Procedure

The overarching bounding procedure is described by Algorithm 5.7 from Bachtenkirch and Bock (2022). For each customer $g \in G$ with a nonempty set of remaining jobs $\bar{J}_g$, the parallel machine problem is constructed by transforming the SFDDHT jobs into agreeable jobs of the parallel machine problem. Specifically, a job set $\bar{J}_g^{II} = \{j_1^{II}, j_2^{II}, \dots, j_{n^{II}}^{II}\}$ is constructed such that the i-th job in this set is assigned the i-th smallest processing time and the i-th largest weight of the original job set $\bar{J}_g$ (see Lines 4 and 5). The machines are constructed under consideration of the bounds for the number of additional transports $n_g^{D,min}$ and $n_g^{D,max}$ and the set of assignable delivery departure times $T_g^a(\zeta)$. The procedure considers the machines $m_1^{II}, m_2^{II}, \dots, m_{n_g^{D,max}}^{II}$ where the first $n_g^{D,max} - 1$ machines are available to start processing jobs at machine release time $r_i^{II} = 0$. The start time of machine $m_{n_g^{D,max}}^{II}$ depends on the existence of an unassigned delivery departure time t_g^1 with $S^1 \leqslant t_g^1 < t_g^{min}$ in schedule ζ. If a transport at time t_g^1 exists, then machine release time $r_{n_g^{D,max}}$ equals $t_g^1 - S^1$, otherwise t_g^1 has value ∞ and $r_{n_g^{D,max}}^{II}$ is set to 0. Similarly, if the transport at time t_g^{min} exists in schedule ζ for customer g, then an additional machine m_0^{II} is introduced, with

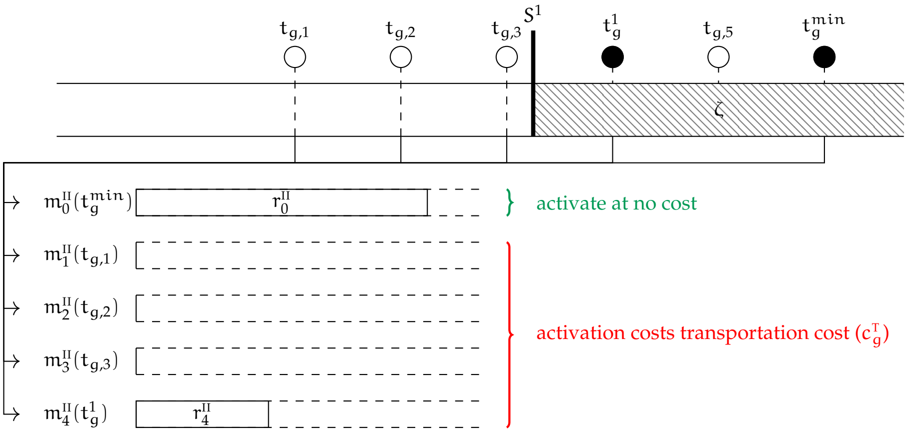
Algorithm 5.7 The Bounding Procedure (LB II)

```
1:  procedure BOUND($\zeta$)
2:      $z^e \leftarrow 0$
3:      for $g \in G$ do
4:          $\bar{J}_g^{II} \leftarrow \bar{J}_g$ with $p_j^{II} = p_j$ and $w_j^{II} = c_j^H$ for all jobs;
5:          Reindex processing times and weights such that $p_i^{II} \leqslant p_j^{II}$ and $w_i^{II} \geqslant w_j^{II}$
            for all $i, j \in \bar{J}_g^{II}$ with $i < j$;
6:          $M_g^{II} \leftarrow \{m_1^{II}, \dots, m_{n_g^{D,max}}^{II}\}$ with $r_i^{II} = 0$ for each $i = 1, \dots, |M_g^{II}|$;
7:          if $t_g^{min} \neq \infty$ then $M_g^{II} \leftarrow M_g^{II} \cup \{m_0^{II}\}$ with $r_0^{II} = t_g^{min} - S^1$;
8:          if $t_g^1 \neq \infty$ then $r_{n_g^{D,max}}^{II} = t_g^1 - S^1$;
9:          $z^e \leftarrow z^e +$ SOLVE_CUSTOMER($n_g^{D,min}, n_g^{D,max}, M_g^{II}, \bar{J}_g^{II}$)
10:     return $z(\zeta) + z^e$;
```

machine release time $r_0^{II} = t_g^{min} - S^1$, that models the unavoidable holding time for a job $j \in \bar{J}_g$ assigned to this delivery departure time. Note that WSPT-List is still valid even if machine release times are considered, because the release times can be modeled as *dummy jobs* with sufficiently large weights such that scheduling these jobs first adheres to the required WSPT ordering. The machine structure is depicted in Figure 5.15.

Figure 5.15

Illustration of the Problem Transformation for LB II



Note. The original problem is transformed into problem $Pm||\sum w_j S_j$. Delivery dates of customer g are transformed into identical parallel machines with an infinite production capacity. The machines m_0^{II} and $m^{II}_{n_g^{D,max}}$ require an initial setup time that models the unavoidable holding times.

5.7.3.2 Calculation of Customer-Related Costs

After constructing the job and machine data, the bounding procedure calculates the minimal cost value attainable by scheduling the jobs optimally on machines M_g^{II}. This procedure is displayed by Algorithm 5.8 from Bachtenkirch and Bock (2022).

Algorithm 5.8 Bounding Customer Costs (LB II)

```
procedure SOLVE_CUSTOMER($n_g^{D,min}, n_g^{D,max}, M_g^{II}, \bar{J}_g^{II}$)
    $z_g^{II,*} \leftarrow \infty$
    for $m = n_g^{D,min}, \dots, n_g^{D,max}$ do
        $z_{g,m}^{II,T} \leftarrow c_g^T \cdot m$;
        $z_{g,m}^{II,H} \leftarrow$ solve $Pm||\sum w_j C_j$ with WSPT-List on machines
            $\{m_i^{II} \mid m_i^{II} \in M_g^{II}, i \leqslant m\}$ with jobs $\bar{J}_g^{II}$;
        $z_{g,m}^{II} \leftarrow z_{g,m}^{II,T} + z_{g,m}^{II,H}$;
        $z_g^{II,*} \leftarrow \min\{z_g^{II,*}, z_{g,m}^{II}\}$;
        if $(z_{g,m}^{II,H} < c_g^T)$ or $(z_{g,m}^{II,H} = z_{g,m-1}^{II,H})$ or $(z_{g,m}^{II,H} - z_{g,m+1}^{II,H,min} < c_g^T)$ then return $z_g^{II,*}$;
    return $z_g^{II,*}$;
```

The total costs for the remaining jobs $j \in \bar{J}_g$ are computed by iteratively solving parallel machine problems with an increasing number of machines in each iteration. Specifically, solutions to instances that enable $n_g^{D,min} \leqslant m \leqslant n_g^{D,max}$ additional transports are computed.

Hence, in the m-th iteration, WSPT-list optimally schedules the jobs $j_1^{II}, j_2^{II}, \dots, j_{n^{II}}^{II}$ on the machines m_i^{II} with $i \leqslant m$. The application of objective function $\sum w_j S_j$ yields estimate $z_{g,m}^{II,H}$ for the holding costs of jobs $j \in \bar{J}_g$ with up to m additional transports allowed. The transportation costs are estimated by $z_{g,m}^{II,T} = m \cdot c_g^T$. In total, the decision to have m additional deliveries for customer g results in estimated costs $z_{g,m}^{II} = z_{g,m}^{II,H} + z_{g,m}^{II,T}$.

The iterative process terminates early under the following three conditions:

1. $z_{g,m}^{II,H} < c_g^T$: The current estimated holding cost with m additional transports is smaller than the transportation cost of an additional delivery. As a result, even if no holding costs are present in an estimate with $m+1$ additional transports, the total costs can not be lowered by an additional transport.
2. $z_{g,m}^{II,H} = z_{g,m-1}^{II,H}$: The holding costs were not reduced by the addition of a transport, because each job is solely processed on a machine.
3. $z_{g,m}^{II,H} - z_{g,m+1}^{II,H,min} < c_g^T$: The term $z_{g,m+1}^{II,H,min}$ is an estimate for the holding costs derived by a bound from Eastman et al. (1964). The original bound for $Pm||\sum w_j C_j$ is stated as

   $$\mathcal{C}_m \leqslant \max\left\{\mathcal{C}_n, \frac{1}{m}\mathcal{C}_1 + \frac{m-1}{2m}\mathcal{C}_n\right\},$$

   where $\mathcal{C}_m$ is the optimal value of an instance of $Pm||\sum w_j C_j$ with m machines, value $\mathcal{C}_1$ is the optimal cost of scheduling all jobs on a single machine, and $\mathcal{C}_n$ is the optimal cost of scheduling the n jobs in an environment with n machines. The value for $\mathcal{C}_1$ is calculated by scheduling the jobs in WSPT order, which is the optimal policy in the single machine case. $\mathcal{C}_n$ returns total cost $\sum_{j=1}^n w_j p_j$, since each job starts processing at time 0 on a unique machine. As the holding costs of jobs are equivalent to weighted start times, the above bound must be corrected by the constant difference $\sum_{j=1}^n w_j p_j = \mathcal{C}_n$ between an optimal schedule for objectives $\sum w_j C_j$ and $\sum w_j S_j$. Hence, the procedure uses bound value

   $$z_{g,m}^{II,H,min} = \frac{1}{m}\mathcal{C}_1 - \frac{m+1}{2m}\mathcal{C}_n.$$

   The lower bound $z_{g,m+1}^{II,H,min}$ in iteration m provides the maximal holding cost difference between iterations m and $m+1$. If the resulting value indicates that an additional delivery can not lower total costs, then the cost-minimal decision was already found.

Remark. One might suspect that $z_{g,m+1}^{II} \geqslant z_{g,m}^{II}$ implies $z_g^{II} = z_{g,m}^{II}$; that is, once an additional delivery or machine does not decrease the cost, then $z_{g,m}^{II}$ is the minimum value. Attempts to prove the validity of this suspicion by using the lower and upper bounds provided by Eastman et al. (1964) failed. The question of whether the initial suspicion is correct remains open.

For each customer $g \in G$, value $z_g^{II,*}$ is the minimal $z_{g,m}^{II}$ value computed in the process. Algorithm 5.8 returns the lowest cost value $z_g^{II,*}$ of all tested machine configurations. The final estimate of the proposed bound is $z_2^e(\zeta, \bar{J}) = \sum_{g \in G} z_g^{II,*}$, which sums the individual customer-related cost estimates.

5.7.4 Partition Bound

The *third* lower bound (LB III), namely, the partition bound, partitions jobs into two classes. Each job $j \in \bar{J}$ is classified either as a *terminal* or a *nonterminal* job. Recall from Section 5.2.5 that a terminal job is a job that completes last in a delivery batch in a schedule ζ, while a nonterminal job has at least one successor in schedule ζ that is terminal. The proposed bound executes two steps: In the first step, an initial partition is computed that minimizes the overall cost; the second step utilizes the bound values for the minimal and maximal additional transports that may require the reclassification of jobs.

5.7.4.1 Computation of the Initial Partition

The proposed bound assumes that a terminal job causes the requested transport and, therefore, contributes the total transportation cost value $c^{T}_{g^{N}(j)}$ to the total objective value, while all nonterminal jobs can be transported for free. It holds by definition that a nonterminal job j has at least one successor of an identical customer that is processed after job j and completes before job j starts transportation. Hence, a nonterminal job always contributes holding costs to the objective function value (assuming $c^{H}_{j} > 0$). Therefore, the cost contribution of a nonterminal jobs depends on the set of feasible successors of an identical customer, as described by Definition 5.7.4.1:

Definition 5.7.4.1. Consider a partial schedule ζ, a customer $g \in G$, and a remaining job $j \in \bar{J}^{g}$. The set $\bar{J}^{s}_{j}$ comprises the feasible successors of job j of identical customer g. The definition of this set is solely based on the job time-windows.

$$\bar{J}^{s}_{j} = \left\{k \mid k \in \bar{J}_{g}, k \neq j,\ \max\{r_k, r_j + p_j\} + p_k \leqslant \bar{d}_j, r_j + p_j + p_k \leqslant \bar{d}_k\right\}.$$

For each job k of $\bar{J}^{s}_{j}$, Definition 5.7.4.1 allows the processing sequence (j, k). A delivery batch that comprises nonterminal job j and terminal job k hence may exist in a feasible schedule. The following lower bound computation ensures fast computability by only considering the case where a job j is succeeded by a single job of set $\bar{J}^{s}_{j}$ before the transport of both jobs takes place.The lower bound computation compares the cost of classifying a job as terminal or nonterminal. The estimated cost of remaining job $j \in \bar{J}_{g}$ classified as a terminal job is

$$z^{III,t}_{j} = c^{T}_{g}.$$

If job j is considered to be nonterminal, then it is succeeded either by a job $k \in \bar{J}^{s}_{j}$ or by all already scheduled jobs up to the minimal delivery departure time t^{min}_{g} if a transport for customer g already exists in schedule ζ. To ensure the validity of the proposed lower bound, job k with minimal processing time is chosen such that the assumed holding time of job j is minimized.

$$z^{III,n}_{j} = \begin{cases} \min\left\{(t^{min}_{g} - S^{1}), \min_{k \in \bar{J}^{s}_{j}}\{p_k\}\right\} \cdot c^{H}_{j} & \text{if } J_g \neq \varnothing \\ \min_{k \in \bar{J}^{s}_{j}}\{p_j\} \cdot c^{H}_{j} & \text{otherwise.} \end{cases}$$

The minimal cost contribution of job j of customer g is

$$z^{III,*}_{j} = \min\left\{z^{III,t}_{j}, z^{III,n}_{j}\right\}.$$

The sum of the $z_j^{\mathrm{III},*}$ values over all remaining jobs $\bar{J}_g$ is consequently a valid estimate for the total cost contribution.

5.7.4.2 Reclassification

The initial estimate can be improved by considering the bound values for the number of additional transports. After computing the contribution values, the remaining jobs can be partitioned into the set of jobs classified as terminal and the set of jobs classified as nonterminal:

$$\bar{J}_g^{\mathrm{III},t} = \{j \in \bar{J}_g \mid z_j^{\mathrm{III},t} \leqslant z_j^{\mathrm{III},n}\} \text{ and}$$

$$\bar{J}_g^{n} = \{j \in \bar{J}_g \mid z_j^{\mathrm{III},t} > z_j^{\mathrm{III},n}\}.$$

The resulting classification implies $\left|\bar{J}_g^{\mathrm{III},t}\right|$ additional transports for customer g. This number may contradict the valid bounds $n_g^{\mathrm{D,min}}$ and $n_g^{\mathrm{D,max}}$ for the number of additional transports. The classification is thus modified in a cost-minimal way such that the classification adheres to the transport bounds. Let $n_g^{n\to t}$ and $n_g^{t\to n}$ be the correction values, where

$$n_g^{n\to t} = \max\{0, n_g^{\mathrm{D,min}} - \left|\bar{J}_g^{\mathrm{III},t}\right|\}$$

indicates the number of nonterminal jobs of customer g that must be reclassified to *terminal*, and

$$n_g^{t\to n} = \max\{0, \left|\bar{J}_g^{\mathrm{III},t}\right| - n_g^{\mathrm{D,min}}\}$$

indicates the number of terminal jobs of customer g that must be reclassified to *nonterminal*. Only one of the two values can be simultaneously nonzero for a specific customer g. The final classification of jobs of customer g is established as follows:

1. $n_g^{n\to t} > 0$: The $n_g^{n\to t}$ nonterminal jobs of customer g with highest holding cost contribution $z_j^{\mathrm{III},n}$ are reclassified as terminal jobs. Let $j_{g,n}^{n}$ be the element of $\bar{J}_g^{n}$ with the n-th largest cost contribution $z_j^{\mathrm{III},n}$. Then, the final partitioning of the jobs of customer g is

$$\bar{J}_g^{t*} = \bar{J}_g^{t} \cup \left\{j_{g,1}^{n}, j_{g,2}^{n}, \ldots, j_{g,n_g^{n\to t}}^{n}\right\} \text{ and}$$

$$\bar{J}_g^{n*} = \bar{J}_g^{n} \setminus \left\{j_{g,1}^{n}, j_{g,2}^{n}, \ldots, j_{g,n_g^{n\to t}}^{n}\right\}.$$

2. $n_g^{t\to n} > 0$: The $n_g^{t\to n}$ terminal jobs of customer g with lowest holding cost contribution $z_j^{\mathrm{III},n}$ are reclassified as nonterminal jobs. Let $j_{g,t}^{t}$ be the element of $\bar{J}_g^{\mathrm{III},t}$ with the n-th smallest cost contribution $z_j^{\mathrm{III},n}$. Then, the final partitioning of the jobs of customer g is

$$\bar{J}_g^{t*} = \bar{J}_g^{t} \setminus \left\{j_{g,1}^{t}, j_{g,2}^{t}, \ldots, j_{g,n_g^{t\to n}}^{t}\right\} \text{ and}$$

$$\bar{J}_g^{n*} = \bar{J}_g^{n} \cup \left\{j_{g,1}^{t}, j_{g,2}^{t}, \ldots, j_{g,n_g^{t\to n}}^{t}\right\}.$$

3. $n_g^{n\to t} = n_g^{t\to n} = 0$: The number of terminal jobs is within the specified bounds. Hence, the initially computed partition remains:

$$\bar{J}_g^{t*} = \bar{J}_g^{t} \text{ and}$$

$$\bar{J}_g^{n*} = \bar{J}_g^{n}.$$

Note that the reclassification applied in Cases 1 and 2 does not decrease costs, since the initial classification is cost minimal. In the SFDDHT-B&B implementation, jobs are ordered by their nonterminal cost contribution. For each job, a label is stored that marks a job as either terminal or nonterminal, and the reclassification simply changes the stored label. The final cost estimate amounts to

$$z_3^e = \sum_{g \in G} \left(\sum_{j \in \bar{J}_g^{t*}} z_j^{\mathrm{III},t} + \sum_{j \in \bar{J}_g^{n*}} z_j^{\mathrm{III},n} \right).$$

5.7.5 Discussion of the Lower Bounds

The three proposed lower bounds all focus on different aspects of the SFDDHT to estimate the remaining scheduling costs.

LB I, presented in Section 5.7.1, constructs time slots for the remaining jobs and allows for jobs to be assigned to a time slot if the completion time is feasible. This definition does not consider whether the considered job fits into a time slot (i.e., has lower or equal processing time compared to the time slot's interval length). While a valid bound is attained, this relaxation of the processing constraints may result in poor bounding values for a set of jobs with heterogeneous processing times, since jobs with high p_j values may be assigned to time slots that occupy only a fraction of the required processing duration. On the other hand, with relatively homogeneous processing times, the available machine time is only relaxed slightly. Depending on the structure of the considered SFDDHT instance, the transportation cost associated with an assignment to a transport may be low if jobs have generally large time windows such that many jobs of the same customer can be transported at the same time. For instances with relatively few different batching possibilities, the estimated cost values may result in good estimates. A strong point of the proposed bound is that it largely preserves the time window restrictions of jobs if the cost-efficient choice of jobs for each time slot is limited in the LAP instance in which the computed assignment is expected to resemble a feasible schedule of the remaining jobs.

LB II, proposed in Section 5.7.3, relaxes the time window constraints of all jobs and additionally considers the scheduling of jobs for each customer separately without coordinating the production between jobs by different customers. The estimated costs calculated by transformation into parallel machine problems can be seen as an idealized production and distribution schedule that always produces an identical customer's jobs together in a production block without interference from other jobs. The transformation to agreeable jobs performs especially well for homogeneous job sets. In comparison, to LB I, LB II preserves the impact of transportation costs, also strengthened by the minimal and maximal transport bounds.

Similarly to the second lower bound, the third lower bound (Section 5.7.4) provides a decent estimation of transportation costs by utilizing the proposed bounds of the additional transports. The cost estimate for the holding costs is calculated in a much simpler way which is expected to be less tight but faster computable.

5.8 Algorithm Execution

This section details the execution of the SFDDHT-B&B algorithm that orchestrates the procedures described in the previous sections of this chapter. The algorithm processes an SFDDHT instance that defines the initial subproblem $\mathcal{X}_0^c$. It constructs a search tree $\mathcal{T}^c$ that comprises a single root node for the initial subproblem such that $\mathcal{N}^c = \{\mathcal{X}_0^c\}$. The approach uses a BFS search strategy with an active list $\mathcal{L}^c$ that is implemented by a heap data structure. This heap is initialized with subproblem $\mathcal{X}_0^c$. Moreover, the algorithm optionally accepts an initial solution (canonical schedule) $\hat{x}^c$ with upper bound value $f(\hat{x}^c)$ generated by some heuristic procedure.

Before the search process begins its execution, the preprocessing tables M^{prec} and M^{pos} are precomputed (see Section 5.4). Afterward, the search process iteratively selects the next subproblem $\mathcal{X}_i^c$ from list $\mathcal{L}^c$ with the minimal lower bound value $f(\mathcal{X}_i^c)$. Given this node $\mathcal{X}_i^c$, the corresponding schedule ζ_i is constructible by following the relevant stored parent-child arcs of set $\mathcal{E}^c$ from the i-th node towards the root node with index 0. As the algorithm uses the proposed dominance table (see Section 5.3.6) to eliminate not only newly constructed subproblems but also already evaluated ones, the algorithm may detect a *dominated* label concerning schedule ζ_i at this point. Specifically, the schedule ζ_i or one of the extended schedules towards ζ_i, defined by the ancestor nodes, may be labeled *dominated*. If this is the case, the algorithm deletes the current node and does not branch on this subproblem. Otherwise, if schedule ζ_i is not dominated, then the algorithm generates the branching candidates $B(\zeta_i)$ that define the possible extensions of schedule ζ_i towards an extended schedule with an additional job processed before jobs J and transported at a feasible, non-dominated delivery departure time (see Section 5.6). The algorithm subsequently carries out the following steps for each branching candidate:

1. Construction of branching candidate schedule ζ_b.
2. Execution of the dominance procedures *production block dominance procedure* (Section 5.3.4), *permutation dominance procedure* Section 5.3.3, and *delivery batch dominance procedure* Section 5.3.5 on ζ_b. Suppose that at any point during the dominance testing, schedule ζ_b is dominated by one procedure. In that case, the routine discards the branching candidate, and the algorithm continues to evaluate the next candidate.
3. Processing of ζ_b by the *dominance table* Section 5.3.6. If ζ_b is dominated, then the branching process terminates for this candidate. Otherwise, the table stores the information of ζ_b as a new entry. Furthermore, if ζ_b dominates an already inserted schedule, then the corresponding subproblem is labeled as *dominated*. This label is stored in the corresponding node of the dominated subproblem in search tree $\mathcal{T}$.
4. Application of one or multiple of the lower bounds LB I, LB II, and LB III (described in Sections 5.7.1, 5.7.3, and 5.7.4) to generate lower bound $LB(\zeta)$ with subsequent bound dominance testing. That is, the generated lower bound $LB(\zeta_b)$ is tested against the current upper bound $f(\hat{x})$. If $LB(\zeta_b) \geqslant f(\hat{x}^c)$, then the candidate is discarded. In the case that schedule ζ_b is a complete schedule and $LB(\zeta_b) < f(\hat{x}^c)$, the current best solution $\hat{x}^c$ is replaced by ζ_b.

5. Extensions of the search tree by a new node and insertion into the active list $\mathcal{L}^c$ at the location that depends on the lower bound of ζ_b if ζ_b is not dominated by the pruning rules and is not a complete schedule.

The described subproblem selection and branching phases execute as long as the active list $\mathcal{L}^c$ is nonempty, which means that the algorithm has not solved the problem instance by finding an optimal solution and proving the solution's optimality. The algorithm returns the best-found schedule upon termination if the input instance is feasibly solvable. However, if the opposite holds, then the algorithm returns no solution but signals that no feasible solution exists for the input instance.

Note that most of the required memory is allocated before the execution of the algorithm in the implementation. That is, the search tree may store, at most, a fixed number of nodes simultaneously. If an instance is not solved with this limitation, then the algorithm terminates early. Similarly, memory is pre-allocated for the dominance table entries, while the required hash table is allowed to grow when necessary.

To speed up the schedule generation of subproblems extracted from the active list $\mathcal{L}^c$ and the schedule generation of branching candidates, the implementation performs updates on the schedule data rather than relying on a full reevaluation, if possible. That is, given a schedule ζ_i of subproblem $\mathcal{X}_i^c$, an extended schedule of ζ_i with an additional job scheduled in front of the jobs already can be evaluated simply by adding the new job canonically, adding its holding cost to the previous objective value, and adding transportation costs only if the added job is using a former unused transport.

Similarly, an efficient evaluation of branching candidates is used for two consecutive evaluated schedules. If the job is identical in both schedules, then the evaluation procedure reschedules the added job and evaluates its transport assignment. Otherwise, it removes the last added job and adds the one proposed by the currently investigated branching candidate. The above implementation details minimize the time the algorithm spends while not performing bound or dominance computations.

5.9 Conclusion

This chapter described the SFDDHT-B&B algorithm that solves instances of the SFDDHT optimally. A significant effort went into utilizing the scheduling problem's structural properties to develop an efficient branching strategy and to establish dominance rules and lower bounds that significantly reduce the searched solution space for an optimal solution. Furthermore, the chapter offered specified details for an efficient implementation of the proposed procedures and described important data-structures to attain computational efficiency.

Following a brief introduction to the principles of B&B algorithms in Section 5.1, the remaining sections described the SFDDHT-B&B. First, Section 5.2 defined the solution space searched by the enumerative approach. In particular, we reduced the search space to only search for solutions that are *canonical schedules* and derived several properties of optimal solutions. Moreover, Section 5.3 provided an in-depth description of applicable dominance relations and computationally efficient dominance procedures to reduce the searched solution space. Then, Sections 5.4 to 5.6 were dedicated to the minimization of the number of necessary search tree branches by initial analysis of the provided problem instance. *Three* different lower bounds were proposed in Section 5.7, and the program execution was detailed in Section 5.8.

Chapter 6

The Developed Greedy Randomized Adaptive Search Procedure

The contents presented in this chapter also appear in the work of Bachtenkirch and Bock (2022), titled Finding efficient make-to-order production schedules, which is published in the European Journal of Operational Research. To avoid impairing the reading flow, citations do not reference this article continuously throughout this chapter. However, when introducing a mathematical statement, such as a lemma, proposition, or proof, from the stated paper, the text includes a reference. Moreover, tables, figures, and algorithms that are taken from this paper are referenced as per usual.

This chapter introduces a specifically designed GRASP that generates heuristic solutions for the SFDDHT. GRASP is a meta-heuristic framework applicable to various optimization problems originally derived from a procedure proposed by Feo and Resende (1989). Since the SFDDHT is **strongly $\mathcal{NP}$-hard** the B&B algorithm described in Chapter 5 inevitably requires an unreasonably large amount of computer memory to store nodes, or it mandates unacceptable amounts of computing time to provide optimal solutions when executed on large problem instances. Therefore, a heuristic procedure is required to provide high-quality solutions for large instances in a reasonable time frame. Additionally, the heuristic construction of a solution provides an upper bound for the objective value. Such a derived upper bound can be utilized to test for bound dominances during the B&B search process, even before the exact procedure encounters a leaf node itself. Furthermore, the heuristic approach is a suitable optimization tool usable in time-critical applications such as the RTC approach described in Chapter 7.

The remainder of this chapter explains the key ideas and components of GRASP in Section 6.1. The GRASP algorithm for the SFDDHT (SFDDHT-GRASP) is subsequently described in detail. The heuristic applies the general premise of the iterative randomized construction and the improvement of solutions. Since the *feasible* construction of a solution can not be ensured, the SFDDHT-GRASP additionally implements a repair procedure that attempts to establish feasible solutions. The heuristic is complemented by the application of a so-called *path relinking* procedure, which combines the solution characteristics of two solutions to explore the solution space in regions close to solutions of high quality. In pursuit of a fast sampling of the solution space, the SFDDHT-GRASP enables parallel execution on multiple threads. The heuristic procedure is described in the following order. Section 6.2 describes the construction phase, and the improvement and repair procedures are then outlined in Section 6.3. Afterward, Section 6.4 elaborates on the path relinking procedure used by the SFDDHT-GRASP. Lastly, Section 6.5 explains the interaction of the previously described components and their application in the SFDDHT-GRASP. As the implementation of the algorithm runs simultaneously on multiple threads, a brief description of implementation details is also provided. Finally, Section 6.6 concludes this chapter.

6.1 Principles of the Solution Method

According to Feo and Resende (1995), GRASP is an iterative randomized sampling technique. In each iteration of the procedure, a new solution to an optimization problem is generated from scratch. Upon termination of the procedure, the best-found solution is returned. Each *GRASP iteration* is split into two phases: solution construction and solution improvement. In the construction phase, to generate different solutions, a solution is generated with some randomized construction techniques involved. In the improvement phase, a local improvement procedure is applied to a constructed solution. The basic procedure by Feo and Resende (1995) is illustrated by Algorithm 6.1 for an instance $\mathcal{X}$ of a minimization problem. Each iteration constructs (procedure ConstructGreedyRandomizedSolution) and improves (procedure LocalSearch) a new solution without relying on information from previous iterations.

Algorithm 6.1 A GRASP Algorithm for a Minimization Problem

```
procedure GRASP($\mathcal{X}$)
    Initialize $\hat{s}$ arbitrarily with $f(\hat{s}) = \infty$;        ▷ $\hat{s}$: best found solution
    while GRASP stopping criterion not satisfied do
        $s \leftarrow$ ConstructGreedyRandomizedSolution();
        LocalSearch($s$);
        if $f(s) \leqslant f(\hat{s})$ then
            $\hat{s} \leftarrow s$;
    return $\hat{s}$;
```

Note. Adapted from Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

The construction phase builds a feasible solution one element at a time. Possible extensions to a solution are called *candidates*, which are stored in a so-called candidate list (CL). The construction is guided by a *greedy function* that measures the myopic benefit of extending a partial solution by a candidate. To generate different solutions, the choice of the next element, which extends the solution, is determined partially by the greedy function and partially by randomness. A selection of the best candidates is inserted into a restricted candidate list (RCL) from which a candidate is drawn randomly. The procedure ConstructGreedyRandomizedSolution as described by Feo and Resende (1995) is depicted in Algorithm 6.2.

Algorithm 6.2 The GRASP Construction Phase

```
procedure ConstructGreedyRandomizedSolution
    while Solution construction of $s$ incomplete do
        CL $\leftarrow$ MakeCL(s)                    ▷ evaluated by a greedy function
        RCL $\leftarrow$ MakeRCL(CL);
        $c$ $\leftarrow$ SelectCandidateAtRandom(RCL);
        $s \leftarrow s \cup \{c\}$;
    return $s$
```

Note. Adapted from Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

In general, the construction of partial solutions is problem dependent, as is the implementation of the greedy function (denoted as f in Algorithm 6.2). A logical choice for the greedy function is the objective function of an optimization problem.

A complete solution generated by procedure ConstructGreedyRandomizedSolution is usually not locally optimal with respect to some local neighborhood definition. The improvement phase of GRASP generates a locally optimal solution based on the initially constructed solution s. In each iteration, the local neighborhood of the current solution s is constructed. The solution s is replaced with the best neighbor if s is not locally optimal. Similarly to the construction phase, the actual implementation of the improvement phase is highly problem dependent.

Algorithm 6.3 The GRASP Improvement Phase

```
procedure LocalSearch($s$)
    Let $\mathcal{N}$ denote the neighborhood structure;
    while $s$ is not locally optimal do
        Find best neighbor $n \in \mathcal{N}(s)$
        if $f(n) < f(s)$ then
            $s \leftarrow n$
    return $s$
```

Note. Adapted from Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

Feo and Resende argue that the use of the GRASP framework is appealing, since its implementation is relatively simple and because the framework relies only on a few parameters. An implementation of GRASP only needs to specify a termination criterion and a method to restrict the CL. While the basic GRASP framework provides satisfactory results in many applications (Feo and Resende, 1995), numerous extensions and hybridizations of the GRASP procedure can be found in the literature. Gendreau et al. (2010) note that there is a general trend to hybridize existing procedures to provide better results than with *pure* heuristics. Resende and Ribeiro (2010) discusses various extensions of the GRASP method.

6.2 Construction Phase

This section describes the construction phase of the SFDDHT-GRASP. In each GRASP iteration, a partial solution is extended by adding a single job. The construction procedure consequently requires a total of n^N steps to define a complete solution for the SFDDHT. To avoid confusion between the notation used to describe a schedule found by the B&B approach and a schedule(=solution) constructed by the SFDDHT-GRASP, let $\mathcal{S}$ symbolize a schedule constructed by the SFDDHT-GRASP. Specifically, let $\mathcal{S}_k$ be the schedule built after completing k steps of the construction procedure in a particular GRASP iteration. Furthermore, let the set of scheduled jobs in step k be $J_k \subseteq N$ and the complementary set of remaining jobs in step k be $J_k = N \setminus J_k$. The construction procedure starts with an empty schedule $\mathcal{S}_0$ with no jobs scheduled ($J_0 = \varnothing$) and the remaining jobs set to $\bar{J}_0 = N$. During construction step k, a remaining job j of $\bar{J}_{k-1}$ is selected and inserted into schedule $\mathcal{S}_{k-1}$ at one of the positions $1, \ldots, k$ with an assignment to a delivery departure time t of T_j^N. It follows that after the insertion of job j at position i, the jobs previously scheduled at positions $i, \ldots, k-1$ then occupy positions $i+1, \ldots, k$.

6.2.1 Selection Order of Jobs

Remember that generating a feasible solution for the SFDDHT is a complex task. Since the underlying feasibility problem $1|r_j, \bar{d}|-$ is **strongly $\mathcal{NP}$-hard** (Garey and Johnson, 1979), no construction procedure exists (under the assumption $\mathcal{P} \neq \mathcal{NP}$) that either guarantees the construction of a feasible schedule or detects that no feasible solution for an instance exists, in polynomial time. As a consequence of the problem complexity, the step-wise construction of schedules, applied by the SFDDHT-GRASP may lead to partial schedules with no feasible insertions available. To work around this issue, the construction procedure uses a fixed job selection order that schedules jobs with restrictive time windows first. The idea for this fixed selection order stems from Gendreau et al. (1998), who utilize a fixed selection order to construct feasible tours for the traveling salesman problem with time windows (TSPTW). In the proposed heuristic approach for the TSPTW, a tour is iteratively constructed by considering customer requests in order of their time-window restrictiveness. Analogously to the proposition by the authors, selecting jobs in order of increasing time window flexibility provides a way to potentially increase the chances of generating feasible schedules. This claim is supported by the empirical study of the construction heuristic for the TSPTW, and the SFDDHT-GRASP adopts this idea to construct solutions. The construction heuristic adds jobs to a partial schedule in order of non-decreasing *processing time window length*

$$p_j^w = \bar{d}_j - r_j.$$

The SFDDHT-GRASP consequently establishes an initial ordering of jobs $j_1^*, j_2^*, \ldots, j_{n^N}^*$ such that $p_{j_1^*}^w \leqslant p_{j_2^*}^w \leqslant \cdots \leqslant p_{j_{n^N}^*}^w$ holds. By adhering to this ordering, in each iteration k of the construction phase, job j_k^* is added to the currently considered partial schedule $\mathcal{S}_{k-1}$. The ordering is computed before the first iteration of the SFDDHT-GRASP starts.

Therefore, the choice of the remaining job in each iteration is independent of the previously undertaken construction steps in a GRASP iteration.

6.2.2 Construction Moves

The integration of job j^*_k into partial schedule $\mathcal{S}_{k-1}$ is carried out by setting the relative position of job j^*_k and its delivery departure time. Each of the alternative decisions (construction moves) in step k is represented by a tuple (i, t) with $i \in \{1, 2, \dots, k\}$ describing the insertion position and $t \in T^N_j$ describing the pickup time of j^*_k. A construction move (i, t) is evaluated by computing the resulting objective function value of schedule $\mathcal{S}_{k-1}$ extended by adding job j^*_k at position i and delivery departure time t. Let a schedule $\mathcal{S}_k$ be defined by a set of triples

$$\mathcal{S}_k = \left\{(1, j^{\mathcal{S}}_{k,1}, t^{\mathcal{S}}_{k,1}), \dots, (k, j^{\mathcal{S}}_{k,k}, t^{\mathcal{S}}_{k,k})\right\},$$

where each triple (i, j, t) represents the job j currently scheduled at position i with delivery departure time t in schedule $\mathcal{S}_k$. As the construction procedure extends partial schedules by inserting a job at *some* position, the position index for a triple is only relatively defined. That is, by insertion of a new job at position $i \neq k$ in a schedule $\mathcal{S}_k$, the positions for jobs at positions $i, \dots, k$ increment by *one*. In step k, the evaluation procedure extends the schedule $\mathcal{S}_{k-1}$ to a *temporary schedule*, which is defined as follows:

$$\begin{aligned}\mathcal{S}_{k,i,t} = \Big\{&(1, j^{\mathcal{S}}_{k-1,1}, t^{\mathcal{S}}_{k-1,1}), \dots, (i-1, j^{\mathcal{S}}_{k-1,i-1}, t^{\mathcal{S}}_{k-1,i-1}), (i, j^*_k, t),\\ &\qquad (i+1, j^{\mathcal{S}}_{k-1,i}, t^{\mathcal{S}}_{k-1,i}), \dots, (k, j^{\mathcal{S}}_{k-1,k-1}, t^{\mathcal{S}}_{k-1,k-1})\Big\}.\end{aligned}$$

The addition of j^*_k to schedule $\mathcal{S}_{k-1}$ does not alter preexisting transportation assignments for any job of J_{k-1}. As job j^*_k is now part of schedule $\mathcal{S}_{k,i,t}$, the delivery departure time of the added job j^*_k is set to time t. To compute the total cost and the feasibility of this temporary schedule, the effect of the integration of this new job to the schedule must be considered. This is carried out by computing the completion time of job j^*_k and the recalculation of the completion times for already scheduled jobs that are affected by the integration of job j^*_k. As the local cost optimality of canonical schedules is shown by Lemma 5.2.2.1, the SFDDHT-GRASP only constructs canonical schedules by application of Procedure 5.2.2.1 to compute the completion times. The backward computation of this procedure starts with the insertion of job j^*_k at position i and continues to calculate the new completion times for jobs at the preceding positions $i-1, \dots, 1$ of temporary schedule $\mathcal{S}_{k,i,t}$.

The objective function value (i.e, the return value of the greedy cost function in the SFDDHT-GRASP) of the temporary schedule, denoted as $V^c(\mathcal{S}_{k,i,t})$, is computed alongside the canonical scheduling procedure. For this, the objective value of the original schedule $\mathcal{S}_{k-1}$ is copied and updated by the changed holding costs for the jobs at positions $1, \dots, i-1$. As job j^*_k is newly added, its holding costs are simply added to $V^c(\mathcal{S}_{k,i,t})$. If job j^*_k is the sole job of customer $g^N(j)$ assigned to delivery departure time t, then the customer-specific transportation cost $c^T_{g^N(j)}$ is added.

The application of Procedure 5.2.2.1 may produce schedules that violate the release date constraints, since completions for already scheduled jobs may decrease due to the integration of job j^*_k. If a violation is detected during Procedure 5.2.2.1, then the procedure continues, but labels the construction move as infeasible.

6.2.3 Reducing the Set of Construction Moves

In step k, there are k alternative insertion positions and $|T^N_{j^*_k}|$ alternative pickup time assignments for the integration of job j^*_k. The number of considered alternative construction moves is thus bounded by $k \cdot |T^N_{j^*_k}|$. To speed up the execution of the construction phase, the set of construction moves is reduced by exclusion of some easily testable infeasible combinations. For each position $1, \dots, k$, the construction procedure determines a feasible subset of assignable delivery departure times for job j^*_k. As the following procedure is also used in the repair and improvement phase, the definitions are formulated in a generalized way for a schedule $\mathcal{S}$ with $|\mathcal{S}|$ jobs scheduled in order $j^{\mathcal{S}}_1, j^{\mathcal{S}}_2, \dots, j^{\mathcal{S}}_{|\mathcal{S}|}$, with $j^{\mathcal{S}}_i$ denoting the job at position i in schedule $\mathcal{S}$. Let $C^{\min}(\mathcal{S}, j, i)$ be the minimal completion time of scheduling a job $j \in N$ at position i given schedule $\mathcal{S}$. It is implied that job j is currently not scheduled in schedule $\mathcal{S}$. The minimal completion time is computed by scheduling the jobs $j^{\mathcal{S}}_1, \dots, j^{\mathcal{S}}_{i-1}, j$ in this order as early as possible. That is, job $j^{\mathcal{S}}_1$ completes at

$$C_{j^{\mathcal{S}}_1} = r_{j^{\mathcal{S}}_1} + p_{j^{\mathcal{S}}_1}.$$

Each job $j^{\mathcal{S}}_k$ with $k \in \{2, \dots, i-1\}$ is subsequently iteratively scheduled and completes at

$$C_{j^{\mathcal{S}}_k} = \max\{r_{j^{\mathcal{S}}_k}, C_{j^{\mathcal{S}}_{k-1}}\} + p_{j^{\mathcal{S}}_k}.$$

Lastly, job j completes at

$$C^{\min}(\mathcal{S}, j, i) = C_j = \max\{r_j, C_{j^{\mathcal{S}}_{i-1}}\} + p_j.$$

It follows that the earliest delivery departure time available for job j scheduled at position i is

$$D^e(\mathcal{S}, j, i) = \min\left\{t \in T^N_j, t \geqslant C^{\min}(\mathcal{S}, j, i)\right\}.$$

Furthermore, the latest delivery departure time time available for job j scheduled at position i is

$$D^l(\mathcal{S}, j, i) = \min\left\{\bar{d}_j, \min\left\{t^{\mathcal{S}}_o \,\middle|\, i \leqslant o \leqslant |\mathcal{S}|, g^N(j^{\mathcal{S}}_o) = g^N(j)\right\}\right\}.$$

The restrictions provided by $D^e(\mathcal{S}_k, j, i)$ and $D^l(\mathcal{S}_k, j, i)$ reduce the set of evaluated construction moves in each iteration k of the construction procedure to

$$CL = \left\{(i, t) \,\middle|\, i \in \{1, \dots, k\} \text{ and } t \in T^N_{j^*_k} \text{ with } D^e(\mathcal{S}_{k-1}, j^*_k, i) \leqslant t \leqslant D^l(\mathcal{S}_{k-1}, j^*_k, i)\right\},$$

which defines the CL for the SFDDHT-GRASP.

In the implementation of the SFDDHT-GRASP, the set of construction moves is determined iteratively, starting with all pairs (i, t) with position $i = 1$ in iteration 1, and continuing to evaluate all pairs (i, t) with $i = 2, \dots, k$. A full computation of all completion times of predecessors to derive $C^{\min}(\mathcal{S}, j, i)$ is therefore not necessary. In each iteration i, only the value $C_{j_i^{\mathcal{S}}} = \max\{r_{j_i^{\mathcal{S}}}, C_{j_{i-1}^{\mathcal{S}}}\} + p_{j_i^{\mathcal{S}}}$ with initial value $C_{j_0^{\mathcal{S}}} = 0$ must be computed. Afterward, this value is stored for the next iteration $i + 1$. The computation of the minimal completion time for job j in each iteration i is simply computed as

$$C^{\min}(\mathcal{S}, j, i) = \max\{r_j, C_{j_{i-1}^{\mathcal{S}}}\} + p_j.$$

6.2.4 Outline of the Construction Procedure

The selection of the construction move that extends schedule $\mathcal{S}_{k-1}$ to define schedule $\mathcal{S}_k$, is based on the definition of an RCL (a subset of the CL). Specifically, the candidate is randomly chosen from the subset of the *best* n^{RCL} construction moves according to the applied greedy function. The definition of this subset for each construction k, depends on whether at least one feasible construction move exists in the CL. Let n^{CL} be the number of moves in the CL, and let n_f^{CL} be the number of feasible moves in the CL. If $n_f^{CL} \neq 0$, then the subset comprises the $\min\{R, n_f^{CL}\}$ best feasible moves in terms of solution quality. Otherwise, if $n_f^{CL} = 0$ and $n^{CL} \neq 0$, then the subset comprises the $\min\{R, n^{CL}\}$ best, albeit infeasible, construction moves. In both cases, a candidate is drawn randomly from the reduced set, and extends the current schedule towards schedule $\mathcal{S}_k$. Due to the exclusion of assignable delivery departure times, the case $n^{CL} = 0$ may occur. No attempt is made to resolve this issue. Instead, the current GRASP iteration ends unsuccessfully, and the next iteration starts with an empty schedule. The entire construction procedure is outlined by Algorithm 6.4.

Algorithm 6.4 The Construction Phase of the SFDDHT-GRASP

```
Let $\mathcal{S}$ be a new solution;
for $k = 1$ to $n^N$ do
    CL $\leftarrow \left\{(i, t) \,\middle|\, i \in \{1, \dots, k\} \text{ and } t \in T^N_{j_k^*} \text{ with } D^e(\mathcal{S}_{k-1}, j_k^*, i) \leqslant t \leqslant D^l(\mathcal{S}_{k-1}, j_k^*, i)\right\}$;
    Evaluate all insertion pairs of CL;
    RCL $\leftarrow \{(i, t) \mid (i, t)$ is feasible and one of the $R$ best solutions$\}$
    if RCL $= \varnothing$ then
        RCL $\leftarrow \{(i, t) \in$ CL $\mid (i, t)$ is one of the $R$ best but infeasible solutions$\}$
    if RCL $= \varnothing$ then
        return $\mathcal{S}$                                ▷ solution is incomplete
    $(i, t) \leftarrow$ draw randomly from RCL
    $\mathcal{S} \leftarrow \mathcal{S}_{k,i,t}$ (insert $j_k^*$ at position $i$ with delivery departure date $t$);
return $\mathcal{S}$                  ▷ solution is complete but not necessarily feasible
```

Note. Adapted from “Finding efficient make-to-order production schedules” by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

6.3 Repair and Improvement Phase

The construction phase ends with the generation of a feasible solution, an infeasible solution, or an incomplete solution. The SFDDHT-GRASP makes no attempt at repairing incomplete solutions. Moreover, if no complete solution could be established in a construction attempt, then the current GRASP iteration ends, and a new solution is constructed from scratch. Hence, only complete solutions, which are either feasible or infeasible, enter the repair and improvement phase.

6.3.1 Reinsertion Moves

The repair and improvement phase uses a single move operator to repair and improve infeasible and feasible solutions, respectively. This move operator modifies a solution by moving a job j_i, located at position i, to a sequence position i' and replaces the currently assigned delivery departure time t with time t'. Such a move is defined by a quadruple (i, t, i', t') with $(i, t) \neq (i', t')$, and it consequently modifies either the position of a job, or the delivery departure time of a job, or both attributes.

Conceptually, the move operator is executed by removing job j_i from the schedule and reinserting this job at position i' with delivery departure time t'. Hence, a move carried out by this operator is referred to as a *reinsertion move*. Analogously to the previous section, a complete schedule, simply denoted by $\mathcal{S}$, is defined by a set of triples that define the positions and completion times of the scheduled jobs. As each completed schedule comprises $k = n^N$ triples, the subscript $k = n^N$ is omitted from the notation. By this simplification, a complete schedule is defined by

$$\mathcal{S} = \left\{(1, j_1^{\mathcal{S}}, t_1^{\mathcal{S}}), \ldots, (n^N, j_{n^N}^{\mathcal{S}}, t_{n^N}^{\mathcal{S}})\right\}.$$

A reinsertion move (i, t, i', t') defines a new modified solution

$$\mathcal{S}' = \left\{(1, j_1^{\mathcal{S}'}, t_1^{\mathcal{S}'}), \ldots, (n^N, j_{n^N}^{\mathcal{S}'}, t_{n^N}^{\mathcal{S}'})\right\}.$$

The elements of $\mathcal{S}'$ depend on whether the new position i' is greater than i. For $i' > i$, it holds that

$$\forall h = 1, \ldots, i-1, i'+1, i'+2, \ldots, n^N \colon j_h^{\mathcal{S}'} = j_h^{\mathcal{S}} \text{ and } t_h^{\mathcal{S}'} = t_h^{\mathcal{S}},$$

$$\forall h = i, \ldots, i'-1 \colon j_h^{\mathcal{S}'} = j_{h+1}^{\mathcal{S}} \text{ and } t_h^{\mathcal{S}'} = t_{h+1}^{\mathcal{S}} \text{ and}$$

$$j_{i'}^{\mathcal{S}'} = j_i^{\mathcal{S}} \text{ and } t_{i'}^{\mathcal{S}'} = t_i^{\mathcal{S}}.$$

In contrast, if $i' \leqslant i$ holds, then the new solution $\mathcal{S}'$ is defined as follows:

$$\forall h = 1, \ldots, i'-1, i+1, i+2, \ldots, n^N \colon j_h^{\mathcal{S}'} = j_h^{\mathcal{S}} \text{ and } t_h^{\mathcal{S}'} = t_h^{\mathcal{S}},$$

$$\forall h = i'+1, \ldots, i \colon j_h^{\mathcal{S}'} = j_{h-1}^{\mathcal{S}} \text{ and } t_h^{\mathcal{S}'} = t_{h-1}^{\mathcal{S}} \text{ and}$$

$$j_{i'}^{\mathcal{S}'} = j_i^{\mathcal{S}} \text{ and } t_{i'}^{\mathcal{S}'} = t_i^{\mathcal{S}}.$$

The completion times in the modified schedule are determined by Procedure 5.2.2.1 such that the resulting schedule is canonical. Since the solution $\mathcal{S}'$ is identical to solution $\mathcal{S}$ at positions $\max\{i', i\} + 1, \ldots, n^N$, only completion times for the lower indexed positions are newly computed. Alongside the computation of the completion times, the feasibility of $\mathcal{S}'$ is determined by checking for release date violations.

To evaluate a reinsertion move, the SFDDHT-GRASP uses *two* different evaluation functions: The first function $V^c(\mathcal{S})$, which also evaluates construction moves, returns the objective function value of solution $\mathcal{S}$, while the second function $V^f(\mathcal{S})$ measures the *degree of infeasibility* of a solution $\mathcal{S}$. The degree of infeasibility is defined as the total violation of the job-time windows in a solution $\mathcal{S}$, measured in TUs. Since solutions generated by the SFDDHT-GRASP never violate deadlines, this measure is defined as

$$V^f(\mathcal{S}) = \sum_{j \in N} \max\{0, r_j - S_j\}.$$

The repair and improvement phase executes the *hill-climbing* procedure, already outlined by procedure *LocalSearch* (Algorithm 6.3), for repairing and improving solutions. That is, the iterative process incrementally modifies the input solution to construct better solutions. The process halts if no improving modification is found. Specifically, the SFDDHT-GRASP uses the reinsertion operator to construct candidate solutions.

6.3.2 Reducing the Set of Reinsertion Moves

The repair and improvement stages exclude from full evaluation any moves that define detectable infeasible modifications to the schedule. In addition to the exclusion of assignments to delivery departure times, moves that lead to infeasible sequences are also excluded from full evaluation. The repair and improvement phase utilizes precomputed matrices M^{prec} and M^{pos}, which are also used by the Branch&Bound algorithm (see Section 5.4). The set of reinsertion moves $\mathcal{M}^r$, that are fully evaluated, comprises the quadruples (i, t, i', t') with $i, i' \in \{1, \dots, n^N\}$ and $t, t' \in T^N_{j^{\mathcal{S}}_i}$ that additionally satisfy each of the following conditions:

(a) The considered job at position i, is reinserted at a different position or is assigned to a different delivery departure time. That is, the reinsertion leads to the generation of a different schedule.

(b) The considered job at position i, is allowed at the reinsertion position i' and not prohibited by the position constraints given by matrix M^{pos}.

(c) The considered job at position i, is a feasible successor to the jobs at positions $1, \dots, i' - 1$ of schedule $\mathcal{S}'$, due to the precedence constraints stored in matrix M^{prec}.

(d) The considered job at position i, is a feasible predecessor to the jobs at positions $i' + 1, \dots, n^N$ of schedule $\mathcal{S}'$, due to the precedence constraints stored in matrix M^{prec}.

(e) The assigned delivery departure time t' is non-dominated according to the time bounds $D^e(\mathcal{S}^r, j^{\mathcal{S}}_i, i')$ and $D^l(\mathcal{S}^r, j^{\mathcal{S}}_i, i')$, with $\mathcal{S}^r$ denoting the schedule that is given by removing the considered job $j^{\mathcal{S}}_i$ from schedule $\mathcal{S}$.

Formally, for each valid reinsertion move $(i, t, i', t') \in \mathcal{M}^r$, the following holds:

(a) $(i, t) \neq (i', t')$ and

(b) $m^{pos}_{j^{\mathcal{S}}_i, i'} = 1$ and

(c) $\sum_{h=1}^{i'-1} m^{prec}_{j^{\mathcal{S}}_{i}, j^{\mathcal{S}}_{h}} = 0$ and

(d) $\sum_{h=i'+1}^{n^N} m^{prec}_{j^{\mathcal{S}}_{h}, j^{\mathcal{S}}_{i}} = 0$ and

(e) $D^e(\mathcal{S}^r, j^{\mathcal{S}}_i, i') \leqslant t' \leqslant D^l(\mathcal{S}^r, j^{\mathcal{S}}_i, i')$.

In each iteration of the hill-climbing procedure, the current solution is replaced by the best solution constructable from a reinsertion move, if and only if the move improves the current solution. The repair and improvement phase dynamically changes the active evaluation function, based on the feasibility status of the input solution. If the initial solution, generated in the construction phase, is infeasible, then all moves of $\mathcal{M}^r$ are evaluated by function $V^f(\mathcal{S})$. As long as infeasibility-reducing solutions can be constructed, the hill climber continues to replace the current solution by solutions with a lower degree of infeasibility until $V^f(\mathcal{S})$ equals *zero*. The repair procedure ends if a feasible solution is found or if the hill climber cannot eliminate the remaining release date violations. In the latter case, the repair and improvement phase ends unsuccessfully, and a new solution construction attempt is made by the SFDDHT-GRASP. In the first case, the evaluation function is switched to $V^c(\mathcal{S})$, and the feasible solution is subsequently improved until no *feasible* improvement move can be applied. Hence, once feasibility is established, the solution remains feasible during the remainder of the improvement phase. Algorithm 6.5 covers the repair and improvement phase.

Algorithm 6.5 The Repair and Improvement Procedure of the SFDDHT-GRASP

```
procedure IMPROVE($\mathcal{S}, \circ$)
    Let $V^\circ(\mathcal{S})$ with $\circ \in \{f, c\}$ denoting the active evaluation function.
    $improved \leftarrow true$;
    while $improved$ do
        $improved \leftarrow false$;
        $\mathcal{M}^r \leftarrow \{(i,t,i',t') \mid i,i' \in \{1,\dots,n^N\}$ and
        $t,t' \in T^N_{j^{\mathcal{S}}_i}$ and (a) - (e) hold for $(i,t,i',t')\}$;
        if $\circ = c$ then $M^r \leftarrow \{(i,t,i',t') \in \mathcal{M}^r \mid (i,t,i',t')$ is feasible$\}$;
        if $\mathcal{M}^r = \varnothing$ return $\mathcal{S}$;
        $(i^*,t^*,i'^*,t'^*) \leftarrow \arg\min\{V^\circ(apply(\mathcal{S},(i,t,i',t'))) \mid (i,t,i',t') \in M^r\}$;
        if $V^\circ(apply(\mathcal{S},(i^*,t^*,i'^*,t'^*))) < V^\circ(\mathcal{S})$ then
            $\mathcal{S} \leftarrow apply(\mathcal{S},(i^*,t^*,i'^*,t'^*))$;
            $improved \leftarrow true$
    return $\mathcal{S}$;
```

Note. Adapted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

6.4 Path relinking

Path relinking is an intensification strategy originally proposed by Glover (1997). This strategy explores the solution space along a trajectory between two constructed solutions. In Resende and Ribeiro's (2005) study, the enhancement of the GRASP procedure by path relinking is advertised as a highly promising extension. Hence, the SFDDHT-GRASP incorporates this strategy. In general, path relinking modifies an initial solution such that it increasingly resembles a given target solution. Therefore, each incremental modification defines an intermediate solution that potentially has a better objective function value than both input solutions.

Algorithm 6.6 The Path Relinking Procedure of the SFDDHT-GRASP

```
procedure PATH_RELINK($\mathcal{S}^I, \mathcal{S}^T$)
    $\mathcal{S}^{U*} \leftarrow \arg\min\{V^c(X) \mid X \in \{\mathcal{S}^I, \mathcal{S}^T\}\}$
    $\mathcal{S}^U \leftarrow \mathcal{S}^I$
    while $\Delta(\mathcal{S}^U, \mathcal{S}^T) > 1$ do
        $\mathcal{M}^p \leftarrow \{(i, t_i^{\mathcal{S}^U}, k, t_k^{\mathcal{S}^U}) \mid \forall i \in \{1, 2, \dots, n^N\},$
                $(i, j_i^{\mathcal{S}^U}, t_i^{\mathcal{S}^U}) \neq (i, j_i^{\mathcal{S}^T}, t_i^{\mathcal{S}^T})$ with $j_i^{\mathcal{S}^U} = j_k^{\mathcal{S}^T}\}$
        $m \leftarrow \arg\min\{V^c(\mathtt{apply}(\mathcal{S}^U, (i, t, i', t'))) \mid (i, t, i', t') \in \mathcal{M}^p\}$
        $\mathcal{S}^U \leftarrow \mathtt{apply}(\mathcal{S}^U, m)$
        if $V^c(\mathcal{S}^U) \leqslant V^c(\mathcal{S}^{U*})$ then
            $\mathcal{S}^{U*} = \mathcal{S}^U$
    return $\mathcal{S}^{U*}$
```

The SFDDHT-GRASP incorporates this concept (as shown by Algorithm 6.6) as follows. The procedure starts with an *initial solution* $\mathcal{S}^I$ and a *target solution* $\mathcal{S}^T$, the distance between these two solutions is expressed by the *symmetric difference* $\Delta(\mathcal{S}^I, \mathcal{S}^T)$. The symmetric difference is defined as the set of elements that are either in $\mathcal{S}^I$ or in $\mathcal{S}^T$, but not in both; that is, $\Delta(\mathcal{S}^I, \mathcal{S}^T) = \mathcal{S}^I \cup \mathcal{S}^T \setminus (\mathcal{S}^I \cap \mathcal{S}^T)$. The procedure iteratively modifies the initial solution $\mathcal{S}^I$ to reduce the size of the symmetric difference in each step. Modification is carried out by application of the reinsertion move operator described in the previous section. Given the triplets $(i, j_i^{\mathcal{S}^I}, t_i^{\mathcal{S}^I}) \neq (k, j_k^{\mathcal{S}^T}, t_k^{\mathcal{S}^T})$ with $j_i^{\mathcal{S}^I} = j_k^{\mathcal{S}^T}$, job $j_i^{\mathcal{S}^I}$ is removed from position i in solution $\mathcal{S}^I$ and reinserted at position k with delivery departure time $t_k^{\mathcal{S}^T}$. In each step, all possible moves are evaluated. If at least one move leads to a feasible solution, then the feasible move with the lowest cost is implemented; otherwise, regardless of the resulting infeasibility, the move with the lowest cost is implemented. The path relinking procedure stores the best-generated feasible solution along the trajectory, and the procedure ends if the next path relinking step results in an identical solution to $\mathcal{S}^T$. Path relinking returns the best-encountered solution, which includes the possible return of either initial solution $\mathcal{S}^I$ or target solution $\mathcal{S}^T$.

6.5 Elite Pool and Implementation

To provide a larger sampling of the solution space in a short time frame, the SFDDHT-GRASP distributes its workload over n^{thd} threads. The concept of the parallelized version is similar to the implementation proposed by Aiex et al. (2003). Each thread iteratively constructs, repairs, and improves solutions, asynchronously. The implementation stores the best-encountered n^e solutions, called *elites*, in an elite pool S^e. This elite pool is synchronized between all threads such that each GRASP procedure has access to the same set of elite solutions. In fact, each thread owns a separate local elite pool. Changes to the local pool are then communicated to the other threads. In the initial iterations the elite pool S^ϵ is filled with the first n^ϵ successfully generated feasible solutions. Once the pool is filled, the path relinking procedure is applied after the construction and the repair and improvement phases for each feasible solution. A feasible solution $\mathcal{S}$ is combined with each elite $\epsilon \in S^\epsilon$ once as the initial solution $\mathcal{S}^I$ and once as the target solution $\mathcal{S}^T$ (see Algorithm 6.7). The path relinking phase returns the best-encountered solution during its application. Since the resulting solution is not necessarily locally optimal, the improvement procedure with evaluation function V^c is applied to the best-generated solution. If it has a better objective value than one of the current elites, then it replaces the worst stored elite.

Algorithm 6.7 Elite Pool-Updating Procedure of the SFDDHT-GRASP

```
procedure UPDATE_POOL($\mathcal{S}, S^\epsilon$)
    $\mathcal{S}^* \leftarrow \mathcal{S}$
    for $\epsilon_i \in S^\epsilon$ do
        $P_1 \leftarrow$ path_relink($\mathcal{S}, \epsilon_i$);
        $P_2 \leftarrow$ path_relink($\epsilon_i, \mathcal{S}$);
        $\mathcal{S}^* \leftarrow \arg\min\{V^c(X) \mid X \in \{\mathcal{S}^*, P_1, P_2\}\}$;
    $\mathcal{S}^* \leftarrow$ IMPROVE($\mathcal{S}^*, c$)
    $\epsilon' = \arg\max\{V^c(X) \mid X \in S^\epsilon\}$                ▷ Worst elite
    if $V^c(\mathcal{S}^*) < V^c(\epsilon')$ then
        Replace $\epsilon'$ by $\mathcal{S}^*$ in $S^\epsilon$;
        Synchronize the local elite pool with all other elite pools
```

This update is initially only carried out for the local copy of the elite pool. To synchronize updates, a buffer data structure is shared between each thread. A solution that is accepted locally as an elite is immediately sent to this buffer. Each thread synchronizes its elite pool in the last step of a GRASP iteration. The thread that requests synchronization locks the buffer and implements the buffered changes locally. That is, each buffered solution is compared to the locally stored elites and replaces the worst elite if the objective value of the buffered solution is better. This acceptance test ensures that better local updates, which are not yet synchronized by all other threads, are not overwritten in the synchronization step.

Algorithm 6.8 Main Procedure of the SFDDHT-GRASP

```
procedure GRASP($n^{thd}$,$n^{iters}$,$t^{lim}$,$n^\epsilon$,$n^{RCL}$)
    $S^\epsilon = \emptyset$;
    $n \leftarrow n^{iters}/n^{thd}$              ▷ Assuming $n^{iters} \mod n^{thd} = 0$ for simplicity
    for $m = 1, ..., n^{thd}$ do
        EXECUTE THREAD($n, t^{lim}, n^\epsilon, S^\epsilon, n^{RCL}$)       ▷ Copy elite pool $S^\epsilon$ and call
    post optimize elite pool $S^\epsilon$           ▷ Apply Algorithm 6.6 for all pairs of elites
    return $\epsilon^{best} = \arg\min\{V^c(\epsilon_i \mid \epsilon_i \in S^\epsilon\}$
```

Note. Adapted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

The SFDDHT-GRASP algorithm is outlined in Algorithms 6.8 and 6.9. The termination of the algorithm is controlled by two parameters: First, the iteration limit n^{iters} defines the number of GRASP iterations to perform in total; second, time limit t^{lim} terminates the execution of the procedure after a period of time. The RCL utilized in the improvement procedure is limited by maximal size n^{RCL}. After all parallel GRASP applications finish execution, a final post-optimization step combines all pairs of elites via path relinking, and the best-encountered solution during the whole process is returned as the result.

Algorithm 6.9 GRASP Execution in a Thread of the SFDDHT-GRASP

```
procedure EXECUTE_THREAD($I$,$T^{\max}$, $\mathcal{E}^{\max}$,$\mathcal{E}$,$n^{\mathrm{RCL}}$)
    while Iteration limit $I$ or time limit $T^{\max}$ not reached do
        $\mathcal{S} \leftarrow \mathtt{construct}(\mathcal{S})$;                       ▷ Calling Algorithm 6.4
        if $\mathcal{S}$ is incomplete then goto next iteration;
        if $\mathcal{S}$ is infeasible then $\mathcal{S} \leftarrow$ IMPROVE($\mathcal{S}, f$);             ▷ Try to repair
        if $\mathcal{S}$ is infeasible then goto next iteration;                 ▷ Repair failed
        $\mathcal{S} \leftarrow$ IMPROVE($\mathcal{S}, c$);                            ▷ Calling Algorithm 6.5
        $\mathtt{update\_pool}(\mathcal{S}, \mathcal{E})$;
```

Note. Adapted from "Finding efficient make-to-order production schedules" by D. Bachtenkirch and S. Bock, 2021, submitted to the European Journal of Operational Research. Copyright 2021 by Elsevier. .

6.6 Conclusion

This chapter introduced the SFDDHT-GRASP that provides heuristic solutions for the SFDDHT. The developed algorithm is usable as a standalone solution method and acts as an initial solution (and upper bound) generation procedure for the SFDDHT-B&B. The described procedure heavily draws on results attained in Chapter 5 in terms of excluding infeasible or dominated solutions and efficient solution evaluation.

This chapter first provided a generic introduction to the GRASP concept in Section 6.1, and the remainder of the chapter was concerned with the SFDDHT-GRASP implementation. Section 6.2 described the solution construction phase of the approach, and Section 6.3 the subsequently executed repair and improvement phase. A major concern in developing the heuristic procedure was the generation of *feasible* solutions, which is a difficult task for instances with restrictive production time-windows. This task was handled by introducing an insertion order of the jobs to generate solutions iteratively and by a repair procedure that reduces time-window violations step by step through the introduced reinsertion operator. The SFDDHT-GRASP was extended by a path relinking procedure, which was described in Section 6.4, to obtain improved solutions with the help of previously found elite solutions. Lastly, Section 6.5 provided a description of the overarching implementation and interaction of the described components. To speed up the computation of a large number of GRASP iterations, the implementation used multiple threads that synchronize found elite solutions to perform individual path relinking operations.

Chapter 7

The Developed Real-Time Control Approach

This chapter describes a RTC approach that solves a *dynamic* extension of the SFDDHT, abbreviated as D-SFDDHT. The dynamic problem models an environment where unpredictable real-time events take place that force changes in the scheduled plans. In particular, the developed approach handles situations with dynamic job arrivals that occur during the schedule execution. The approach concentrates its efforts on this single source of dynamism for evaluation purposes. However, it is easily extendable to handle other dynamic events in practical implementations. This chapter broadens the research regarding IPODS-FD problems by introducing a real-time approach for this problem type. Apart from Liu and Hsu's 2015 application of priority rules in for a dynamic scheduling problem, the conducted literature review in Section 3.3 did not encounter dynamic problem formulations or approaches for IPODS-FDs. While this chapter describes an RTC approach specifically for the D-SFDDHT, the general framework is suitable for related applications.

Using the terminology introduced in Section 2.3, the approach described in this chapter to solve the dynamic problem D-SFDDHT is, in its basic form, a predictive-reactive scheduling approach, which executes schedule repairs and complete rescheduling by applying either a simple insertion heuristic or a meta-heuristic GRASP approach. As initial experiments revealed that the performance of the approach is highly dependent on utilizing machine time correctly, the approach is complemented by a pro-active procedure that controls the machine workload with predicted target values. The real-time approach reschedules periodically, although switching the system to an event-driven configuration is straightforward by stopping the optimization process on new job arrivals.

The remainder of the chapter is structured as follows. Section 7.1 describes the dynamic problem D-SFDDHT investigated in this dissertation, and Section 7.2 describes the implemented RTC approach framework that dynamically responds to new job arrivals. The required updating steps of the approach are detailed in Section 7.3, and the integration of new jobs into previously generated schedules is explained in Section 7.4. The proposed schedules by the RTC approach that minimize the total costs are likely to comprise extended idle periods on the machine, due to the optimality structure of SFD-

DHT schedules. This property may hinder the successful integration of new production requests, as machine capacity is left unused. To tackle this issue, Section 7.5 proposes workload-balancing procedures that force machine capacity usage to anticipate future job requests. Lastly, Section 7.6 summarizes this chapter's contents and discusses possible extensions of the described approach.

7.1 The Considered Dynamic Scheduling Problem

This section describes the dynamic scheduling problem D-SFDDHT that covers dynamically arriving jobs or orders[1]. The considered optimization problem is extended by a temporal component that changes the optimization process in the following notable aspect:

The (static) SFDDHT assumes that schedule computation and schedule execution take place sequentially. Specifically, the computation of an optimized schedule by the SFDDHT-B&B or the SFDDHT-GRASP takes place before the considered planning horizon of the production and distribution process execution. In contrast, the dynamic setting dictates simultaneous planning and execution. Therefore, the D-SFDDHT requires quick responses to new order requests by proposing updated schedules that integrate these new requests.

As a motivational example, consider a two-shift operation where production shuts down during the night and starts each morning. During the production day, several customers place orders for which the manufacturer considers processing. At the start time of the day's first shift, there is an order-backlog of previous days, planned for production and distribution on the current day. At the beginning of each day, the problem is to generate a schedule for the backlogged orders, which can be solved by computing a schedule during the night. Once the first shift starts, the workforce follows the generated schedule. Simultaneously to the execution, new orders arrive with the request to produce and transport the orders during the current production day. Note that it is reasonable to assume that an initial *vetting* phase takes place. Hence, the system typically processes orders that are likely (but not unquestionably) executable during the current work day. With the addition of dynamically arriving orders, the problem arises of quickly deciding whether integrating a new order into an executable schedule is possible and, if so, proposing a modified schedule that takes the already executed operations into account.

Moreover, consider the following assumptions about the dynamic setting:

- *Order requests placed by customers must arrive before a given (e.g., daily) deadline.* The D-SFDDHT considers a finite planning horizon to be optimized. Within this planning horizon, requests that arrive after a given deadline are not considered in the current, but rather in the subsequent planning horizon. A typical problem instance thus comprises several known orders at the beginning of the planning horizon, while the remaining orders arrive dynamically over time. Note that, theoretically, the planning horizon's length may span a large amount of time (e.g., weeks, months, or years);

[1] Note that in the context of D-SFDDHT, the terms job and order are used interchangeably. As the problem formulation allows rejecting customer orders, the term order might be a more natural fit.

this assumption hence does not limit the usefulness of the problem formulation. Therefore, the presented approach to solve the D-SFDDHT is generally applicable in continuous production environments. However, imposing a relatively short planning horizon with a backlog of orders aims at computationally evaluating the approach in Chapter 8.

- *A single source of dynamism, namely, the incoming order requests, exists in the model.* Therefore, all attributes, apart from the arrival times of orders, are expected to be deterministic. There exists no cancellation of orders by customers. Additionally, the delivery is assumed to be reliable, such that deliveries occur precisely at the pre-agreed times. Furthermore, it is generally expected that any decision made by an optimization approach for the D-SFDDHT is executed as planned without any deviations.

- *Dynamically incoming orders are subject to rejection.* As the D-SFDDHT considers hard deadlines for orders, there is a possibility that a new order cannot be accepted and fulfilled timely; in such cases, rejecting orders is allowed. The problem formulation assumes that order acceptance is beneficial from a business stand-point, since rejection may lead to contractual penalties imposed on the manufacturer or loss-of-goodwill on the customer's side. Order rejection is thus only allowed if the applied solution approach cannot feasibly integrate a new order request.

The Dynamic Problem: Formally, the dynamic problem formulation of the D-SFDDHT is equivalent to the static problem SFDDHT (see Chapter 4) with the exception of also modeling dynamically arriving orders and the possibility of rejecting orders. The formulation considers a finite planning horizon with start time 0 and end time t^T during which the schedule execution takes place. To distinguish between the dynamic and static versions, dynamic parameters and variables are accentuated by use of ($\breve{}$). In the considered finite planning horizon *orders*,

$$\breve{N} = \{1, 2, \dots, n^{\breve{N}}\}$$

arrive dynamically at arbitrary times. These orders are placed by *customers*

$$\breve{G} = \{1, 2, \dots, n^{\breve{G}}\}.$$

As in the static problem, a set of fixed delivery departure times $\breve{T}_g$ exists for each customer $g \in \breve{G}$ at which completed orders may commence with transportation.

Dynamic Orders: Each order $j \in \breve{N}$ has an *arrival time* $\breve{a}_j$, which marks the time at which order j first becomes known to the system. One distinguishes between two types of orders: orders known at the beginning of the planning horizon and dynamically arriving orders. The former type has arrival time $\breve{a}_j = 0$, while the latter type has arrival time $\breve{a}_j > 0$. For all intents and purposes, time 0 of the planning horizon marks the D-SFDDHT solving optimization approach's activation time. It must be noted that the problem formulation still includes job production release dates for which $\breve{r}_j \geqslant \breve{a}_j$ holds.

Note that given sufficient computational time before the beginning of the planning horizon, the D-SFDDHT is equivalent to SFDDHT if it holds that $\breve{a}_j = 0$ for each $j \in \breve{N}$; then, the static approaches can solve the problem without requiring any modifications. However, in the real-time context, this scenario is unlikely; therefore, the *set of revealed orders* at some time-point $\tau \in [0, t^T]$ is defined by

$$\breve{N}_\tau = \left\{ j \in \breve{N} \mid \breve{a}_j \leqslant \tau \right\}.$$

An optimization approach that handles the dynamic problem has only partial information on the job set $\breve{N}$ at a specific time. The number and characteristics of not-yet-revealed orders at a time $t < t^T$ is thus unknown to the approach.

Furthermore, order integration is not necessarily possible for all incoming requests; hence, for each order $j \in \breve{N}$, the decision variables are complemented by binary decision variables $\breve{X}_j$ that indicate whether an order is accepted ($\breve{X}_j = 1$) or rejected ($\breve{X}_j = 0$) by the optimization approach. Note that this decision is only made once for each order and does not change over the course of the planning horizon. Moreover, one distinguishes orders by their rejection status. At time τ, the *accepted orders* are

$$\breve{N}_\tau^+ = \left\{ j \in \breve{N}_\tau \mid \breve{X}_j = 1 \right\},$$

and the *rejected orders* are

$$\breve{N}_\tau^- = \left\{ j \in \breve{N}_\tau \mid \breve{X}_j = 0 \right\}.$$

Continuing with this time-dependent notation, one distinguishes accepted orders by their production and distribution status. The currently revealed but *unfulfilled orders* at time τ are

$$\breve{N}_\tau^u = \{ j \in \breve{N}_\tau^+ \mid \breve{D}_j > \tau \}.$$

At time τ, some of the unfulfilled orders are already processed (or in production) on the machine and wait for transport. These *unfulfilled processed orders* are defined as

$$\breve{N}_\tau^{u,p} = \left\{ j \in \breve{N}_\tau^u \mid \breve{S}_j \leqslant \tau \right\}.$$

Since the processing of these orders on the machine is fixed, only the departure time is subject to variable planning decisions at time τ. All other revealed orders that still await processing are therefore defined as

$$\breve{N}_\tau^{u,\text{-}} = \left\{ j \in \breve{N}_\tau^u \mid \breve{S}_j > \tau \right\}.$$

All orders for which the start of distribution has already taken place are no longer part of the problem.

Objective: The dynamic version considers two objectives. The *first objective* seeks to integrate as many orders as possible into a feasible schedule; that is, maximize

$$\breve{z}_1 = \sum_{j \in \breve{N}} \breve{X}_j.$$

The *second objective* is to schedule the integrated orders as efficiently as possible. The schedule's performance is measured by the sum of total holding and transportation costs of the accepted orders:

$$\check{z}_2 = \sum_{j\in\check{N},\check{X}_j=1} \check{c}_j^{H}\left(\check{D}_j - \check{C}_j\right) + \sum_{g\in\check{G}} \check{c}_g^{T}\left|\left\{\check{D}_j \mid j \in \check{N}_g, \check{X}_j = 1\right\}\right|.$$

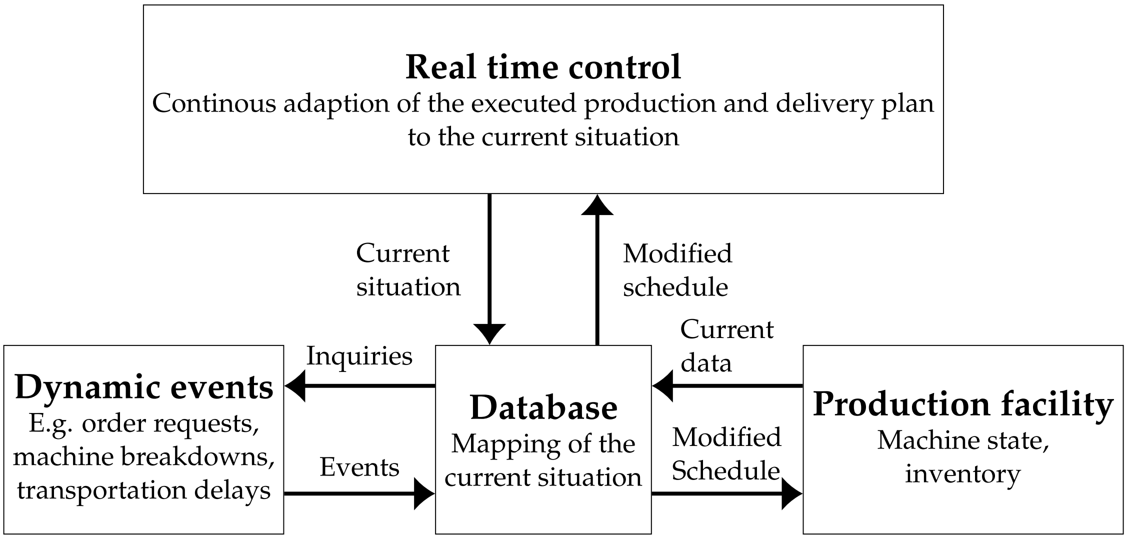
The optimization approach presented in the following sections assumes a hierarchical structuring of both objectives. That is, integrating order requests (maximizing $\check{z}_1$) is the *primary objective*, while optimizing the schedule costs (minimizing $\check{z}_2$) is the *secondary objective*. This structure models an *inflexible scenario* that assumes prohibitive order rejection costs that arise. As a result, whenever schedule integration of an order is possible, accepting an order request is seen as the superior decision. A more *flexible scenario* that allows for the rejection of orders to optimize scheduling costs is not considered, as it requires additional information regarding the exact cost of rejecting orders. Nevertheless, Section 8.7.4 evaluates the results of the proposed RTC approach for D-SFDDHT in light of flexible and inflexible environments.

7.2 The Real-Time Control Approach

In this section, the framework of the developed RTC approach that controls the previously described dynamic environment is presented. The approach generally adapts the approach

Figure 7.1

Information Flow in a Manufacturer's Real-time Control System


Real time control
Continous adaption of the executed production and delivery plan to the current situation
Current situation
Modified schedule
Current data
Inquiries
Dynamic events
E.g. order requests, machine breakdowns, transportation delays
Events
Database
Mapping of the current situation
Modified Schedule
Production facility
Machine state, inventory

Note. Adapted from Bock, S. (2010). Real-time control of freight forwarder transportation networks by integrating multimodal transport chains. *European Journal of Operational Research*, 200(3):733–746. Copyright 2010 by Elsevier.

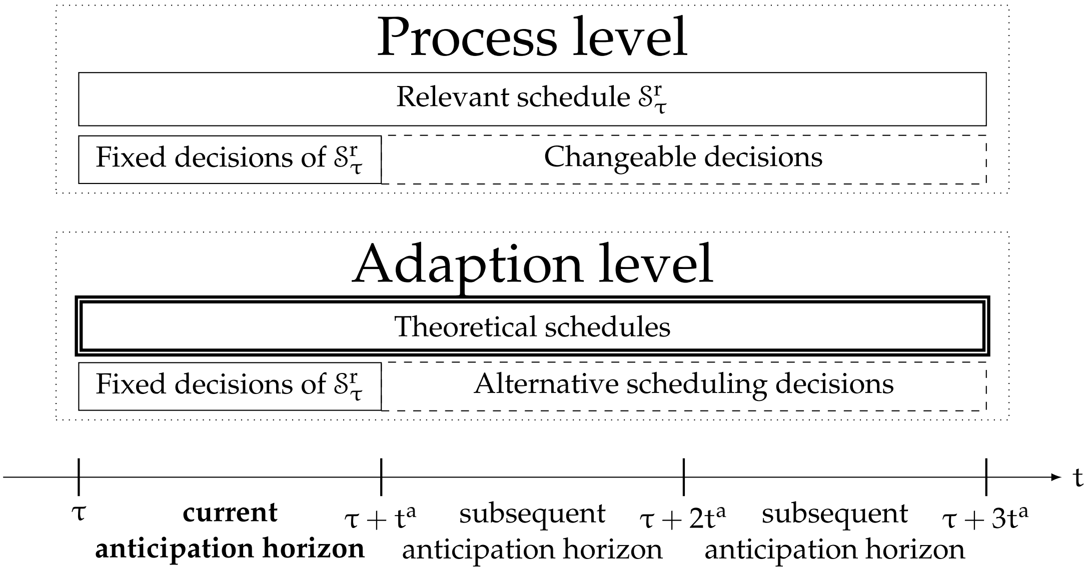
introduced by Bock (2004, 2010) to control freight forwarder transportation networks. A version of this concept is also applied by Ferrucci et al. (2013) and by Ferrucci and Bock (2014, 2015, 2016) for other types of dynamic pickup and delivery problems.

At all times, the approach proposes a feasible production and delivery schedule for the accepted and unfulfilled orders. An optimization algorithm continuously generates alternative schedules to the currently executed one. Moreover, the approach periodically tries to integrate new order requests into the existing schedule alternatives and checks whether a better alternative schedule can replace the currently proposed schedule in execution. The information flow for such a system is illustrated in Figure 7.1 (Bock, 2010). In line with the terminology introduced in Bock (2010), the proposed and currently executed schedule is called the *relevant schedule*. The description of the RTC approach's components assumes that the workforce executes the relevant schedule precisely as proposed. Therefore, the manufacturer carries out the *relevant schedule* without any deviations and delays. Conceptually, the approach works on two different levels, namely, the *process level* and the *adaption level*: On the process level, the manufacturer and distributor execute the relevant schedule. Meanwhile, on the adaption level, the RTC approach carries out computations to generate alternative schedules implementable in the future. These optimized schedules are called *theoretical schedules*.

The RTC approach employs a periodic rescheduling policy. It proposes a relevant schedule at periodic intervals of time, valid for one interval length duration (see Figure 7.2). These intervals of length t^a are called *anticipation horizons*, where duration t^a is a controllable parameter. At the end of each anticipation horizon (which is identical to the beginning of the next anticipation horizon), the approach proposes the relevant schedule $\mathcal{S}^r_\tau$. The subscript $\tau \in \{0, t^a, 2t^a, \ldots, t^T - t^a\}$ refers to the schedule that is proposed for the interval $[\tau, \tau + t^a]$. As time advances, new order requests arrive. These incoming order

Figure 7.2

State of the Real-Time Control Process for an Anticipation Horizon


Process level
Relevant schedule $\mathcal{S}^r_\tau$
Fixed decisions of $\mathcal{S}^r_\tau$
Changeable decisions
Adaption level
Theoretical schedules
Fixed decisions of $\mathcal{S}^r_\tau$
Alternative scheduling decisions
t
τ
current anticipation horizon
$\tau + t^a$
subsequent anticipation horizon
$\tau + 2t^a$
subsequent anticipation horizon
$\tau + 3t^a$

requests are initially buffered until time progresses to the end of the current anticipation horizon. Since the D-SFDDHT also considers hard-time window constraints, feasible integration of requested orders is not necessarily achievable. A request integration strategy is applied to accept or reject an incoming request (see Section 7.4), and the available time during an anticipation horizon $[\tau, \tau + t^a]$ is used to compute a schedule $\mathcal{S}^r_{\tau+t^a}$ that is proposed for the subsequent anticipation horizon $[\tau + t^a, \tau + 2t^a]$. For schedule generation, the current environment state (also called a *snapshot*) is captured and used to generate a static problem instance of the SFDDHT, which is then solved by a static solver. As the approach assumes the execution of decisions proposed by schedule $\mathcal{S}^r_\tau$ for the interval $[\tau, \tau + t^a]$, it simulates the decisions (production and delivery of orders) that lie in this interval. Affected orders are consequently not (or only partially) part of the generated static problem instance. After simulation and instance generation completes, the remaining time of the anticipation horizon is dedicated to generating an improved plan for interval $[\tau + t^a, \tau + 2t^a]$. The details of the approach are discussed in Sections 7.3 to 7.5.

7.3 Update Handling

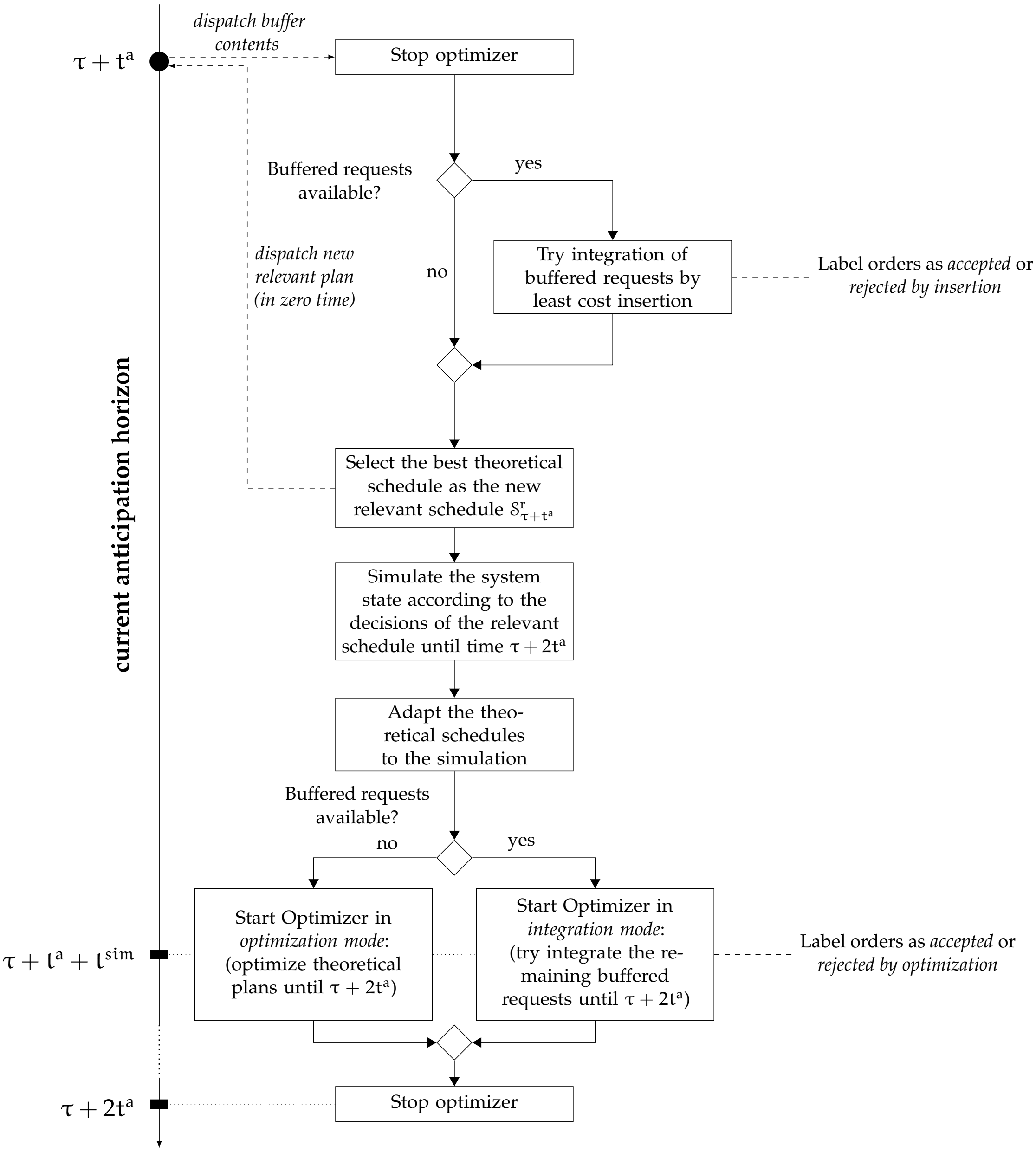
This section discusses the update handling of the proposed RTC approach at the end of each anticipation horizon (see Figure 7.3). The following descriptions detail the update handling initiated at time $\tau + t^a$, which is the end of the *current* anticipation horizon. The following steps are carried out in the presented order:

1. Termination of the optimization process that generated theoretical schedules for the subsequent anticipation horizon $[\tau + t^a, \tau + 2t^a]$.
2. Handling of buffered order requests with arrival during the current anticipation horizon $[\tau, \tau + t^a]$.
3. Selection and proposition of a new relevant schedule $\mathcal{S}^r_{\tau+t^a}$, which is to be executed during the subsequent anticipation horizon $[\tau + t^a, \tau + 2t^a]$.
4. Simulation of the decisions proposed by the relevant schedule $\mathcal{S}^r_{\tau+t^a}$ in the interval $[\tau + t^a, \tau + 2t^a]$. This step leads to the generation of a new static instance and the adaption of all stored theoretical schedules to the now executed events (up to time $\tau + t^a$) and the simulated events (up to time $\tau + 2t^a$).
5. Starting the optimization process that generates plans for the anticipation horizon $[\tau + 2t^a, \tau + 3t^a]$.

The first *three* steps are carried out at the end of the anticipation horizon $[\tau, \tau + t^a]$ at time $\tau + t^a$. For descriptive purposes, assume that these steps can be executed effectively in *zero time*. In practice, one must ensure that $\mathcal{S}^r_{\tau+t^a}$ is proposed at time $\tau + t^a$ at the latest such that decisions that immediately take place at this time (and were not present in the previously proposed plan) are executable. Step 4 starts at the beginning of the next anticipation horizon at $\tau + t^a$. Once the simulation completes, the RTC approach dedicates the subsequent anticipation horizon's remaining time to the optimization process (Step 5).

Figure 7.3

Flow Diagram of the Real-Time Control Approach for an Anticipation Horizon


dispatch buffer contents
$\tau + t^a$
Stop optimizer
Buffered requests available?
yes
no
Try integration of buffered requests by least cost insertion
Label orders as *accepted* or *rejected by insertion*
dispatch new relevant plan (in zero time)
current anticipation horizon
Select the best theoretical schedule as the new relevant schedule $\mathcal{S}^r_{\tau+t^a}$
Simulate the system state according to the decisions of the relevant schedule until time $\tau + 2t^a$
Adapt the theoretical schedules to the simulation
Buffered requests available?
no
yes
Start Optimizer in *optimization mode*: (optimize theoretical plans until $\tau + 2t^a$)
Start Optimizer in *integration mode*: (try integrate the remaining buffered requests until $\tau + 2t^a$)
Label orders as *accepted* or *rejected by optimization*
$\tau + t^a + t^{sim}$
$\tau + 2t^a$
Stop optimizer

7.3.1 Simulation of the Current Anticipation Horizon and Static Instance Generation

As a means of generating a static instance at time τ for the revealed orders $\breve{N}_\tau$, the decisions made according to the relevant schedule $\mathcal{S}^r_\tau$ in the interval $[\tau, \tau + t^a]$ are simulated. This simulation involves (a) all production decisions and (b) all delivery decisions that will be executed according to schedule $\mathcal{S}^r_\tau$. The simulation time is thus fast-forwarded to time $\tau + t^a$. It is important to note that preemption is not allowed in the model, and the processing on the machine hence occasionally fixes decisions beyond time-point $\tau + t^a$ for a job that started production earlier than $\tau + t^a$ but completes production later than $\tau + t^a$. The simulation and static instance generation performs as follows:

- The static instance omits already fulfilled orders and orders that will be fulfilled during the simulation interval $[\tau, \tau + t^a]$ according to schedule $\mathcal{S}^r_\tau$. That is, the set of jobs in the static instance in the interval $[\tau, \tau + t^a]$ is:

  $$N = \check{N}^u_\tau \setminus \left\{ j \in \check{N}^u_\tau \mid \check{C}_j \leqslant \check{D}_j \leqslant \tau + t^a \right\}.$$

  According to the relevant schedule, $\mathcal{S}^r_\tau$, the orders excluded from the set $\check{N}^u_\tau$ already departed or will depart during the current anticipation horizon; therefore, all decisions concerning these orders are already determined.

- The orders that are already produced or will be produced, partly or entirely, in the current anticipation horizon interval *but are not*, according to schedule $\mathcal{S}^r_\tau$, planned to be delivered during the current anticipation horizon are only *partially simulated*. The currently planned completion time $\check{C}_j$ is fixed, while the departure time is kept variable. These orders are referred to as *semi-fixed* orders. The set of semi-fixed orders N^f is defined as

  $$N^f = \check{N}^u_\tau \setminus \left\{ j \in \check{N}^u_\tau \mid \check{S}_j \leqslant \tau + t^a < \check{D}_j \right\}.$$

  To model this aspect, a feasible static schedule, generated during the current anticipation horizon, must fulfill the following constraints:

  $$C_j = \check{C}_j \text{ for each } j \in N^f.$$

  In contrast to the fixed completion times, the departure times D_j oblige no additional restrictions other than those already present in the static problem formulation.

- All non-semi-fixed (i.e., *free*) orders, ignoring release dates, can start production earliest at time $\tau + t^a$ or (if machine time is reserved further) after the latest finishing semi-fixed order on the machine. To avoid allocation of variable jobs before the simulation time window ends during scheduling generation, the release dates of the free jobs are adapted to the simulation. At time τ, the release dates of the free orders are determined as follows:

  $$r_j = \max\left\{ r_j, \tau + t^a, \max_{j \in N^f} \left\{ \check{C}_j \right\} \right\} \text{ for each } j \in N - N^f.$$

- The remaining parameter values concerning the orders in the static problem instance are identical to the parameter values of the dynamic problem instance:

  $$\forall j \in N\colon \quad \begin{aligned} p_j &= \check{p}_j, \\ c^H_j &= \check{c}^H_j, \\ \bar{d}_j &= \check{\bar{d}}_j. \end{aligned}$$

- As the current schedule $\mathcal{S}^r_\tau$ is simulated until time $\tau + t^a$, the available departure times in the static instance are reduced to $T_g = \left\{ t \in \check{T}_g \mid t > \tau + t^a \right\}$ for each customer $g \in \check{G}$. Equivalently, the procedure updates the transports I^T and I^T_g for each customer $g \in \check{G}$.

- The remaining customer information of the static problem is equivalent to the dynamic problem's information.

7.3.2 Solving Static Instances

The proposed real-time concept requires prompt responsiveness to dynamic events. Hence, the implemented approach uses the heuristic SFDDHT-GRASP approach (see Chapter 6) in favor of the exact B&B approach (see Chapter 5) to solve the static problem instances. At the start of each anticipation horizon, a static instance, generated according to Section 7.3.1, is forwarded to the SFDDHT-GRASP solver. The heuristic approach executes until time $\tau + t^a$ and returns several generated schedules. These schedules define the set of theoretical schedules at the end of an anticipation horizon, one of which will be selected as the relevant schedule $\mathcal{S}^r_{\tau+t^a}$. The existence of semi-fixed orders N^f requires slight modifications to the heuristic approach that are discussed next.

7.3.2.1 Construction Phase

As described in Section 6.2, the SFDDHT-GRASP adds jobs to a partial schedule in order of non-decreasing *time-window length* p^w_j, which is computed before the first iteration of the SFDDHT-GRASP starts. This ordering is modified such that the construction heuristic adds

- all free jobs $N - N^f$ in order of non-decreasing *time-window length* p^w_j first, and
- all semi-fixed jobs N^f in order of decreasing fixed completion time C^f_j second.

Semi-fixed orders complete a partial schedule by iterative insertion at the front of the schedule, since each semi-fixed order's schedule position in a complete schedule is fixed. The choice of the best move for a semi-fixed order reduces to selecting the most cost-effective departure date assignment. Note that the partial schedule after inserting all free orders can be infeasible. In particular, a free order $j \in N \setminus N^f$ with release date $r_j \geqslant \max\left\{C^f_j \mid j \in N^f\right\}$ may violate its release date such that $S_j < \max\left\{C^f_j \mid j \in N^f\right\}$ holds. That is, a free job occupies machine time allocated to the semi-fixed orders. Even for such schedules, semi-fixed orders are inserted into the schedule at their allocated times without triggering any rescheduling procedure. The approach ignores such violations because they are simultaneously handled by penalizing release date violations and corrected in the repair phase.

7.3.2.2 Repair, Improvement, Path Relinking

In the repair and improvement phases and the path-relinking procedure of the SFDDHT-GRASP, schedule modifications are carried out by reinsertion moves. The following modifications to the heuristic handle the semi-fixed jobs:

- No reinsertion move applied to a semi-fixed job $j \in N^f$ is allowed to change the jobs position. Therefore, the procedure constructs only moves that change the departure time of a semi-fixed job in the schedule.
- No reinsertion move applied to a free job is allowed to change the position of a free job to one of the fixed positions $1, \ldots, |N^f|$.

- The rescheduling procedure that generates canonical schedules ignores the semi-fixed orders. That is, the rescheduling procedure that optimizes completion times to minimize holding costs does not apply to the first $|N^f|$ positions.

7.3.2.3 Integration into the Real-Time Control Approach

With these changes implemented, the SFDDHT-GRASP is applied as described in Chapter 6. The RTC approach starts the execution of the SFDDHT-GRASP after simulation and static instance generation at the beginning of an anticipation horizon, and the procedure only halts upon receiving a termination signal from the RTC approach at the end of an anticipation horizon. No other termination criterion is active. During execution, the SFDDHT-GRASP generates a set of elite schedules, which form the set of theoretical schedules. If new order requests arrive, the RTC approach considers these theoretical schedules for integrating new requests. Thereafter, the approach implements the best theoretical schedule as the relevant schedule of the next anticipation horizon.

7.3.3 Adaption of the Stored Theoretical Schedules

The best theoretical schedule is chosen as the relevant schedule $\mathcal{S}^r_{\tau+t^a}$ at time $\tau + t^a$. The decisions made by this schedule are simulated up to the end of anticipation horizon $[\tau + t^a, \tau + 2t^a]$. In conjunction, the stored theoretical schedules (elites) are updated to follow the simulated decisions. In particular, the stored theoretical schedules are modified in two ways: First, the previously simulated and now executed events of the previous anticipation horizon $[\tau, \tau + t^a]$ are handled; second, the simulated decisions of plan $\mathcal{S}^r_{\tau+t^a}$ are adopted. Note that this modification also affects the theoretical schedule chosen as the relevant schedule. Specifically,

1. Each order that executed (departed from the manufacture) in the interval $[\tau, \tau + t^a]$ from the theoretical schedules is removed. As these orders were semi-fixed already, the remaining scheduling decisions are unaffected by the removal.

2. The simulated decisions in interval $[\tau + t^a, \tau + 2t^a]$ according to schedule $\mathcal{S}^r_{\tau+t^a}$ lead to two types of modifications. Each order that is planned to depart in interval $[\tau + t^a, \tau + 2t^a]$ is removed from each theoretical schedule. Note that removing such an order may result in a sub optimal schedule concerning the holding costs (i.e., non-canonical). Therefore the schedule is canonized after removal. Furthermore, an order that is planned to complete but not depart in interval $[\tau + t^a, \tau + 2t^a]$ is marked as semi-fixed in the now applicable static instance and each schedule is consequently modified to guarantee the semi-fixed order by simple positional changes of the relevant orders. Again, this triggers the rescheduling procedure to minimize holding costs after completing the modification.

It must be noted that these modifications may lead to infeasible schedules that are removed from the list of theoretical schedules. The remaining feasible theoretical schedules define the elites for the subsequent optimization process. Evidently, at least one schedule – the relevant schedule $\mathcal{S}^r_{\tau+t^a}$ – remains and is reused as an elite.

7.4 Integration of Order Requests

This section discusses the procedure executing two strategies that attempt to integrate new order requests into the planning process. Consider time-point $\tau + t^a$ that is the end of the current anticipation horizon. During the interval $[\tau, \tau + t^a]$, new order requests may have arrived. A buffer stores these requests until the end of the anticipation horizon, and the task of the integration procedure at time $\tau + t^a$ is to decide whether the requests will be accepted or rejected. The decisions is made by either generating a schedule that integrates the dynamic requests (resulting in order acceptance) or failing to find such a schedule (resulting in order rejection). At the present time $\tau + t^a$, the current relevant schedule is $\mathcal{S}^r_\tau$, which proposed decisions for the interval $[\tau, \tau + t^a]$ that were executed. During this time, the optimization procedure generated theoretical schedules $\mathcal{S}^t_{\tau,1}, \mathcal{S}^t_{\tau,2}, \dots, \mathcal{S}^t_{\tau,n^{\mathcal{S}}_\tau}$ that are now implementable as the subsequent relevant plan $\mathcal{S}^r_{\tau+t^a}$. The integration procedure attempts to insert the new order requests into these theoretical schedules.

The procedure handles the buffered orders in a first-come-first-served (FCFS) manner; that is, it considers the integration of buffered requests in order of their arrival times. Two strategies are employed to integrate these requests: The first strategy is called the *integration strategy*, which tries to integrate a request into an existing plan quickly by applying a least-cost insertion procedure, and the second strategy is called the *optimization strategy*, where the SFDDHT-GRASP attempts to find schedules that integrate the requested orders throughout the subsequent anticipation horizon. The procedure that implements the first strategy returns positive answers to order requests quickly if integration succeeds and delegates a final decision on the acceptance or rejection of the remaining (initially not accepted) requests to an integration procedure that executes during the subsequent anticipation horizon. The RTC approach's default strategy is to first apply the integration strategy to all incoming requests in order of arrival and then utilize the optimization strategy for all requests for which the integration strategy failed. The two strategies are presented in Algorithm 7.1 and Algorithm 7.2. As illustrated in Figure 7.3, the integration strategy procedure is immediately called after stopping the optimization process at the end of each anticipation horizon.

Algorithm 7.1 Insertion Strategy

```
Require: $S^\epsilon$ - the current pool of elite solutions generated by SFDDHT-GRASP
Require: $N^R_\tau$ - the indexes of buffered order requests
for $j \in N^R_\tau$ do
    Add job $j$ to the current static instance
    for $\epsilon \in S^\epsilon$ do
        Apply the best insertion move (see Section 6.2) for job $j$ to extend $\epsilon$
    if at least one $\epsilon \in S^\epsilon$ is feasible then
        Remove all infeasible elites from the elite pool $S^\epsilon$
    else
        Remove job $j$ from the current static instance
        Revert the addition of job $j$ to each elite $\epsilon \in S^\epsilon$
        Label job $j$ as *rejected by insertion*
```

Algorithm 7.1 labels each buffered job as either accepted or rejected by insertion. In the latter case the customer that requested the order must wait for a final decision on the order acceptance until the end of the next anticipation horizon.

Algorithm 7.2 Optimization Strategy

```
Require: $t^{opt}$ - the total available optimization time in the current anticipation horizon
Require: $S^{\epsilon}$ - the current pool of elite solutions generated by SFDDHT-GRASP
Require: $N_{\tau}^{R2}$ - the indexes of buffered order requests labeled rejected by insertion
$S'^{\epsilon} \leftarrow S^{\epsilon}$                                   ▷ copy the current elites
$S^{\epsilon} \leftarrow \varnothing$                                    ▷ clear the current elite pool
$t^{int} \leftarrow t^{opt}/|N_{\tau}^{R2}|$
for $j \in N_{\tau}^{R2}$ do
    Add job j to the current static instance
    Execute SFDDHT-GRASP with time-limit $t^{int}$
    if $S^{\epsilon} \neq \varnothing$ then
        Label job j as accepted
    else
        Remove job j from the current static instance
        Label job j as rejected
if $S^{\epsilon} = \varnothing$ then                        ▷ All order requests are rejected
    $S^{\epsilon} = S'^{\epsilon}$                          ▷ Restore the old set of elite solutions
```

The *optimization strategy* is only activated if at least one request is labeled *rejected by insertion*. During the next anticipation horizon, the computing power dedicated to optimization is instead used to find plans that integrate the initially rejected requests (Algorithm 7.2). If the optimization strategy leads to the acceptance of at least one additional order, then the best theoretical schedule is implemented as the subsequent relevant schedule. Otherwise, the set of theoretical plans that existed at the end of the previous anticipation horizon is restored. Note that if no additional job could be integrated during this phase, then the GRASP-heuristic cycled through the construction and repair phases without finding a feasible schedule. Consequently, the RTC-approach restores the stored elite solutions, then continues with potentially handling new buffered order requests and selects the best theoretical schedule as the new relevant schedule as usual. Following the notation introduced in Section 7.1 the acceptance variable $\check{X}_j$ of order $j \in \check{N}$ is set to 1 if either of the two strategies labels order j as accepted. If the optimization strategy fails to integrate an order $j \in \check{N}$, then the variable $\check{X}_j$ is set to 0.

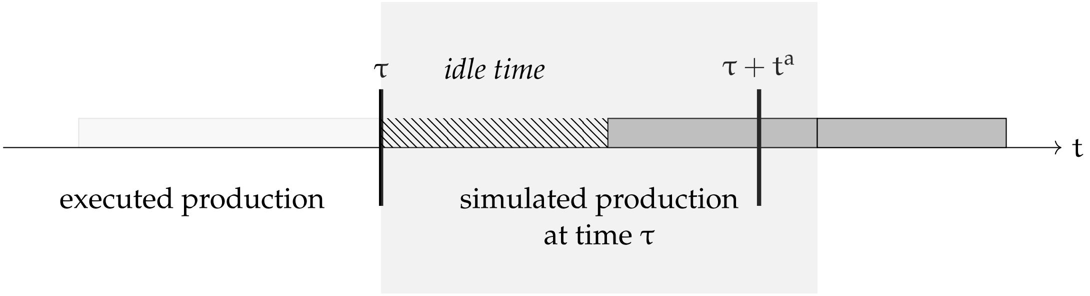
7.5 Workload-Balancing Techniques

Sections 7.2 to 7.4 describe the RTC approach with all components, which jointly form a fully functional procedure to solve D-SFDDHT instances. However, the application of this procedure may result in the under utilization of machine capacity that subsequently leads to situations wherein order requests are rejected, which could be avoided by proactively utilizing idle times on the machine. This section presents techniques that control the workload on the machine in real time to tackle the described issue.

The need to control the workload arises due to the uncertainty of the required machine time to process the unknown order requests within their hard processing time windows. As the RTC has no knowledge about future requests, the plan generation, avoids unnecessary holding costs by proposing canonical schedules. The typical structure of canonical schedules incorporates unforced idle time periods that minimize total holding costs. The proposed relevant schedule consequently suggests keeping the machine idle for extended periods of time, if the current situation does not require full machine utilization in the following anticipation horizon. Specifically, the relevant schedule at time $\mathcal{S}^r_\tau$ may propose leaving the machine idle for the whole, or parts of, interval $[\tau, \tau + t^a]$, as illustrated in Figure 7.4. As the schedule is carried out, the unused machine capacity is irretrievably lost.

Figure 7.4

Underutilized Machine Capacity in the Anticipation Horizon $[\tau, \tau + t^a]$



Note. The relevant schedule for anticipation horizon $[\tau, \tau + t^a]$ initially leaves the machine idle. As a new relevant schedule is first implemented at time $\tau + t^a$, the machine capacity is not utilized during execution of the RTC approach.

To integrate orders that arrive after time τ, the idle time could have been used by already released but delayed production jobs. This effect of wasted production capacity is cumulative. Idle-time adds up in periods with low processing requirements, and decreases available machine capacity in periods with high processing demand. The real-time approach is thus likely to reject many orders in environments with fluctuating workload requirements.

To overcome this issue, this section proposes methods that control the machine utilization. Specifically, the extended RTC approach forces schedule generation that utilizes idle periods to improve the chances to integrate dynamic order requests. Let these methods be known as *workload-balancing techniques*. In line with the terminology used by Xu and Xu (2015) (in the context of maintenance activities), the time interval between two idle states on a machine is called a *working interval*. The *workload* of a working interval is the total processing time of the jobs processed during this interval. As the SFDDHT considers only a single machine, the processing capacity or *maximum workload* up to time point $t \in \mathbb{N}_0$ on the machine is simply defined as

$$\check{w}^{\max}_t = t.$$

In the real-time context, the *committed workload* at time s, which is committed up to time t, is denoted as $\breve{w}^{c}_{s,t}$. The term *committed* is used to refer to already executed machine time allocations up to time s and simulated or planned processing up to time t, with $t \geqslant s$ and $s, t \in \mathbb{N}_0$. The committed workload is

$$\breve{w}^{c}_{s,t} = \sum_{j \in \breve{N}_s :\, \breve{S}_j < t} \min\left\{\breve{p}_j, t - \breve{S}_j\right\}.$$

The above calculation takes into account the processing of a job before t for a job that completes after t. The *ratio of committed workload* at time s up to time t is hence defined as

$$\breve{q}^{c}_{s,t} = \breve{w}^{c}_{s,t} / \breve{w}^{\max}_{t},$$

with $0 \leqslant \breve{q}^{c}_{s,t} \leqslant 1$. In the real-time context at time τ, the question arises as to whether the committed workload $\breve{w}^{c}_{\tau,\tau+t^{a}}$ (proposed by relevant schedule $\mathcal{S}^{r}_{\tau}$) is sufficiently large to generate enough *breathing room* to integrate future order requests that arrive after time τ. The following intrusive workload-balancing techniques' key idea is to force the static solver to utilize available machine capacity if this decision is likely to be necessary to accept future order requests. This approach relies on additional information about future requests to control the workload on the machine. Different methods are described in the following sections that *suggest* a workload to the system.

7.5.1 Expected Workload Balancing

To handle the workload-balancing problem, recognize the following additional assumption about the real-time environment: **The RTC system receives information about the *expected workload* in the current planning period.** This extension is motivated by the fact that historical data of past order requests and executed plans is likely available in a real-world setting. To research the effect of information available to the system, two different scenarios are studied that vary in the provided workload information's granularity:

1. *Total workload scenario.* In this scenario, the RTC receives the total (estimated) workload throughout the planning horizon. The assumption is that the current planning horizon's total workload is predictable by averaging or smoothing the workload of past planning periods.

2. *Interval workload scenario.* In this scenario, sectional workload information is available. Therefore, fluctuations in the required workload throughout the planning horizon are captured and provided to the RTC. The system periodically receives the estimated workload up to the end of the simulation interval, and the RTC consequently has information on intervals with lower and higher demand for machine time. As with the first scenario, it is reasonable to assume that the workload might follow a predictable pattern here and, is therefore anticipated with a high degree of certainty.

In both scenarios the RTC receives the time-indexed parameter $\breve{q}^{r}_{t}$ ($t \geqslant 0$), which describes the *recommended total workload ratio* up time t with $0 \leqslant \breve{q}^{r}_{t} \leqslant 1$ for each $t \in \{0, t^{a}, \ldots, T\}$ (i.e., the proportion of recommended working time compared to idle time on the machine until time t). Naturally, $\breve{q}^{r}_{0}$ is set to 0. At the end of each anticipation horizon at time $\tau + t^a$, the RTC receives value $\check{q}^r_{\tau+2t^a}$. This additional information is utilized to generate schedules during the interval $[\tau, \tau + t^a]$ that are implementable as the relevant plan $\mathcal{S}^r_{\tau+t^a}$, which decides on the machine utilization in interval $[\tau + t^a, \tau + 2t^a]$. In addition, in the total workload scenario, $\check{q}^r_t$ is constant over time; therefore, $\check{q}^r_t = \check{q}^r_{t-1}$ for each $t \in \{2, \dots, T\}$. In contrast, in the interval workload scenario, the predicted workload changes from one anticipation horizon to another.

7.5.2 Calculation of the Required Utilization

The RTC utilizes the suggested workload as follows: At the beginning of an anticipation horizon at time τ the relevant schedule $\mathcal{S}^r_\tau$ is implemented, and the decisions proposed by this schedule are simulated. Therefore, the workload in interval $[\tau, \tau + t^a]$ is committed. Additionally, as production that has begun on the machine is not preemptable, any processing that is proposed to start before $\tau + t^a$ and that ends after time $\tau + t^a$ is also committed.

In the interval $[\tau, \tau + t^a]$, theoretical schedules are generated that are candidates to be implemented for the subsequent anticipation horizon $[\tau + t^a, \tau + 2t^a]$. The decisions of one of these theoretical schedules will commit the workload at least up to time $\tau + 2t^a$. However, the workload $\check{w}^c_{\tau,\tau+2t^a}$ that will be committed by the relevant schedule $\mathcal{S}^p_{\tau+t^a}$ is still controllable by lowering or increasing the idle time on the machine in interval $[\tau + t^a, \tau + 2t^a]$. The earliest start of an idle period (considering only the executed and simulated machine time allocation) that may exist in this interval is

$$u^e_{\tau+t^a} = \min\{\max\{\tau + t^a, \max_{j \in \check{N}_\tau, \check{S}_j < \tau+2t^a} \{\check{C}_j\}\}, \tau + 2t^a\},$$

with latest end time

$$u^l_{\tau+t^a} = \tau + 2t^a.$$

The amount of controllable idle time $v^c_{\tau+t^a}$ is the interval length $l([u^e_{\tau+t^a}, u^l_{\tau+t^a}])$. If the recommended workload ratio $\check{q}^r_{\tau+2t^a}$ is greater than the committed workload ratio $q^c_{\tau,\tau+2t^a}$, then the difference $\Delta^w_{\tau+2t^a} = w^c_{\tau,\tau+2t^a} - w^r_{\tau+2t^a}$ with $w^r_{\tau+2t^a} = q^r_{\tau+2t^a} \cdot (\tau + 2t^a)$ describes the amount of additional processing required on the machine until time $\tau + 2t^a$ to emulate the recommended workload. To encourage the static solver to generate plans to approach or attain the recommended workload, the objective function considered by the SFDDHT-GRASP is extended to penalize schedules that do not utilize the available machine time in interval $[\tau + t^a, \tau + 2t^a]$. Specifically, let $v^r_{\tau+t^a}$ denote the additionally required workload in interval $[\tau_+ t^a, \tau + 2t^a]$ to increase the workload to the maximum value for which $w^c_{\tau,\tau+2t^a} \leqslant w^r_{\tau+2t^a}$ holds. This value is

$$v^r_{\tau+t^a} = \min\{\Delta^w_{\tau+2t^a}, l([u^e_{\tau+t^a}, u^l_{\tau+t^a}])\}.$$

The objective function V^c of the SFDDHT-GRASP is extended to $V^{c'}$:

$$V^{c'} = V^c + c^I \cdot \max\{v^r_{\tau+t^a} - P(u^e_{\tau+t^a}, u^l_{\tau+t^a}), 0\},$$

where $P(a, b)$ expresses the processing in TUs on the machine in interval $[a, b]$, and c^I is a sufficiently large cost value such that schedules with higher machine time utilization are prioritized by the SFDDHT-GRASP.

7.5.3 Rescheduling

Algorithm 7.3 Left-Shifting Scheduled Jobs

```
Require: The jobs in schedule $\mathcal{S}$ are indexed by their positions $1, \dots, n$
Require: $p^f$ - the index of the largest fixed position in the schedule
 1: $U \leftarrow \nu^c_{\tau+t^a}$
 2: for $j = p^f + 1$ to $n$ do
 3:     $C_j^{min} \leftarrow \{C_{j-1}^{min}, r_j\} + p_j$                ▷ Minimal feasible completion time
 4:     $S_j^{min} \leftarrow C_j^{min} - p_j$                            ▷ Minimal feasible start time
 5:     $S_j^{new} \leftarrow \max\{S_j^{min}, u^l_{\tau+t^a} - \min\{p_j, U\}\}$
 6:     if $S_j^{new} \geqslant S_j$ then
 7:         return                                               ▷ No further left-shifting possible.
 8:     $U \leftarrow U - S_j + S_j^{new}$
 9:     $S_j \leftarrow S_j^{new}$
10:     for $k = j - 1$ to $p^f + 1$ do
11:         $S_k \leftarrow S_{k+1} - p_{k+1}$
12:     if $U = 0$ then
13:         return                                               ▷ No further left-shifting required.
```

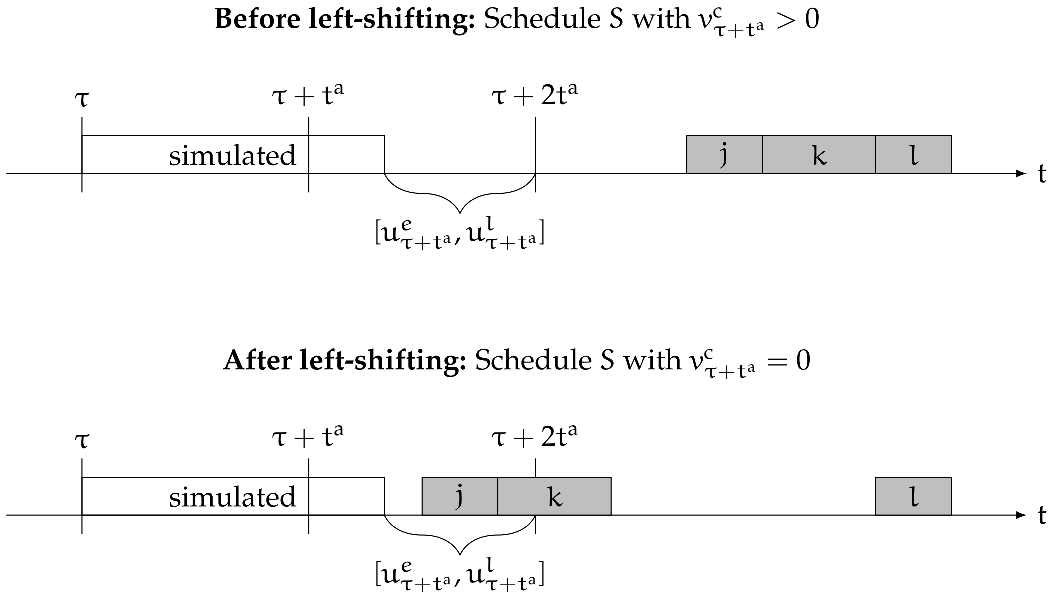
A rescheduling procedure (Algorithm 7.3) minimizes a schedule's idleness penalty in a (partial) schedule $\mathcal{S}$ during the application of the SFDDHT-GRASP. This procedure reschedules jobs without altering the job sequence or the date assignment of a solution and minimizes the total schedule cost under this constraint. The rescheduling procedure executes on each canonical schedule (which minimizes the holding costs) generated during the SFDDHT-GRASP. As idle-time is penalized only in a small time interval, the rescheduling procedure usually affects only a small subset of jobs in a schedule. A schedule generated by the SFDDHT-GRASP usually comprises some semi-fixed jobs at the first positions, followed by free jobs at the remaining positions. The rescheduling procedure starts at the first position in the schedule occupied by a free job and continues to evaluate subsequent positions. The procedure iteratively computes whether scheduling jobs up to the current position earlier reduces the total cost $V^{c'}$ by increasing $P(u^e_{\tau+t^a}, u^l_{\tau+t^a})$. This rescheduling activity is referred to as *left-shifting*. The procedure terminates when all penalized idle time has been eliminated or when no further reduction is possible.

The procedure increases the machine utilization in the penalized interval up to $\nu^c_{\tau+t^a}$ TUs by iteratively shifting allocated processing times in this interval. Specifically, the algorithm checks whether it is possible to produce the current job partially or entirely in interval $[u^e_{\tau+t^a}, u^l_{\tau+t^a}]$. If this modification is possible, then the procedure again schedules the previously rescheduled jobs earlier to accommodate the processing of an additional job in this interval. Each left-shift maximizes the additional time utilization in the interval while minimizing the increase in holding costs by this modification.

Note that the SFDDHT-GRASP allows for the violation of release dates in the construction and repair phases. Therefore, the scheduled jobs may violate their release dates. If the left-shifting procedure detects the violation of a job's release date, then the condition in Line 6 evaluates to *true*, and the procedure terminates. The final idle time violation cost after rescheduling is $c^i \cdot U$. The introduction of the new objective function $V^{c'}$ and

Figure 7.5

Illustration of the Applied Left-Shifting Procedure



Before left-shifting: Schedule S with $v^{c}_{\tau+t^a} > 0$

After left-shifting: Schedule S with $v^{c}_{\tau+t^a} = 0$

application of Algorithm 7.3 guides the SFDDHT-GRASP to generate schedules that properly utilize the workload as suggested. Figure 7.5 illustrates the left-shifting procedure. The original schedule does not produce jobs in interval $[u^{e}_{\tau+t^a}, u^{l}_{\tau+t^a}]$, since canonically scheduling all jobs results in maximized completion times. As $v^{c}_{\tau+t^a} > 0$ holds for the schedule, the Algorithm 7.3 reschedules the free jobs j and k to increase the workload until $\tau + 2t^a$.

7.6 Conclusion

This chapter presented the dynamic problem D-SFDDHT and an RTC approach for solving it. Section 7.1 described the dynamic problem with dynamic order arrivals. The uncertainty of future order requests hinders the proposing of high-quality schedules throughout the planning horizon. Additionally, this problem turned out to be challenging due to considering hard time-window constraints: Proposing scheduling decisions that minimize holding and transportation costs makes it more challenging to integrate future requests. Sections 7.2 to 7.5 presented a real-time approach that continuously optimizes schedules that take the current situation into account. The approach generates static instances throughout the planning horizon to optimize snapshots by the SFDDHT-GRASP and periodically propose updated schedules. The approach offers two methods to integrate newly arriving orders into the proposed schedule: First, a least-cost insertion method quickly updates the optimized schedules with new orders; if this method fails, then an optimization approach attempts to integrate the remaining orders. Section 7.5 described two workload-balancing techniques that proactively try to balance the machine utilization to increase the chance of successfully integrating new order requests.

The dynamic version D-SFDDHT includes a single source of dynamism; dynamically arriving orders. The model concentrates on this single dynamic aspect because including additional dynamic components would interfere with the analysis of the RTC approach's performance. Some other dynamic aspects that are present in practical applications are, for example, cancellation of orders, cancellation or delays of transports, changes in the production and delivery time windows of orders, and non-problem-specific occurrences such as machine breakdowns. However, the RTC framework can be easily adapted to handle these aspects by modifying the static instance generation and the schedule adaption procedures. This adaptability of the approach makes it a powerful tool for real-time optimization.

Section 7.5 introduced workload-balancing techniques without explaining how to obtain the workload suggestions specifically – this aspect is out of the scope of this dissertation but offers a starting point for further research. The two suggested methods, namely, *total workload* and *interval workload*, require analysis of historical data. The general assumption is that the currently optimized planning horizon is similar to past planning horizons; that is, a pattern of customer order requests is expected. To offer useful workload values, one must define and match existing historical patterns to the current request pattern. A recent approach by Ferrucci and Bock (2016) presents a proactive real-time routing approach with multiple request patterns. A similar methodology may be adapted to generate workload predictions and optimize the presented RTC approach.

The conclusion of this chapter also concludes the description of the optimization approaches for the SFDDHT and the D-SFDDHT. The computational evaluation of the developed approaches follows in Chapter 8.

Chapter 8

Evaluation of the Developed Optimization Approaches

Some results presented in this chapter also appear in the work of Bachtenkirch and Bock (2022), titled Finding efficient make-to-order production schedules, which is published in the European Journal of Operational Research. To avoid impairing the reading flow, citations do not reference this article continuously throughout this chapter. However, tables, figures, and algorithms that are taken from this paper are referenced as per usual.

This chapter presents the evaluation of the specifically developed optimization approaches from Chapters 5 to 7 that solve the static problem SFDDHT and the dynamic problem D-SFDDHT through computational experiments. The experiments aim to gain insights into the developed solution approaches' relative performance and their best-performing configurations. The analysis of the experimental results provides a comprehensive overview of the approaches' strengths and limitations and their applicability in various scenarios, and it answers the following questions:

- What problem instances are solvable in a reasonable time by the two exact optimization approaches CPLEX and the SFDDHT-B&B?
- Do differences in the solvability of instances exist based on the tightness of time windows and different settings of transportation cost values for the exact solution approaches?
- Which proposed lower bound performs best in terms of bound strength and runtime behavior?
- How does the heuristic SFDDHT-GRASP approach perform in terms of solution quality and speed?
- What roles do the repair, improvement, and path relinking phases play in the performance of the heuristic GRASP approach?

- Is the SFDDHT-GRASP approach suitable as an initial upper bound procedure for the B&B algorithm, a stand-alone heuristic, and an optimization procedure for the RTC approach?
- Can the RTC approach attain high-quality solutions without full information available, compared to solutions generated with access to the complete instance information?
- Are the workload-balancing methods able to improve the integration of dynamic orders? If so, at what additional scheduling cost?
- Do scenarios exist in which the workload-balancing methods are more useful than in others?

The optimization approaches SFDDHT-B&B and SFDDHT-GRASP were implemented in the programming language C++ to perform the experiments that answer the formulated research questions. Additionally, the SFDDHT-MILP model was formulated with the CPLEX Concert Technology Library for C++ so that the CPLEX Solver (v. 12.8) could solve the generated MILPs. The C++ source code was compiled with the G++ Compiler (v. 10.2.0) from the *GNU Compiler Collection (GCC)*. All tests were executed on *four* hardware-identical machines with 6 × 3.6GHz CPUs and 64GB RAM.

The SFDDHT-GRASP solver and the CPLEX solver were executed on 12 threads in parallel using Intel's Hyper-Threading technology, which allows the operating system to address two virtual cores for each of the six processor cores.

The SFDDHT-B&B implementation performed a single-threaded execution. Note that an efficient multi-threaded implementation of the B&B approach is not straightforward. A naive implementation that only parallelized the evaluation of child nodes did not attain convincing results due to the high cost of synchronization compared to the fast execution of the dominance and bounding procedures. In addition, as the experiments' results revealed, the limitations of the implemented B&B approach stem from the stored nodes and their memory requirement, not from the execution times.

All the conducted experiments use generated problem instances that vary several problem instance parameters. Section 8.1 starts by describing the specifically designed static and dynamic problem instance generators and the methods to establish the recommended workload ratios for the RTC approach. The subsequent sections describe the computational experiments conducted on these generated instances. This chapter organizes the description of the computational study as follows:

Computational study for the SFDDHT:

1. Setup of the computational experiments for the approaches CPLEX, SFDDHT-B&B, and SFDDHT-GRASP that solve the SFDDHT; (Section 8.2)
2. Comparison of the *three* different lower bounds (see Sections 5.7.1, 5.7.3, and 5.7.4) in terms of bound strength and computational complexity; (Section 8.3)

3. Evaluation of the performance of the SFDDHT-B&B with different bound configurations and the CPLEX solver; (Section 8.4)

4. Evaluation of the SFDDHT-GRASP's performance in comparison to the exact solution approaches, and analysis of different intensification configurations' impact on the solution quality. (Section 8.5)

Computational study for the D-SFDDHT:

1. Setup of the computational experiments for the RTC approach solving the D-SFDDHT; (Section 8.6)

2. Comparison of workload-balancing methods for the RTC approach in terms of schedule costs compared to statically generated schedules and dynamic order acceptance success rate; (Section 8.7)

3. Analysis of the impact of varying the anticipation horizon length on the performance of the overall system; (Section 8.8)

4. Analysis of the approach's consistency in terms of solution quality for different proportions of dynamic orders throughout the planning horizon; (Section 8.9)

5. Analysis of the effect on solution quality by omitting the optimization process while relying on a best-fit insertion procedure, and analysis of the more elaborate integration procedure's success compared to the faster, but less elaborate, insertion method. (Section 8.10)

8.1 Instance Generation

This section describes instance generation procedures that enable the experiments in the subsequent sections. Since the SFDDHT and D-SFDDHT problems are novel problem formulations, and no suitable benchmark instances of related problems are published, specifically designed and implemented instance generation procedures are proposed.

Section 8.1.1 describes the generator for static problem instances that sets the parameters of the SFDDHT introduced in Section 4.2. To evaluate the dynamic approach's performance for problem D-SFDDHT, a specifically developed procedure extends the static instances by adding arrival times. Section 8.1.2 describes the implemented modification procedure. Furthermore, Section 8.1.3 proposes different methods to generate workload predictions for the real-time setting.

8.1.1 Static Instance Generator

The static instance generator constructs SFDDHT instances by specifying the parameters of Table 8.1. It accepts a set of configurable parameters that specify the instance characteristics from which it draws the exact parameter values by using a random number generator. Specifically, the instance generator uses an implementation of the *Mersenne*

Table 8.1

Data of a Static Instance

Parameter	Brief description
N/n^N	Jobs
G/n^G	Customers
$\forall j \in N\colon p_j$	Processing times (in TU)
$\forall j \in N\colon c_j^H$	Holding costs (in MU)
$\forall j \in N\colon r_j$	Release dates (in TU)
$\forall j \in N\colon \bar{d}_j$	Deadlines (in TU)
$\forall g \in G\colon N_g$*	Job-customer allocation
$\forall g \in G\colon T_g$	Departure times (in TU)
$\forall g \in G\colon c_g^T$	Transportation costs (in MU)

* $\bigcup\limits_{g \in G} N_g = N, \forall h, i \in G, h \neq i, N_h \cup N_i = \varnothing$

Twister generator (see Matsumoto and Nishimura, 1998 for the original description of this random number generator) of 64-bit numbers with a state size of 19,937 bits, supplied with the used C++ standard library implementation (libstdc++).

The instance generator expects the following three input parameters that determine the *size* of a problem instance to generate:

- The number of jobs n^N;
- The number of customers n^G;
- The number of departures $n_g^T = |T_g|$ for each customer $g \in G$. This number is identical for each customer.

From an initial parameter setting of the three size parameters, the generator determines the job-customer allocation in two phases: First, each customer $g \in G$ is assigned a single job; second, an iterative procedure assigns the remaining jobs to customers by successively drawing the customer for a still unassigned job from the discrete uniform distribution $\mathcal{U}(1, n^G)$. After the procedure completes, each customer-job set N_g comprises at least one job. The *processing time* of each job $j \in N$ is a uniformly distributed integer in the interval $p_j \sim \mathcal{U}(1, 100)$ and has *holding cost* value $c_j^H \sim \mathcal{U}(1, 10)$. The customer-dependent *transportation costs* depend on choosing either a low or high transportation cost setting, and the costs are chosen from either a low-valued distribution ($c_g^T \sim \mathcal{U}(100,\ 500)$) or a high-valued distribution ($c_g^T \sim \mathcal{U}(500,\ 2{,}500)$) for each customer $g \in G$.

After generating these parameters, the generator constructs *job release dates* as follows: The sum of processing times of the generated jobs is $P(N) = \sum_{j \in N} p_j$. The release date for each job $j \in N$ is drawn from the distribution $\mathcal{U}[0, r^{ub}]$ with r^{ub} being randomly generated from distribution $\mathcal{U}(0.75P(N), P(N))$.

Afterward, the *deadlines* are generated by constructing a non-delay schedule (an ERD schedule) by application of the ERD rule (see Section 5.4.2.1). Let the job completion times and makespan of this schedule be C_j^{erd} for each $j \in N$ and M^{erd}, respectively. Provisional deadlines $\bar{d}^p_j$ for each $j \in N$ are initially drawn from the distribution $\mathcal{U}[C^{erd}_j, M^{erd} \cdot \alpha]$, where α is a configurable parameter that controls the tightness of the job time windows. In the experiments, $\alpha = 0.8$ is chosen for instances with *tight* time windows, and $\alpha = 1.2$ for instances with *relaxed* time windows. Thereafter, the departure times for each customer $g \in G$ are generated within the interval

$$I_g = \left[\min\left\{C^{erd}_j \mid j \in N_g\right\}, \max\left\{\bar{d}^p_j \mid j \in N_g\right\}\right] \quad \forall g \in G.$$

The latest departure time t_{g,n^T_g} is set to the largest value of interval I_g, whereas the remaining $n^T_g - 1$ departure dates are distributed evenly throughout this interval; this distribution defines the departure times T_g for each customer $g \in G$. As the provisional deadlines do not necessarily align with the generated departure times, the actual deadline $\bar{d}_j$ for each job $j \in N$ is set to the nearest departure time with a value equal to or greater than $\bar{d}^p_j$:

$$\bar{d}_j = \min_{t \in T_{g^N(j)}} \left\{t \mid t \geqslant d^p_j\right\}.$$

Note that the generation procedure does not necessarily produce feasibly solvable instances. Therefore, the generator accepts a generated instance only if the ERD schedule defines a feasible production schedule for this instance. This post generation step concludes the description of the static instance generator.

Table 8.2

Parameter Configurations for the Generated Static Instances

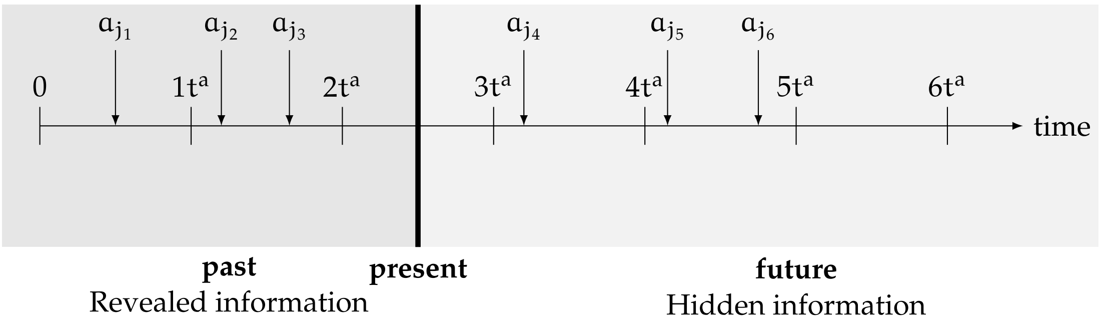
Set	Size	Transportation costs	Time windows	$\mathbf{n}^N$	$\mathbf{n}^G$	$\mathbf{n}^T_g$	#
SLT	Small	Low	Tight	$\{8, 10, 12, 14, 16, 18\}$	$\{2, 3, 4\}$	$\{2, 4, 6\}$	162
SLR			Relaxed				162
SHT		High	Tight				162
SHR			Relaxed				162
LLT	Large	Low	Tight	$\{20, 30, 40, 50\}$	$\{2, 4, 6\}$	$\{4, 6, 8, 10\}$	144
LLR			Relaxed				144
LHT		High	Tight				144
LHR			Relaxed				144
						Total	1224

#: Total number of instances for each set

For the computational experiments in this study, the generator produced a total of 1224 instances with the parameter configurations listed in Table 8.2. The analysis presented in the subsequent sections groups instances with similar parameters into instance sets. The following naming scheme is used throughout the discussion of the experimental results: An instance is either **S**mall or **L**arge, has **L**ow or **H**igh transportation costs, and has **R**elaxed or **T**ight time windows. Combining these distinctions yields eight instance sets: SLR, SLT, SHR, SHT, LLR, LLT, LHR, and LHT. For each identical parameter combination, *three* different instances (due to application of the random number generator) were generated.

Figure 8.1

Information Revelation while Simulating a Dynamic Problem Instance



Therefore, there are 4 × 162 instances in the small sets and 4 × 144 instances in the large sets.

8.1.2 Dynamic Instance Generator

The developed procedure to generate dynamic problem instances for the D-SFDDHT extends the static problem instances of Section 8.1.1 such that statically generated solutions constitute benchmarks for the dynamically generated schedules. A dynamic instance represents the input data from a real-world system sent to the RTC approach to generate optimized schedules dynamically. Concretely, time-dependent information – the order arrival times – supplements a static problem instance; that is, the dynamic instance generation procedure adds an arrival time a_j parameter for each job $j \in N$.

Over time, the RTC approach receives portions of the generated dynamic instances' information. At a given time t during the execution of the RTC process, all jobs with arrival time $a_j \leqslant t$ are visible, while the implementation hides the remaining jobs from the approach. The other instance data that concerns the customers and departure times is immediately available for the RTC approach. Figure 8.1 illustrates the described information reveal that concerns the dynamically arriving orders.

The RTC approach processes the dynamic problem instances in real time; hence, the parameters defined in the abstract measure of time (the TU) of the generated instances are mapped to a real-life duration. This mapping is not part of the dynamic instance generation procedure. Instead, the implementation of the RTC approach allows for control of the specific mapping by setting initial configuration parameters. For example, the RTC approach may execute with a setting that maps one TU to 100 milliseconds. This setting controls the anticipation horizon length t^a and the time available to optimize plans within each anticipation horizon.

The D-SFDDHT environment distinguishes between two types of jobs: jobs already known when starting the optimization and jobs that arrive dynamically during the execution of the RTC approach. The generation procedure partitions the jobs N of a static instance into two subsets; the first subset $\check{N}^0 \subseteq N$ comprises the jobs known in advance, while the remaining jobs are $\check{N}^d = N \setminus \check{N}^0$. For the experiments, two different scenarios are considered: The first scenario considers instances with half of the jobs known, and the second scenario reduces this quantity to approximately a quarter of jobs. In both scenarios, the most urgent jobs form the set $\check{N}^0$. Specifically, $\check{N}^0$ comprises jobs with the earliest deadline values such that no dynamic job has an earlier deadline than an initially known one.

In a dynamic instance, all arrival times a_j of the known jobs $\check{N}^0$ have the value zero. For the remaining jobs the generation procedure chooses the arrival times for dynamically arriving jobs $\check{N}^d$ such that the RTC approach (theoretically) has a chance of generating the same schedules as the static approaches. That is, the generator guarantees that the RTC registers a dynamically arriving job before its start time in the best-found static schedule such that the RTC approach can schedule a dynamic job within the statically determined processing interval on the machine. Additionally, the generator ensures that a job's arrival time takes place on the job release date at the latest. Arrival times were consequently generated by considering the start times S^*_j with $j \in \check{N}^d$ of the best-found schedule s^* for static instance x by drawing from a discrete uniform distribution:

$$\check{a}_j \sim \mathcal{U}\left[0, \min\left\{r_j, S^*_j - t^a\right\}\right]$$

for each job $j \in \check{N}^d$. In the above expression, the parameter t^a (the anticipation horizon length) ensures that a job j arrives before the occurrence of anticipation horizon interval in which the start time S^*_j lies. Therefore, job j can be integrated into the schedule and theoretically start at time S^*_j in the dynamic schedule. In all the experiments, the anticipation length is $t^a = 20$ TUs long.

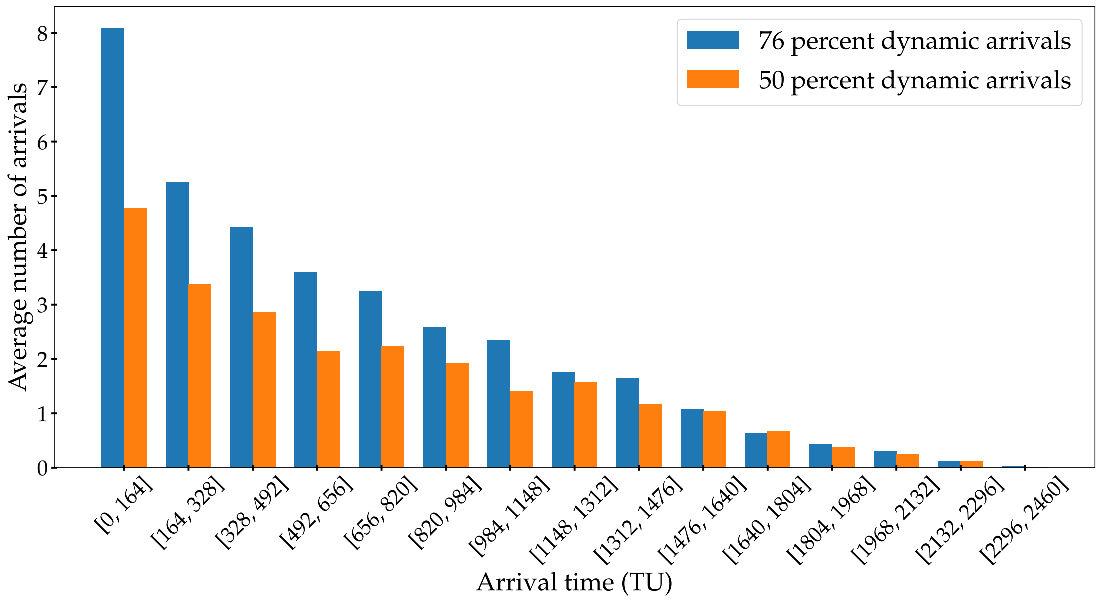
In this study, to conduct the experiments for the RTC approach, the generator constructed dynamic instance from all large static instances with 50 jobs. Overall, it generated 144 instances with 50% of dynamic orders (25 jobs) and 144 instances, with 76% of orders (38 jobs) dynamically arriving. Figure 8.2 illustrates the resulting distribution of dynamic arrival times for both instance sets. The dynamic arrivals tended to occur at the beginning of the planning horizon, while later arrivals were less frequent.

8.1.3 Recommended Workloads

To apply the workload-balancing methods discussed in Section 7.5, the RTC approach requires workload recommendations for the planning horizon to optimize. The generated static instances and corresponding statically generated schedules provide the information to generate the recommendation values. The experiments include tests for the two proposed methods, namely, total and interval workload, each with an *omniscient* and a *predictive* setting. On the one hand, the omniscient settings act as benchmarks for the predictive settings. Here, workload estimates stem from the best-found schedule to a dynamic instance's corresponding static instance for the omniscient settings. On the other hand, the predictive settings gather estimates from schedules to comparable instances. The subsequent sections discuss the *four* implemented and tested workload-balancing variations.

Figure 8.2

Arrival Time Distribution of the Dynamic Instances for both Scenarios



8.1.3.1 Omniscient Total Workload-Balancing

The omniscient total workload-balancing (WBOT) setting provides the RTC approach with a constant total workload ratio. This value is calculated from a high-quality statically generated feasible schedule. The setting stimulates the RTC approach to mimic this schedule in terms of workload utilization. In the computational experiments, the omniscient total workload value for a given dynamic instance is computed from the best-found schedule $s^*(x)$ for the corresponding static instance x. This value results from dividing the total processing times $P(x)$ of all jobs of instance x by the makespan of the best-found schedule $M(s^*(x))$. Hence, the predicted workload ratio that remains constant during the execution of the RTC approach is

$$\forall t \in \{t^a, 2t^a, \ldots, t^T\}\colon \check{q}^r_t = \frac{P(x)}{M(s^*(x))}.$$

Note that the best-known schedule was either calculated by the SFDDHT-B&B algorithm or the SFDDHT-GRASP. Both algorithms exclusively generate canonical schedules where the maximum completion time is also the latest departure time of a job. Table 8.3 displays statistics for the total workload recommendations generated from the best-found solutions. On average, the recommended workload ratio is 81% for instances with relaxed time windows and about 97% for instances with tight time windows.

8.1.3.2 Predictive Total Workload-Balancing

The predictive total workload-balancing (WBPT) setting is more realistic than its omniscient counterpart. This workload estimate was generated for a dynamic instance without detailed data of the corresponding static schedule. Instead, it was assumed that the

Table 8.3

Total Workload Values Recommended to the RTC Approach

Set	Min	Mean	Max	SD
Low / Relaxed	0.76	0.81	0.84	0.03
High / Relaxed	0.76	0.82	0.88	0.03
Low / Tight	0.91	0.97	1.00	0.03
High / Tight	0.91	0.97	1.00	0.03
Total	0.76	0.89	1.00	0.08

Min = minimum total workload; Mean = mean total workload; Max = maximum total workload; SD = average standard deviation

planning horizon's events of a dynamic instance will follow a predictable pattern for which a workload prediction is available. This prediction was calculated from best-found solutions to static instances related to the static instance from which the dynamic instance stems. Specifically, in the generated test set, a static instance was defined as *related to* another static instance if the transportation cost and time-window generation settings are identical. Since the real-time experiments use only 50 job instances, there are 4 × 36 related instances in the dynamic instance set. The dynamic instance's expected workload ratio was computed by averaging the expected workload ratios of the omniscient setting of its 35 related instances $X^r(x)$ of an instance x. The recommended workload ratios are consequently

$$\forall t \in \{t^a, 2t^a, \ldots, t^T\}\colon\ \breve{q}^r_t = \frac{1}{|X^R(x)|} \sum_{x' \in X^R(x)} \frac{P(x')}{M(s^*(x'))}$$

for all instances. Note that the above calculation is independent of the proportion of dynamically arriving jobs.

8.1.3.3 Omniscient Interval Workload-Balancing

The omniscient interval workload-balancing (WBOI) setting provides the RTC with a new expected workload ratio at the end of each anticipation horizon. Similarly to the omniscient total workload setting, the omniscient interval workload setting calculates the expected workload ratio from the best-found solution to a corresponding static instance. The workload ratio is calculated for all relevant multiples of t^a TUs such that a workload suggestion is available for each RTC update step. Equivalent to the computation of the committed workload $\breve{w}^c_{s,t}$, introduced in Section 7.5, the workload of a schedule at time $t \in \mathbb{Z}$ is defined as

$$w_t(s^*(x)) = \sum_{j \in N\,:\, S_j < t} \min\left\{p_j, t - S_j\right\}.$$

The recommended workload ratios are calculated as follows:

$$\forall t \in \{t^a, 2t^a, \ldots, t^T\}\colon \check{q}^r_t = \frac{w_t(s^*(x))}{t}.$$

8.1.3.4 Predictive Interval Workload-Balancing

The predictive interval workload-balancing (WBPI) setting averages the WBOI values of related instances for each dynamic instance. The calculation is carried out as follows:

$$\forall t \in \{t^a, 2t^a, \ldots, t^T\}\colon \check{q}^r_t = \frac{1}{|X^r(x)|} \sum_{x' \in X^r(x)} \frac{w_t(s^*(x'))}{t}.$$

Figure 8.3 illustrates the (average) workload ratio computed for the four sets, each of which comprises 36 related instances. The workloads attained by the statically generated schedules for relaxed and tight instances differ significantly. The average workload for tight instances is much higher than the average workload of schedules for relaxed instances. With some exceptions, the schedules for tight instances usually produce without interruption after the initial scheduling phase. One can explain this behavior by requiring a feasible schedule to produce all jobs before exceeding the tight deadline restrictions. For relaxed instances, most schedules initially have an extended idle time period or have a brief high utilization period followed by an idle time phase. While the pattern appears to be similar for all relaxed instances, one can identify high variance in the level of workload utilization.

8.2 Setup of the Computational Experiments for the SFDDHT

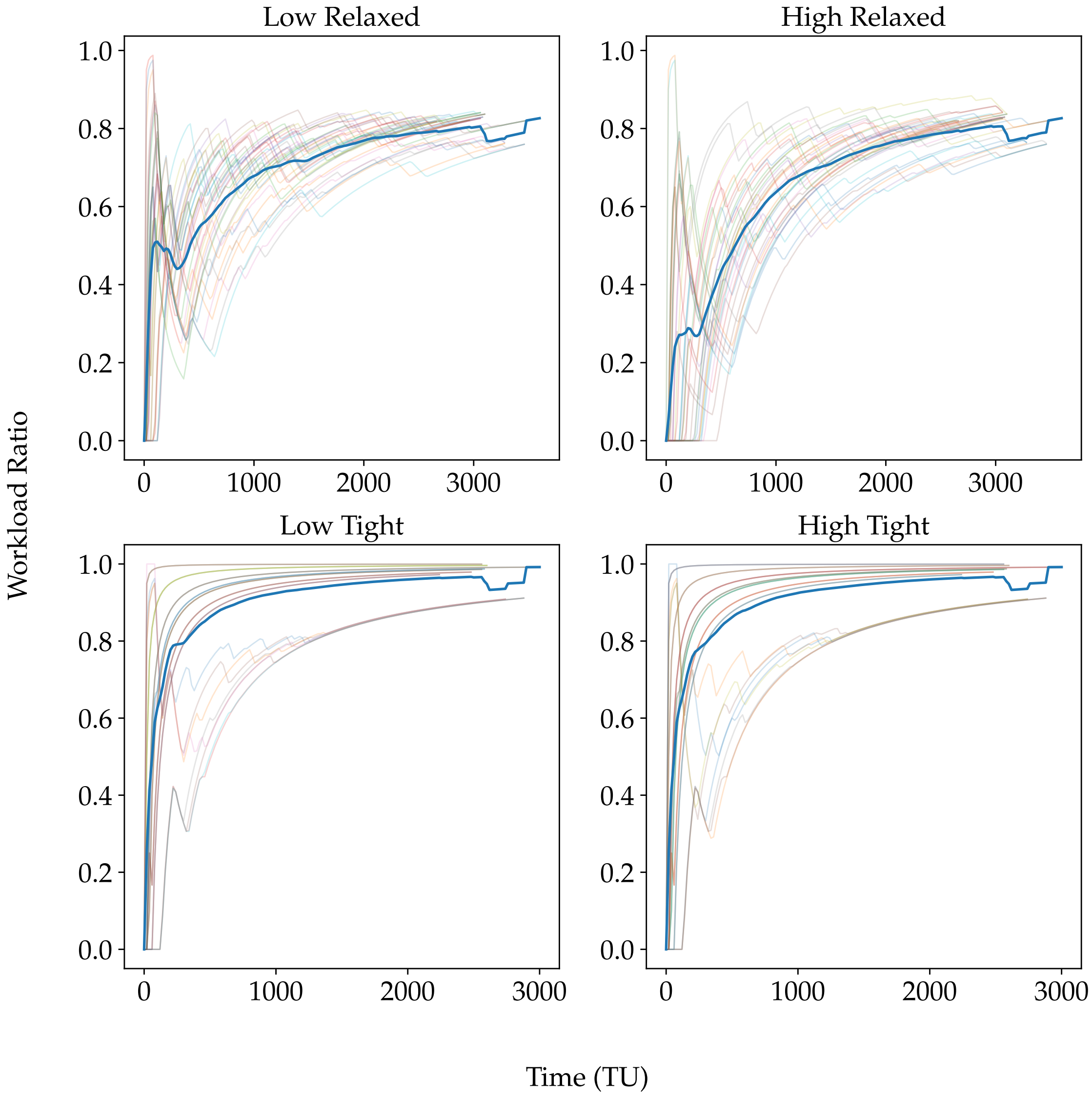
The computational experiments for the static problem D-SFDDHT analyzed the exact optimization approaches SFDDHT-B&B and CPLEX and the heuristic optimization approach SFDDHT-GRASP.

The execution times of the exact approaches were limited to one hour per instance. Upon exceeding this time-limit, the solvers received a termination signal and returned their respective best-found schedules and lower bound values. As the B&B approach requires a large amount of memory to store nodes in the search tree and dominance table, the implementation imposed a limit of, at most, 100 million concurrently stored nodes in the search tree. Note that the program deletes nodes of dominated schedules such that the algorithm is allowed to evaluate more than 100 million nodes in total. The identical limit applies to the dominance table. Upon reaching this limit, the program prohibited the insertion of a new entry until an already inserted entry was deleted to free up memory. This design decision turned out to be inconsequential in the conducted experiments, as the B&B algorithm never reached the dominance table limit. The SFDDHT-B&B implementation was executed with different bound configurations to test the bound effectiveness.

The SFDDHT-GRASP heuristic generated initial upper-bound values for the B&B approach. In the initial experiments, the heuristic executed with an iteration limit of 10,000 iterations for a single instance. The RCL had a capacity limit of 10 entries, independent of

Figure 8.3

Cumulative Workload Ratios of Best-Found, Statically Generated Schedules



Note. The four diagrams display the cumulative workload ratios of the best-found schedules for all 50 job static instances generated by the SFDDHT-B&B or the SFDDHT-GRASP. The time window and transportation cost settings group the plots. The blue lines display the average workload ratio computed from the individual workload ratios for the 36 instances of the respective types. Furthermore, the lighter plots illustrate the individual workload ratios of the generated schedules.

the instance size. In initial tests, increasing the limit or setting the limit as a proportion of the number of entries did not noticeably change the generated solutions' quality. Hence, the reported computational experiments omit the results for limit variations. The SFDDHT-GRASP required no other parameter settings.

The reported times measured the execution times of the approaches. The presented numbers exclude the time spent during the initialization of the approaches (e.g., loading the problem instance, performing memory allocation, and computing preprocessing steps, etc.). Additionally, the B&B timings were measured without the initial bound computation by the SFDDHT-GRASP. Therefore, time measurement for the SFDDHT-B&B started upon inserting the root node into the active list and ended either after detecting that the active list is empty or upon receiving a termination signal because of overstepping the time limit or trying to allocate more than 100 million nodes. The CPLEX solver time measurement started before calling the CPLEX-libraries IloCplex::solve() method and after the program exited this method call. The library parameter IloCplex::TiLim was set to 3,600 seconds. The SFDDHT-GRASP time measured the execution time starting upon calling a solve method that signals each of the GRASP instantiations to construct its first local solution. The end time was taken after stopping the execution of each thread and applying the post-optimization procedure. To simplify the implementation, all solvers were allowed to complete their current computation after receiving a signal to stop reaching a set time limit. Some reported time measurements consequently slightly exceeded the time limit of 3600 seconds for the CPLEX and SFDDHT-B&B solvers. Note that the initial setup for all solvers took less than a second in all of the conducted experiments.

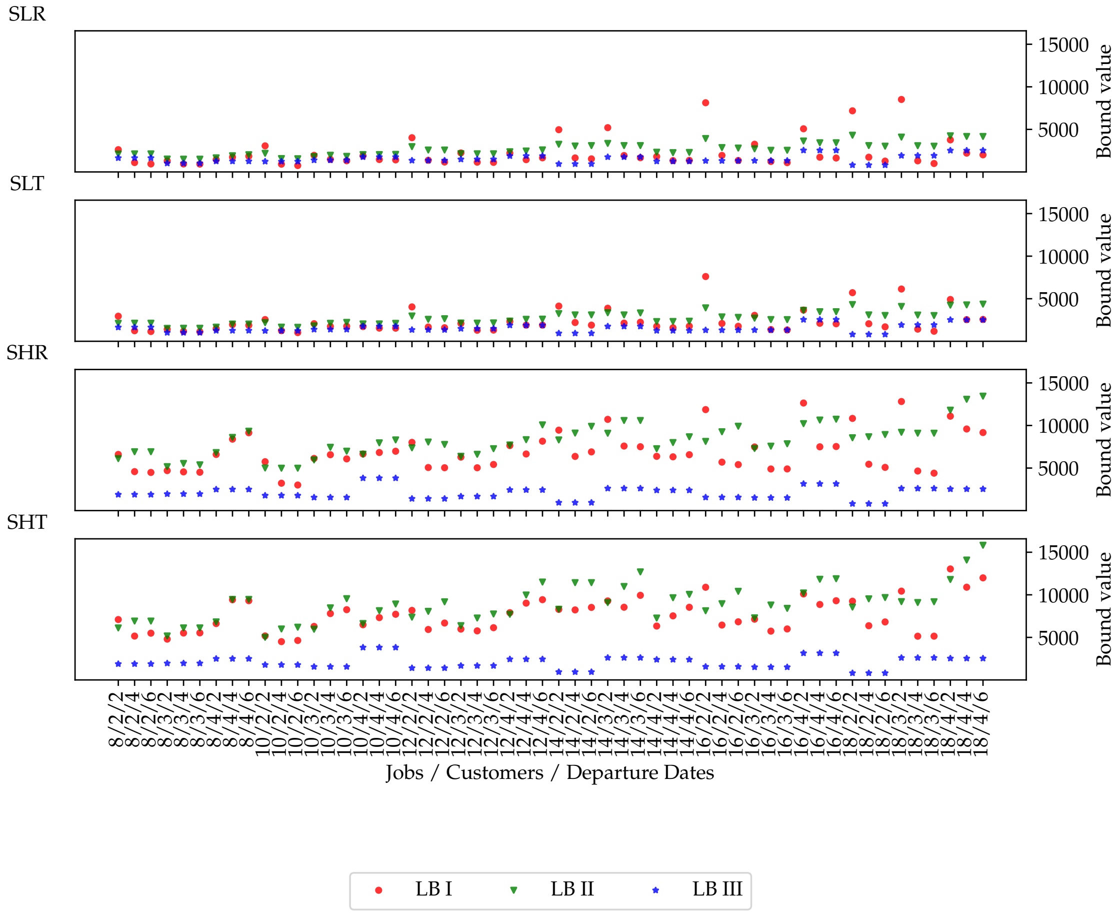
8.3 Comparison of the Proposed Lower Bounds

This section describes the results of the experiment that compares the proposed lower bound procedures LB I, LB II, and LB III in terms of the bound quality and computational complexity. The three procedures were computed for each root problem of the 1,224 generated instances from Section 7.3.1. The experiment results are illustrated in Figure 8.4 for small instances, and in Figure 8.5 for large instances. Both figures separately depict the different instance sets (low or high transportation costs, relaxed or tight time windows). Each point represents the mean bound value computed by one of the three lower bound procedures for a specific instance configuration (each configuration comprises three generated instances). A visual analysis of the illustrations concludes that lower bound LB II outperformed both of the other lower bounds. However, lower bound LB I occasionally returned stronger values than lower bound LB II, and the *third* lower bound returned the weakest bound value in most cases. This general relationship between the *three* bounds is valid over all instance sets. In detail, LB II returned the strongest lower bound in 1,098 out of all 1,224 instances, while in 125 cases, lower bound LB I performed best, and in *one* instance, both lower bounds produced an identical bound value. In terms of solution quality, lower bound LB III was dominated by LB II, as it strictly performed worse. In 1,013 cases, LB III was worse than LB I, while it performed better in the remaining cases.

A summary of the computational complexity for the *three* lower bounds is presented in Tables 8.4 to 8.6. Table 8.4 displays the computational times aggregated by the number of jobs n^N. As expected, the computational time necessary to compute a bound value increased with an increase in the number of jobs for all *three* bounds. The lower bound $LBIII$ computed fastest, followed by lower bound $LBII$. The lower bound LBI required dramatically more computational time than the other *two* lower bounds. This observation is explainable by the cubic computational complexity of the Hungarian method that computes the bound value for LB I. Tables 8.5 and 8.6 show that the lower bound LB I was also highly sensitive to an increase in the number of customers (Table 8.5) and the number of departure dates (Table 8.6). The same conclusion cannot be made for the lower bounds

Figure 8.4

Mean Bound Values of the Proposed Lower Bounds for the Small Instance Sets



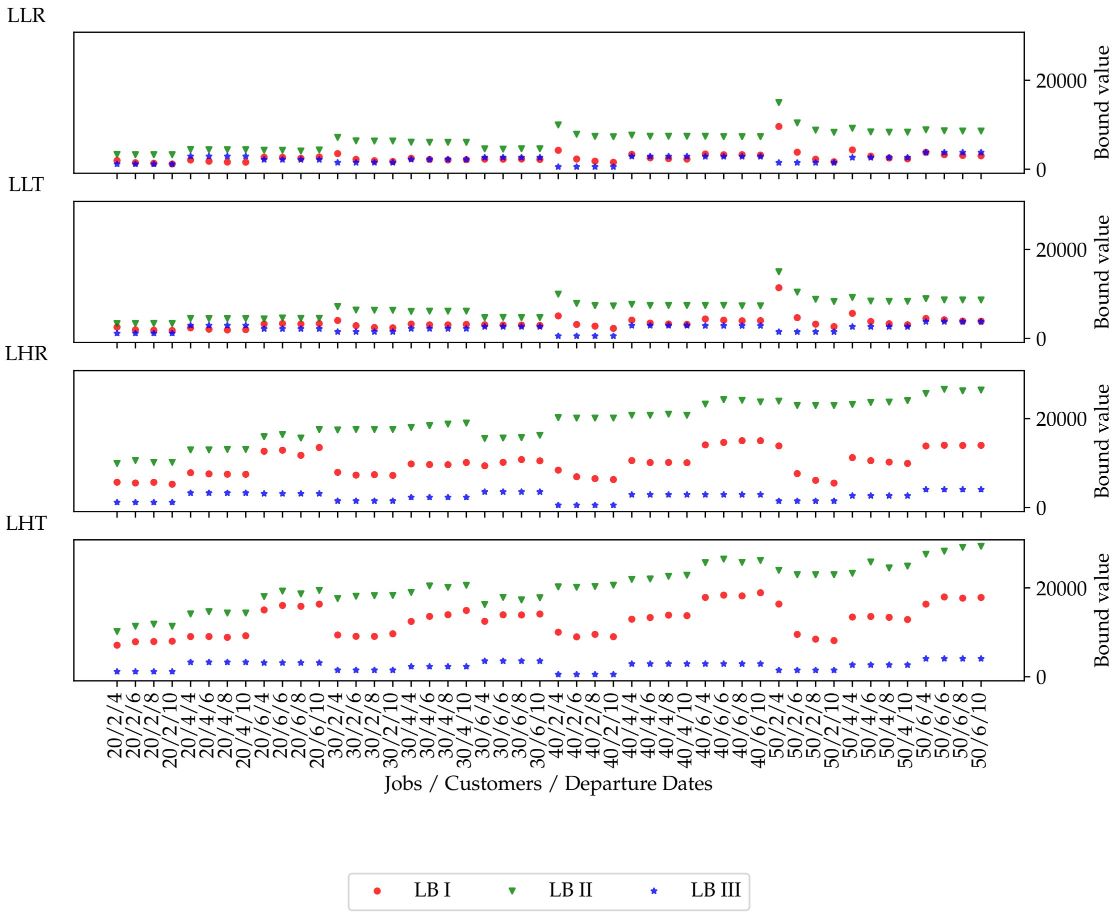
Note. The graphs display the data aggregated by instances with an identical setting for jobs, customers, and departure dates for each small instance set.

LB II and LB III, as we found no correlation between an increase in the two parameters n^G and n^T_g and the observed computational time.

In conclusion, the lower bound LB II appears to be the most promising lower bound for application in the B&B approach due to its strong bounding values with only moderate computational complexity.

Figure 8.5

Mean Bound Values of the Proposed Lower Bounds for the Large Instance Sets



Note. The graphs display the data aggregated by instances with an identical setting for jobs, customers, and departure dates for each small instance set.

Table 8.4

Summary of the Computational Times in Microseconds *for the Three Proposed Bounds Grouped by the Number of Jobs*

	LB I				**LB II**				**LB III**			
(n^N, size)	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**
(8, Small)	12.90	64.49	26.64	10.13	5.27	8.53	6.82	0.70	1.37	2.78	1.81	0.25
(10, Small)	17.05	79.15	33.50	13.41	5.50	9.03	7.41	0.85	1.50	2.69	1.92	0.23
(12, Small)	21.39	94.81	44.25	17.35	5.70	9.76	7.63	0.96	1.62	2.97	1.99	0.25
(14, Small)	28.29	223.35	59.92	33.41	6.05	17.85	8.28	1.44	1.55	6.13	2.12	0.47
(16, Small)	32.50	191.08	73.07	34.28	6.45	10.76	8.66	1.09	1.68	9.30	2.22	0.73
(18, Small)	36.52	304.19	91.88	49.48	6.79	12.03	8.83	1.25	1.85	2.91	2.28	0.25
(20, Large)	69.42	527.82	190.04	104.67	8.92	31.13	11.95	2.03	1.96	3.58	2.61	0.32
(30, Large)	142.23	1510.91	489.46	274.12	9.78	26.13	14.57	2.39	2.41	3.71	2.92	0.28
(40, Large)	289.70	3240.44	1054.46	606.13	12.42	25.69	17.57	2.83	2.92	4.07	3.34	0.25
(50, Large)	501.10	6393.00	1838.44	1187.27	14.70	30.72	21.46	3.83	3.35	4.43	3.78	0.23
Mean	**-**	**-**	**390.17**	**233.03**	**-**	**-**	**11.32**	**1.74**	**-**	**-**	**2.50**	**0.33**

Min = minimal value; Max = maximal value; Mean = mean value; SD = standard deviation

Table 8.5

Summary of the Computational Times in Microseconds *for the Three Proposed Bounds Grouped by the Number of Customers and Instance Size*

	LB I				LB II				LB III			
(n^G, size)	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**
(2, Small)	12.90	141.67	44.40	25.87	5.27	11.34	7.79	1.22	1.37	9.30	1.96	0.57
(3, Small)	17.06	304.19	58.60	42.49	5.50	17.85	7.91	1.44	1.57	6.13	2.08	0.40
(4, Small)	16.64	223.35	61.63	39.47	5.69	10.76	8.11	1.18	1.67	2.99	2.14	0.27
(2, Large)	69.42	2007.58	535.41	424.07	8.92	30.72	17.13	5.68	1.96	4.27	3.01	0.58
(4, Large)	105.53	4117.57	892.42	749.20	9.37	31.13	15.68	3.89	2.21	4.29	3.17	0.47
(6, Large)	89.05	6393.00	1251.48	1258.93	10.12	26.94	16.36	3.70	2.42	4.43	3.30	0.46

Min = minimal value; Max = maximal value; Mean = mean value; SD = standard deviation

Table 8.6

Summary of the Computational Times in Microseconds *for the Three Proposed Bounds Grouped by the Number of Departure Dates and Instance Size*

	LB I				LB II				LB III			
(n^T_g, size)	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**
(2, Small)	12.90	58.45	31.99	11.76	5.27	17.85	6.87	1.00	1.37	6.13	2.09	0.41
(4, Small)	18.82	179.76	57.75	31.66	5.95	10.32	8.01	0.86	1.40	9.30	2.06	0.56
(6, Small)	24.36	304.19	74.88	46.11	6.43	12.03	8.93	1.06	1.37	2.99	2.03	0.29
(4, Large)	69.42	2554.39	518.32	446.39	8.92	26.94	13.53	2.97	1.99	4.43	3.16	0.50
(6, Large)	86.38	4209.65	781.98	734.84	10.62	31.13	15.96	3.83	1.99	4.26	3.18	0.53
(8, Large)	83.48	5212.05	1024.73	974.67	10.81	28.23	17.08	4.27	2.11	4.29	3.17	0.51
(10, Large)	99.71	6393.00	1247.38	1210.00	11.98	30.72	18.98	5.06	1.96	4.27	3.13	0.54

Min = minimal value; Max = maximal value; Mean = mean value; SD = standard deviation

8.4 Comparison of the CPLEX Solver and the SFDDHT-B&B

This section compares the performance of the B&B algorithm and the CPLEX solver, both of which computed solutions with the settings discussed in Section 8.2 for all 1,224 instances. The B&B algorithm was executed *four* times with the following bound configurations: LB I, LB II, LB III, and LB I + LB II. The last configuration executed both lower bounds LB I and LB II for each non-dominated node. For this configuration, a node's lower bound is defined as the largest of both computed bound values. This configuration was included in the experiments, since no clear domination exists between both bounds.

Table 8.7

Results of the CPLEX Solver Applied to all 1,224 *Instances, Compared to the Results of the Branch-and-Bound Approach with Different Bound Configurations*

		CPLEX			B&B (II)		B&B (III)		B&B (I)		B&B (I+II)	
	n^N	**Opt**	**Feas**	**Time**	**Opt**	**Time**	**Opt**	**Time**	**Opt**	**Time**	**Opt**	**Time**
Tight	8	100.00	100.00	0.19	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	10	100.00	100.00	0.80	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	12	100.00	100.00	2.99	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	14	100.00	100.00	13.80	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	16	94.44	100.00	352.04	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	18	94.44	100.00	642.81	100.00	0.00	100.00	0.00	100.00	0.01	100.00	0.01
	20	54.17	100.00	2027.99	100.00	0.01	100.00	0.01	100.00	0.03	100.00	0.03
	30	0.00	31.94	3603.65	100.00	0.17	100.00	0.17	100.00	1.20	100.00	1.28
	40	0.00	0.00	3602.89	100.00	21.05	100.00	21.15	98.61	190.23	98.61	195.22
	50	0.00	0.00	3603.35	75.00	1065.60	75.00	1078.30	55.56	1964.92	55.56	1979.46
	Total (T)	58.33	68.46	-	97.06	-	97.06	-	94.61	-	94.61	-
Relaxed	8	100.00	100.00	1.02	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	10	100.00	100.00	7.85	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
	12	100.00	100.00	167.03	100.00	0.01	100.00	0.01	100.00	0.02	100.00	0.03
	14	74.07	100.00	1363.26	100.00	0.02	100.00	0.02	100.00	0.05	100.00	0.08
	16	11.11	100.00	3368.63	100.00	0.23	100.00	0.14	100.00	0.73	100.00	0.94
	18	0.00	100.00	3605.92	100.00	0.73	100.00	0.49	100.00	2.08	100.00	2.79
	20	0.00	100.00	3604.51	100.00	4.67	100.00	1.93	100.00	44.71	100.00	48.73
	30	0.00	81.94	3603.77	77.78	809.79	86.11	596.42	48.61	2517.15	47.22	2630.99
	40	0.00	4.17	3607.92	6.94	2325.95	13.89	2142.15	0.00	3600.00	0.00	3600.00
	50	0.00	0.00	3605.47	0.00	2741.61	0.00	2597.36	0.00	3600.01	0.00	3600.01
	Total (R)	33.99	74.84	-	74.67	-	76.47	-	70.42	-	70.26	-
	Total	46.16	71.65	-	85.87	-	85.86	-	82.52	-	82.43	-

Opt = percentage of optimal solutions; Feas = percentage of feasible solutions; Time = mean computational time until termination

Table 8.7 presents the results of the experiments. In general, the tight instances were more easily solvable than the relaxed ones for the CPLEX solver and the B&B solver. The different transportation cost settings in the generated instances did not affect the performance of the approaches. Therefore, the table displays the results differentiated by time window tightness, but not transportation costs. The B&B approach solved more than 97% of tight instances with the application of either LB II or LB III. In contrast, the performance of the B&B approach suffered from the high computational complexity of LB

I, as it solved fewer tight instances with configurations LB I and LB I +LB II. Moreover, the aspired performance boost by the combined bound application of the latter configuration did not occur. This configuration's performance was the worst one due to the largest amount of computational time spent in the bounding process out of all B&B configurations.

The B&B approach dominated the CPLEX solver in all respects. All B&B configurations also solved any instance that CPLEX solved in significantly less time, and, beyond that, the B&B approach could solve more instances. This conclusion also holds for the relaxed instances, although both solvers performed worse on relaxed than tight instances. The performance difference stems from the fact that tight instances generally allow fewer variations in the schedule due to each job's limited feasible production interval. Therefore, both of the exact solution approaches needed to evaluate fewer possibilities. Interestingly, the weak lower bound LBIII performed comparably to the strongest bound LBII on tight instances, while providing slightly worse results on relaxed instances. This outcome indicates that the branching scheme and dominance procedures mainly drive the B&B approach's performance.

The B&B solver in its best configuration, with LB II active, could optimally solve all tight instances with up to 40 jobs in seconds, while the CPLEX solver began to struggle at instance sizes with 16 or more jobs. The B&B approach optimally solved all relaxed instances with up to 20 jobs and a significant amount of relaxed instances with up to 30 jobs, while CPLEX barely solved any of the 16 job instances optimally. In total, B&B (CPLEX) solved 594 (357) out of 612 tight instances and 468 (208) relaxed instances. Furthermore, the CPLEX solver failed to produce feasible integer solutions for 193 of the tight instances and 154 of the relaxed instances.

Table 8.8 also highlights the performance difference of the approaches by comparing the executed approaches' lower and upper bound values. The table lists the gaps between the lower/upper bound value and the best-known upper bound value in percentages. The upper bound value originated from the best-produced feasible integer solution from both solvers. Furthermore, the lower bound value of the CPLEX solver was the best-provided objective value with integrality constraints relaxed, while the lower bound value of the B&B solver is the minimal lower bound value selected from all non-dominated leaf nodes of the branching tree at the point of termination (i.e., the branching candidates in the active set). The results stated in Table 8.8 suggest that the lower bounds provided by the CPLEX solver on unsolved instances (in the time limit of *one hour*) are usually low compared to the best-found objective value. As expected, the best results for the different B&B configurations are in line with the already discussed results. The configuration with LB II active was the best-tested one – the achieved lower bound gaps were significantly lower compared to the ones produced by the CPLEX solver. As the B&B approach with LB II active was terminated early due to the imposed node limit of 100 million nodes and never due to the time limit of *one hour*, improved bounding results are likely with increased memory allocation. Please see Appendix A for more gathered statistics regarding the created and dominated nodes of the SFDDHT-B&B.

Overall, the SFDDHT-B&B algorithm proved to be an effective solution method for

Table 8.8

Comparison of Lower and Upper Bound Gaps to the Best-Found Solution for the CPLEX Solver and the B&B Configurations

		CPLEX		B&B (II)		B&B (III)		B&B (I)		B&B (I+II)	
	n^N	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**	**LB**	**UB**
Tight	8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	16	1.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	18	2.24	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	20	10.44	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	30	55.27	4.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	40	68.75	-*	0.00	0.00	0.00	0.00	0.12	0.15	0.15	0.15
	50	76.70	-*	9.06	0.00	12.66	0.00	17.89	1.53	19.09	1.53
Relaxed	8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	14	5.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	16	34.44	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	18	47.65	1.21	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	20	43.83	2.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	30	57.10	11.20	2.12	0.00	6.73	0.02	11.49	0.06	12.57	0.06
	40	65.19	22.97	20.29	0.00	42.00	0.03	47.65	0.11	48.17	0.11
	50	73.86	-*	37.12	0.00	65.26	0.00	64.12	0.00	64.28	0.00

LB = mean gap between lower bound and best upper bound; UB = mean gap between upper bound and best upper bound; * no feasible integer solution

small- to medium-sized instances and is superior to the standard solver approach. To a large degree tight instances in particular were solved quickly by the developed approach.

8.5 Evaluation of the SFDDHT-GRASP

The analysis of the performance of the SFDDHT-GRASP heuristic starts with Table 8.9, which reports the results of the SFDDHT-GRASP configuration (see Section 8.2) that generated the initial upper-bound values for the B&B procedure. The table displays the proportion of instances for which SFDDHT-GRASP computed the best-found solution (best); that is, the B&B procedure did not improve the initial upper bound, either because the heuristic found the optimal solution or because the SFDDHT-B&B terminated forcibly due to its memory limitations without finding a better solution. In the latter case, the quality of the heuristic solution cannot be assessed accurately. Furthermore, the table reports the gap to the best-found solution (gap) and the execution time for computing 10,000 iterations (time).

Table 8.9

Summary of the Initially Generated Solutions for the B&B Approach by the SFDDHT-GRASP within 10,000 Iterations.

		SFDDHT-GRASP					SFDDHT-GRASP		
	n^N	**Best**	**Gap**	**Time**		n^N	**Best**	**Gap**	**Time**
Tight	8	100.00	0.00	0.11	**Relaxed**	8	100.00	0.00	0.19
	10	100.00	0.00	0.17		10	100.00	0.00	0.31
	12	100.00	0.00	0.32		12	100.00	0.00	0.54
	14	100.00	0.00	0.48		14	100.00	0.00	0.78
	16	100.00	0.00	0.76		16	100.00	0.00	1.28
	18	100.00	0.00	1.15		18	98.15	0.02	1.86
	20	95.83	0.02	1.53		20	95.83	0.03	3.44
	30	80.56	0.23	5.55		30	88.89	0.06	11.98
	40	34.72	1.75	13.85		40	91.67	0.11	31.19
	50	44.44	2.49	31.75		50	100.00	0.00	70.93
Total (T)		83.01	0.53	6.46	**Total (R)**		97.06	0.03	14.26

Best = percentage of instances where the SFDDHT-GRASP found the best-known solution; Gap = mean gap to the best-known solution; Time = mean computing time for 10,000 iterations

Overall, the SFDDHT-GRASP generated mostly optimal or close-to-optimal solutions for all problem instances. The solution gaps of suboptimal solutions to the best-found solution were low, with a mean gap of under 1%. The SFDDHT-GRASP computed the best-found solution for over 97% of the relaxed instances and over 83% of the tight instances. As the B&B solver initially started with an upper bound generated by the SFDDHT-GRASP (executed with the stated configuration) in the experiments, the exact procedure certainly profited from the tight upper bounds provided by the heuristic procedure. Bear in mind that the low gaps reported for the large relaxed instances with 30, 40, and 50 jobs likely stem from the SFDDHT-B&B's failure to compute optimal solutions, although this conjecture cannot be proven by the computational results.

Overall, the time spent to compute 10,000 iterations was quite low for small instances but over-proportionally increased on larger instances. This behavior can be explained by the quadratic and cubic complexity of the construction and improvement procedures.

Evidently, lowering the iteration limit on larger instances solves the issue of increased execution times. Alternatively, one could, for example, evaluate only a random sample of construction and improvement moves on larger instances to still construct a large number of solution samples. However, the considered purposes of the heuristic approach did not mandate testing a sampling evaluation scheme.

In addition to this single run on the static instances, experiments were conducted to test the improvement phases' and path-relinking procedures' impacts on the overall solution quality. Different SFDDHT-GRASP configurations were tried with or without the following components: the improvement phase (I), the path-relinking procedure (P), and the post-improvement of a new elite (E). Note that the setting (-/E/-) only applies the improvement procedure to constructed high-quality solutions. Each configuration executed three times on every instance, each time with a different seed for the random number that decides the selection of the next insertion move during the approaches' construction phase.

Table 8.10 reports solution gaps to the previous experiments' best-found solutions and execution times in seconds. As expected, the configuration with all components active (I/E/P) performed best on average (average gap under 1%). The solution quality suffered slightly from deactivating the path relinking procedure. Moreover, both configurations (I/E/P) and (I/E/-) had the highest computational costs. The configuration that only improved the already well-constructed solutions (-/E/-) performed worse, with an average gap of about 5%, but was computationally much less exhausting. The additional activation of the path relinking procedure did not improve the results in this case. In comparison, the sole application of the path-relinking procedure performed even worse. The deactivation of all components did not lead to satisfactory results, with an average gap of over 43%. Overall, the improvement procedure appears to be vital for generating high-quality solutions, while applying the path-relinking procedure is much less critical.

To investigate the different settings' performance under tight time restrictions, the SFDDHT-GRASP additionally executed the six different configurations with a time limit of 6 seconds. This experiment simulated the limited optimization time during the execution of the RTC approach. Table 8.11 reports the results of this experiments, listing the number of executed iterations (iter) performed within 6 seconds, the average gap to the best-known solution (gap) and the average iteration (iter*), and (time*) that found the best solution of the configuration. For example, for tight instances with eight jobs, the configuration (I/E/P) performed 978,164.78 iterations on average and always returned optimal solutions generated after executing on average of 10.43 iterations in under 50 milliseconds. The experiment's overall results are in line with the results of the experiment with the iteration limit active. The intensification procedures turned out to be instrumental for generating high-quality solutions. Although the configuration (-/-/-) executed on average almost thrice (11 times) as many iterations compared to the base configuration (I/E/P) on tight (relaxed) instances, the solution quality was poor in comparison. Especially on small instances, the best-performing solution was generated early during execution; however, this behavior changed for larger instances. The execution of the configuration (I/E/P)

Table 8.10

Results of the SFDDHT-GRASP Solver Applied to all 1,224 Instances Limited to 10,000 Iterations

		I/E/P		I/-/-		-/E/-		-/-/-		-/E/P		-/-/P	
	n^N	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
Tight	8	0.00	0.11	0.00	0.09	2.02	0.06	2.67	0.06	2.81	0.09	2.03	0.09
	10	0.00	0.16	0.00	0.13	3.14	0.08	4.08	0.08	3.65	0.11	4.15	0.11
	12	0.00	0.29	0.07	0.27	0.26	0.13	2.05	0.13	0.23	0.18	0.91	0.18
	14	0.01	0.43	0.00	0.40	0.39	0.20	3.43	0.20	0.87	0.25	1.21	0.25
	16	0.02	0.70	0.01	0.65	0.42	0.29	7.31	0.28	0.80	0.36	2.54	0.36
	18	0.02	1.08	0.10	1.02	1.18	0.47	9.74	0.46	1.69	0.55	3.25	0.56
	20	0.09	1.31	0.11	1.20	2.40	0.62	16.45	0.61	2.21	0.78	3.52	0.79
	30	0.77	4.68	1.65	4.48	8.20	2.31	40.80	2.30	7.95	2.80	11.46	2.83
	40	2.56	13.26	4.01	12.43	13.16	6.12	57.17	6.08	14.34	7.44	19.88	7.54
	50	4.52	25.63	5.80	24.73	22.69	11.10	77.83	11.04	23.86	13.20	32.71	13.45
Total (T)		0.94	5.52	1.38	5.27	6.12	2.48	25.20	2.46	6.58	2.99	9.19	3.03
Relaxed	8	0.04	0.18	0.10	0.14	0.93	0.05	3.88	0.05	0.67	0.12	1.52	0.12
	10	0.04	0.31	0.02	0.23	0.39	0.06	5.74	0.06	0.57	0.17	1.06	0.17
	12	0.01	0.55	0.00	0.45	0.68	0.08	16.16	0.08	1.08	0.25	3.02	0.25
	14	0.00	0.82	0.01	0.68	1.40	0.08	21.23	0.08	2.18	0.33	5.61	0.32
	16	0.07	1.41	0.07	1.19	2.15	0.10	31.89	0.09	2.78	0.50	7.88	0.48
	18	0.00	2.05	0.13	1.75	3.50	0.12	36.98	0.12	3.57	0.63	10.72	0.63
	20	0.05	3.98	0.11	3.21	5.07	0.17	54.97	0.16	3.81	1.54	6.93	1.41
	30	0.23	15.84	0.56	13.38	7.11	0.46	103.57	0.44	5.56	5.44	12.89	4.81
	40	0.68	42.71	1.25	37.27	8.87	0.98	132.51	0.93	8.50	12.72	21.33	11.30
	50	0.65	98.50	1.29	88.96	7.94	1.59	153.87	1.44	7.00	25.81	22.52	21.67
Total (R)		0.20	19.41	0.41	17.20	4.21	0.42	62.57	0.39	3.88	5.53	10.12	4.78
Total		0.57	12.46	0.89	11.24	5.16	1.45	43.88	1.42	5.23	4.26	9.65	3.90

Gap = mean gap to the best-found solution; Time = mean computational time

Note. Each configuration was executed *three* times with different seeds for the utilized random number generator.

attained better results with the less restrictive iteration limit for large instances, compared to the 6-seconds time limit.

Table 8.11

Results of the SFDDHT-GRASP Solver Applied to all 1,224 *Instances Limited to 6 seconds*

		I/E/P				I/-/-				-/-/-			
	n^N	**Iter**	**Gap**	**Iter***	**Time***	**Iter**	**Gap**	**Iter***	**Time***	**Iter**	**Gap**	**Iter***	**Time***
Tight	8	978164.78	0.00	10.43	0.00	1402473.98	0.00	11.67	0.00	2879823.28	1.28	648.56	0.00
	10	610596.98	0.00	17.67	0.00	738190.96	0.00	17.07	0.00	1827564.38	3.54	6437.47	0.07
	12	258722.69	0.00	17.72	0.00	292982.72	0.00	30.94	0.00	884151.96	1.03	35569.90	0.44
	14	165890.30	0.00	33.59	0.00	179848.80	0.00	48.26	0.00	436816.87	1.31	70597.80	1.13
	16	103480.52	0.00	40.19	0.00	109408.39	0.00	59.00	0.01	287664.70	3.14	112410.06	2.45
	18	63303.02	0.04	125.37	0.01	66044.15	0.12	257.31	0.03	156087.37	6.07	73903.94	2.93
	20	82477.46	0.07	935.11	0.14	89521.24	0.08	1707.94	0.19	163950.94	10.92	91153.38	3.16
	30	18373.04	0.32	5435.29	1.36	19670.86	1.23	7211.71	2.02	41795.71	37.20	21675.03	3.12
	40	6516.06	2.92	3091.83	2.82	6905.65	4.56	3552.22	2.67	12753.33	57.54	6465.99	2.96
	50	3804.81	5.34	1643.67	2.96	3918.24	6.61	1746.60	2.76	6956.54	79.90	3444.06	2.98
Total (T)		205647.15	1.02	1328.83	0.86	260203.26	1.48	1710.20	0.90	595136.84	23.41	40828.68	2.07
Relaxed	8	507058.78	0.12	16.57	0.00	705191.96	0.00	22.30	0.00	3847197.21	1.76	148992.51	0.32
	10	254186.63	0.00	30.72	0.00	329926.69	0.04	87.89	0.00	2813248.61	0.78	418888.70	1.09
	12	133793.59	0.00	60.06	0.00	156199.94	0.00	76.96	0.00	1739453.09	4.98	720396.61	2.50
	14	86602.56	0.00	73.78	0.01	98949.94	0.00	79.77	0.01	1508360.52	8.66	754351.50	2.92
	16	50700.76	0.03	601.20	0.08	56775.91	0.02	616.06	0.08	1206062.24	15.65	546402.98	2.78
	18	32655.13	0.00	775.37	0.16	36737.46	0.00	2033.11	0.34	862302.50	21.11	409132.83	2.81
	20	17760.83	0.05	1269.47	0.61	21825.32	0.12	1974.07	0.70	540941.19	37.95	260067.99	2.94
	30	4216.43	0.46	1241.89	1.88	4976.93	0.89	2082.40	2.58	183042.38	83.45	80801.06	2.81
	40	1562.31	1.69	824.24	3.12	1757.50	2.53	936.88	3.25	89837.75	120.79	42409.39	2.81
	50	647.62	2.33	391.10	3.52	699.42	3.30	350.28	2.93	56777.75	140.15	29755.74	3.11
Total (R)		96815.92	0.55	575.88	1.10	125584.18	0.81	887.28	1.15	1154785.01	49.73	313404.89	2.47
Total		151231.54	0.78	952.36	0.98	192893.72	1.14	1298.74	1.02	874960.92	36.57	177116.78	2.27

		-/E/P				-/E/-				-/-/P			
	n^N	**Iter**	**Gap**	**Iter***	**Time***	**Iter**	**Gap**	**Iter***	**Time***	**Iter**	**Gap**	**Iter***	**Time***
Tight	8	1362521.00	1.88	255.66	0.00	2844963.59	1.97	112.06	0.00	1356443.17	1.21	468.45	0.00
	10	1020279.98	2.89	259.30	0.00	1843392.89	2.94	210.15	0.00	989079.79	1.68	900.47	0.01
	12	494010.09	0.07	1981.98	0.04	862510.13	0.03	673.09	0.01	498079.02	0.42	13850.43	0.21
	14	306262.94	0.09	6914.89	0.15	435391.98	0.00	1368.94	0.02	307611.30	0.58	25093.02	0.54
	16	214035.25	0.10	3120.51	0.11	286068.70	0.05	4742.35	0.17	212612.58	0.85	49599.57	1.38
	18	128961.39	0.39	5800.69	0.29	155301.33	0.98	7071.43	0.34	129842.19	1.17	52051.08	2.28
	20	129144.96	1.05	19073.38	0.76	162724.93	1.76	14783.97	0.55	130001.92	2.10	34799.65	2.11
	30	30037.39	5.24	6540.12	1.28	41512.49	6.07	4683.79	0.84	29765.11	8.69	14236.18	3.06
	40	10602.85	11.09	2610.19	1.53	12706.46	11.47	2158.69	1.01	10536.21	19.32	6460.39	3.52
	50	5957.64	23.72	2144.44	2.08	6923.26	23.74	2062.38	1.76	5927.51	34.20	3990.38	3.95
Total (T)		328936.06	5.35	5212.45	0.72	593481.01	5.59	4037.92	0.54	327097.79	8.16	19420.58	1.88
Relaxed	8	988192.48	0.66	607.87	0.01	3721896.44	0.22	37618.59	0.08	986683.40	0.43	6197.83	0.07
	10	671465.87	0.06	6878.50	0.13	2713356.30	0.09	1090.24	0.00	667308.54	0.14	55883.48	0.58
	12	527428.33	0.22	15859.56	0.28	1693399.83	0.43	51651.76	0.26	525159.28	0.89	163532.20	1.13
	14	416167.30	0.71	26056.04	0.51	1481394.57	1.14	52331.76	0.26	416766.35	2.64	181837.13	1.87
	16	322415.00	0.98	28907.26	0.64	1186022.98	1.82	46560.78	0.27	315413.17	4.51	115470.87	1.91
	18	263503.43	2.57	13740.69	0.57	841595.72	3.07	53451.37	0.44	250224.89	4.89	114426.48	2.75
	20	50214.18	2.82	10311.26	1.31	534654.17	5.21	11492.83	0.10	52670.69	4.72	25739.58	2.73
	30	14207.60	5.08	3069.99	1.53	182902.65	6.81	393.49	0.03	14812.17	11.59	8616.64	3.58
	40	6459.29	8.13	816.89	0.85	88702.93	9.06	413.07	0.08	7137.72	21.80	4727.81	4.21
	50	3093.99	7.24	459.81	0.75	54516.93	7.76	469.14	0.18	3635.33	28.42	2781.83	4.57
Total (R)		290100.51	3.20	9846.51	0.71	1128120.71	3.99	22917.28	0.16	287024.13	9.03	61251.94	2.51
Total		309518.29	4.28	7529.48	0.72	860800.86	4.79	13477.60	0.35	307060.96	8.59	40336.26	2.19

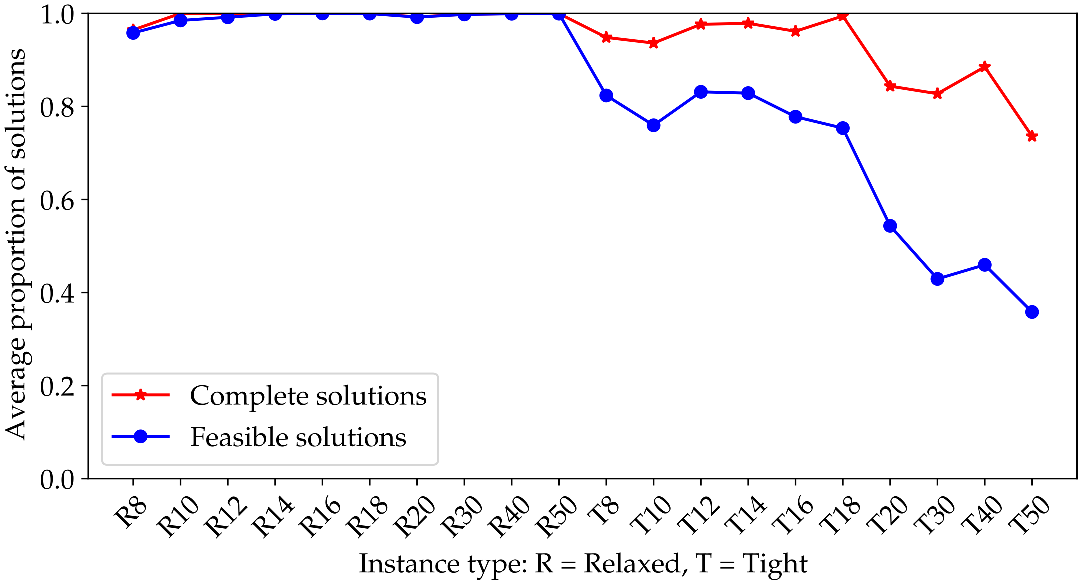
Iter = mean number of iterations executed within 6 seconds; Gap = mean gap to the best-found solution; Iter* = mean iteration number in which the best solution was found; Time* = mean computational time after which the best solution was found

Note. Each configuration was executed *three* times with different seeds for the utilized random number generator.

An interesting aspect worth mentioning is the important role of the repair procedure, especially for tight instances generally. To demonstrate this importance, the implemented program tracked the number of constructed complete solutions and the number of feasible solutions generated, with or without repair, during execution of the SFDDHT-GRASP. Figures 8.6 to 8.8 illustrate the results of tracking these numbers for the (-/-/-) configuration as it performed only the construction and repair steps, thus computing the largest number of iterations in the specified time limit of 6 seconds. Figure 8.6 displays the proportion of complete solutions and feasible solutions for the processed instances broken down by time window tightness and number of jobs. For relaxed instances, the completion success rate was approximately 99.58% on average, whereas this value decreased to 89.80% on tight instances. The completion success for large (20 jobs or more) and tight instances was only 82.30%, and the rate of produced feasible solutions (with and without repair) followed the same pattern. While in 99.30% of the iterations, a feasible solution could be constructed for relaxed instances, the proportion dropped to 62.98% for tight instances.

Figure 8.6

Proportion of Incomplete Solutions Generated in 6 Seconds by the SFDDHT-GRASP with Setting (-/-/-)

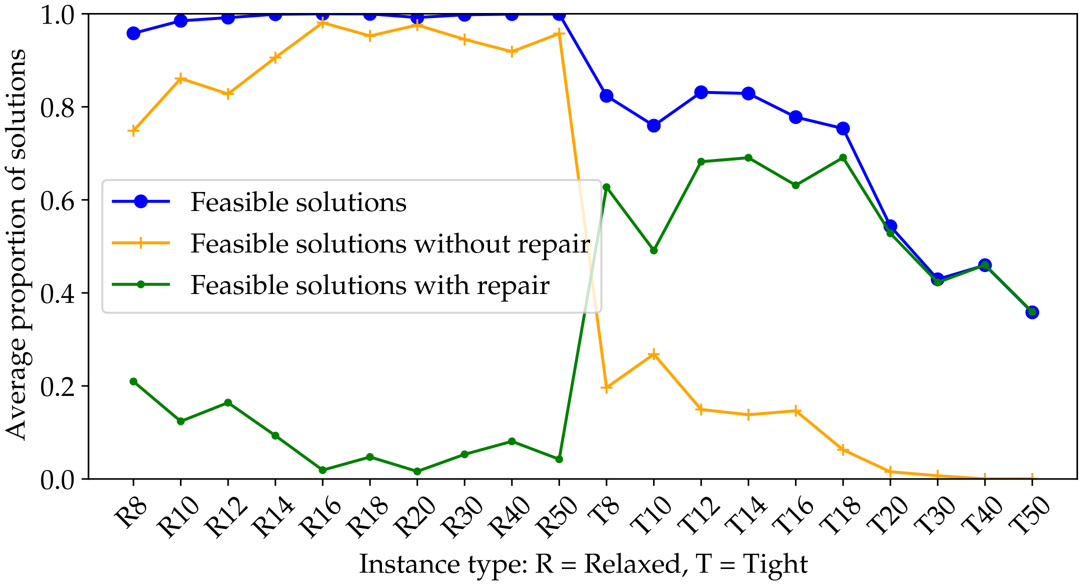


Note. Incomplete solutions are discarded, and no repair attempt is made during application of the SFDDHT-GRASP. The labels Rx/Tx indicate the time-window tightness and the number of jobs of the instances.

Figure 8.7 displays the feasibility aspect in more detail. The figure additionally shows the proportion of feasible solutions with and without repair. About 91.36% of iterations produced a complete and feasible solution for relaxed instances without requiring the repair procedure. In addition, roughly 8% of initially infeasible solutions could be repaired. The results for the tight instance set highlights the importance of the repair phase. Only 8.65% initial construction attempts were successful, whereas in 54.36% of the iterations, the repair phase produced a feasible solution. Furthermore, less than 1% of the iterations for tight instances with 30 or more jobs resulted in a feasible solution.

Figure 8.7

Proportion of Feasible Solutions Constructed in 6 Seconds by the SFDDHT-GRASP with Setting (-/-/-)



Note. The labels Rx/Tx indicate the time-window tightness and the number of jobs of the instances.

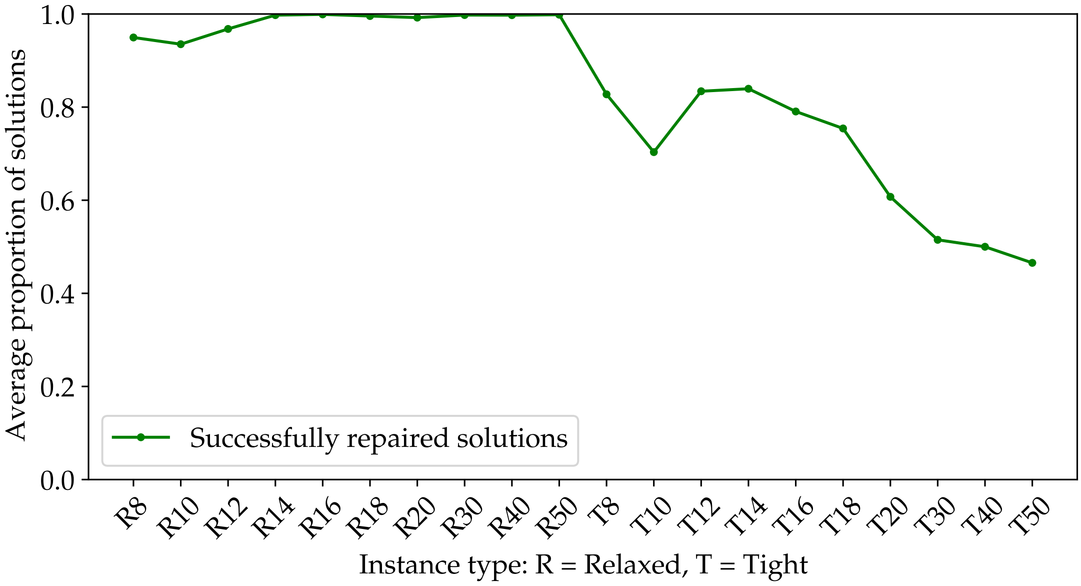
The repair phase almost always successfully repaired relaxed instances, as depicted in Figure 8.8. The figure illustrates the proportion of successfully repaired solutions. The success rate was at 98.76% for relaxed instances, and it decreases for tight instances to approximately 66.16%.

Overall, the repair procedure was highly successful in producing feasible solutions for the considered instance types. For larger instances, it might be advisable to implement techniques to increase the likelihood of producing complete solutions. For example, a backtracking procedure could rewind a number of executed solutions during the construction phase when encountering the situation where no extension of a partial schedule exists. However, evaluating the necessity and possible solutions of additional components for the SFDDHT-GRASP is out of the scope of this work.

In summary, the SFDDHT-GRASP results are promising, as the heuristic generates high-quality solutions with low gaps. In particular, the heuristic seems to be a fitting tool to compute schedules for relaxed instances that provide less opportunity for restricting the subproblem space during application of the SFDDHT-B&B algorithm. Therefore, it is suitable as a stand-alone heuristic for the SFDDHT. In addition, the computational time necessary to attain these results turned out to be very low. This property enables the SFDDHT-GRASP application to be an optimization procedure for the RTC approach. Since the SFDDHT-GRASP evaluates many solutions whose generation does not depend on prior computations, the approach is highly scalable, and the anticipation horizon length for the RTC approach can consequently be adjusted as required. Even with very low

Figure 8.8

Proportion of Successful Repair Attempts in 6 Seconds by the SFDDHT-GRASP with Setting (-/-/-)



Note. The labels Rx/Tx indicate the time-window tightness and the number of jobs of the instances.

optimization time, useful results can be expected.

8.6 Setup of the Computational Study for the D-SFDDHT

The computational experiments for the dynamic problem D-SFDDHT tested the implementation of the RTC approach executed with different configurations on the dynamic instances generated from the static instance sets. The implementation of the RTC approach allows for the configuration of the following parameters:

Either one of the workload-balancing settings – WBOT, WBPT, WBOI, or WBPI– is active, or the RTC enforces no workload balancing. The abbreviation WBN indicates the latter option.

Additionally, the implementation allows for a free configuration of the anticipation horizon length, the initial plan construction duration, and the translation of the instance parameters' TU values into milliseconds. The experiments comprise *three* different configurations for these timings:

1. *Short*: In this setting, a time unit equates to 100 milliseconds. Therefore, the average job processing duration is approximately five seconds (50 TUs) long. The anticipation horizon length of t^a is 2 seconds, and the initial plan solve time is limited to 5 seconds.

2. *Medium*: This setting triples the parameter values of the short setting: A TU is 300 milliseconds long, while the anticipation horizon length is 6 seconds, and the initial plan solve time is limited to 15 seconds. Note that the medium setting is considered

to be the base case for the experiments, which is why the vast number of results presented in the experimental study use this setting.

3. *Long*: In this configuration, the timings are five times as large as in the short configuration. Hence, the values are 500 milliseconds for each TU, 10 seconds for each anticipation horizon, and 25 seconds for the initial solve time.

Furthermore, the optimization procedure's deactivation during the execution of the RTC approach was tested. In addition to the base configuration that optimizes theoretical plans and uses the optimizer to integrate jobs as well as the initial least-cost insertion, the following two additional settings were used:

- *Optimization and Insertion*: In this setting, only the least-cost insertion integrates new order requests. If the insertion procedure does not successfully generate a feasible plan that integrates the new request(s), then the RTC immediately rejects the order request(s). The optimizer reschedules during each anticipation horizon.
- *Insertion*: In this setting, the RTC does not produce optimized schedules after the initial plan generation. New order requests are integrated only by least-cost insertion. As in the previous setting, rejection of new requests occurs when the insertion fails for a new request.

The RTC approach executed with all combinations of the five workload configurations and the three time-setting configurations on all 144 dynamic instances with 50% dynamic orders. The number of tested configurations for the experiments concerning the 76% instances and the deactivation of the optimization components was limited. Therefore, throughout Sections 8.7 to 8.7.4, the results pertain to the instances with 50% of dynamic orders solved by the RTC approach with a medium time setting, optimization activated, and both integration strategies. Section 8.8 presents the results for different time settings, while Section 8.9 varies the number of dynamic order arrivals, and Section 8.10 compares the different optimization and integration settings.

To assess the quality of the generated dynamic schedules, these schedules are compared to statically generated schedules by the SFDDHT-B&B or the SFDDHT-GRASP. As explained in Section 8.1.2, each dynamic instance was generated with the information of a static instance and the best-performing schedule. The best-performing schedule for the static instance was thus taken as a benchmark for the dynamic scheduling cost. Note that since the best performing schedule is not necessarily optimal, it is possible – but unlikely – that a dynamically produced schedule performs better. For the following experiments, the static results are reported alongside the dynamic results. Furthermore, the following sections report the total cost gap for the dynamic schedules compared to the static schedules. For a dynamic schedule $s(\check{x})$ for instance $\check{x}$ and the corresponding best-found static schedule $s^*(x)$ with static instance x, the objective gap is computed as

$$z^{gap}(s^*(x), s(\check{x})) = \frac{z(s(\check{x})) - z(s^*(x))}{z(s^*(x))} \cdot 100. \tag{8.1}$$

Allowing order rejections in the dynamic case resulted in occasional negative gaps because the overall scheduling costs significantly decrease for schedules without the rejected orders.

8.7 Comparison of the Real-Time Control Workload-Balancing Methods

This section presents the results of the RTC approach executed on the 144 50% instances with medium time settings and with optimization as well as both integration strategies active. The RTC approach was executed *five* times on each instance with each of the *five* different workload-balancing configurations: WBN, WBOT, WBPT, WBOI, and WBPI. The presented analysis of the experimental results offers insights into the overall performance of the RTC approach. Section 8.7.1 presents an initial analysis of the results of the generated schedules; it primarily focuses on the dynamic schedule costs and the order acceptance rates of the configurations. As the interpretation of the results proved to be difficult with the unprocessed information alone, a number of additional, different evaluation approaches were tested to investigate various aspects of the solutions and compare the configurations in different ways. Section 8.7.2 describes the performance indicators and their interpretation for the RTC approach that exclude the cost contribution of orders. Section 8.7.3 evaluates the dynamic schedules by building a new static instance for each dynamically produced schedule that only comprises the accepted orders. Afterward, an optimized static schedule (with full information reveal) is compared to the dynamically generated schedule. Lastly, Section 8.7.4 discusses the attained results in light of flexible and inflexible environments that value the rejection of orders differently.

8.7.1 Initial Analysis of the Results

In this subsection, the dynamically produced final schedules of the five different RTC settings are compared with the best-known static schedules generated by the SFDDHT-B&B or the SFDDHT-GRASP. Table 8.12 summarizes the results of the first experiment. The first data column (Static) of the table presents the best-found schedules for the relevant static instances, while the second to sixth data columns display the values attained by the five RTC configurations. The data is aggregated by the different transportation cost settings (low or high) and the time window tightness settings (relaxed or tight). Each cell displays five values with the following interpretation: The first value is the mean objective value for the grouped instances; the second value is the mean objective value gap computed by Equation (8.1); the third and fourth values are the mean holding and transportation costs; and the last value is the mean number of rejected orders during the execution of the RTC approach.

The discussion starts with the results regarding the acceptance or rejection of dynamic orders. The attained results of configuration WBN highlight the need for the proposed workload-balancing techniques because executing the RTC without balancing resulted in rejecting, on average, 5.64/4.89 of the 25 dynamic orders, which is about 20% of all

Table 8.12

Comparison of Different Workload-Balancing Methods Applied to all 144 Dynamic Instances with 50% Dynamic Arrivals and Medium Time Settings

Instances		**Static**	**RTC configuration**				
			WBN	WBOT	WBPT	WBOI	WBPI
High	Relaxed	38077.36*	38720.92	53041.61	54622.58	43788.25	46724.33
		-	1.89	38.93	41.95	15.31	22.77
		19966.94	21068.44	33430.00	34602.14	24675.22	27948.69
		18110.42	17652.47	19611.61	20020.44	19113.03	18775.64
		-	0.94	0.00	0.00	0.03	0.00
High	Tight	51090.36	35682.67	57308.75	56897.31	53722.33	53642.50
		-	**-28.46**	11.86	11.24	5.34	5.03
		27256.78	15032.08	33590.58	32560.31	30351.75	29912.56
		23833.58	20650.58	23718.17	24337.00	23370.58	23729.94
		-	**5.64**	0.78	0.89	1.14	1.31
Low	Relaxed	21569.56	21929.25	34757.97	36238.17	26842.47	30525.97
		-	3.31	66.64	75.32	29.02	46.03
		16323.42	16942.58	29450.31	31030.22	21497.50	25404.83
		5246.14	4986.67	5307.67	5207.94	5344.97	5121.14
		-	0.83	0.00	0.00	0.06	0.00
Low	Tight	30350.36	17525.33	36234.25	35587.69	32781.89	32807.00
		-	**-39.35**	21.50	19.39	10.26	11.49
		24508.81	12143.86	30478.06	29844.94	27121.86	27183.28
		5841.56	5381.47	5756.19	5742.75	5660.03	5623.72
		-	**4.89**	0.69	0.89	1.06	1.17

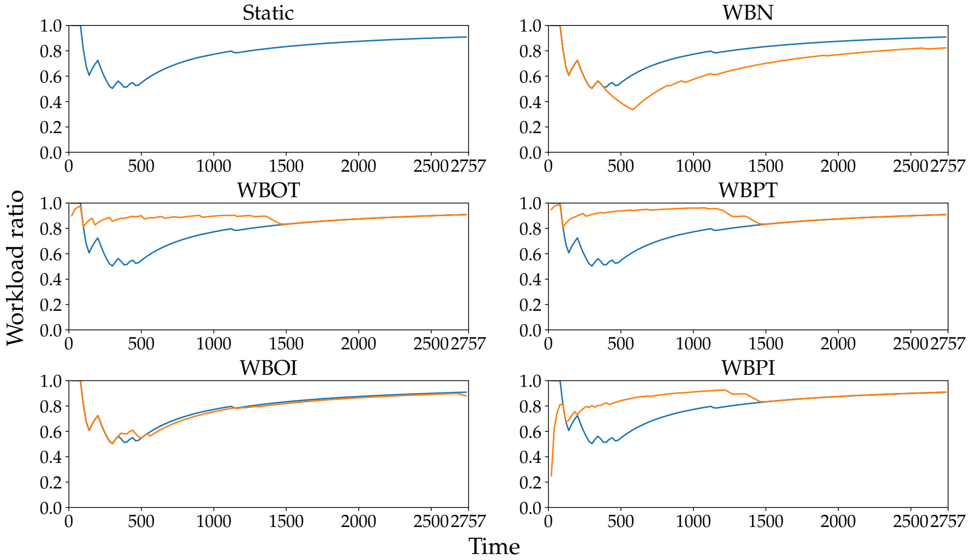
* Total cost, objective gap to best-found static objective, holding cost, transportation cost, and rejected jobs

requests. In comparison, the balancing methods introduced in Section 7.5 significantly reduce the number of rejected orders. In particular, on each of the 72 relaxed instances, the configurations WBOT, WBPT, and WBPI were able to integrate all dynamic orders. The issue of rejecting orders is much more prevalent for tight instances compared to relaxed instances. This discrepancy occurs because each job's feasible production interval is, on average, wider in relaxed instances than in tight instances. Therefore, balancing the workload is more critical in tight environments compared to relaxed environments. In terms of minimizing the number of rejected jobs, the total omniscient and total predictive workload methods perform slightly better than the interval omniscient and interval predictive workload methods. Further investigating this result led us to conclude that the total workload methods often overestimate the required workload in the beginning compared to the best-found solution(s).

Figure 8.9 illustrates this observation for a selected instance. The figure displays the workload ratios of the RTC schedule with different workload-balancing configurations active and compares these ratios to the best-known solution's workload ratios. The predicted total workload ratio followed by the RTC approach for WBOT and WBPT configurations was significantly larger than the actual workload ratios at the beginning of the static schedule. The WBN configuration underutilized the machine compared to the static schedule. As the plotted lines do not converge at the end of the schedule for this configuration, the

Figure 8.9

Comparison of the Workload Ratios of the Different Balancing Configurations for a Selected Instance with Tight Time Windows



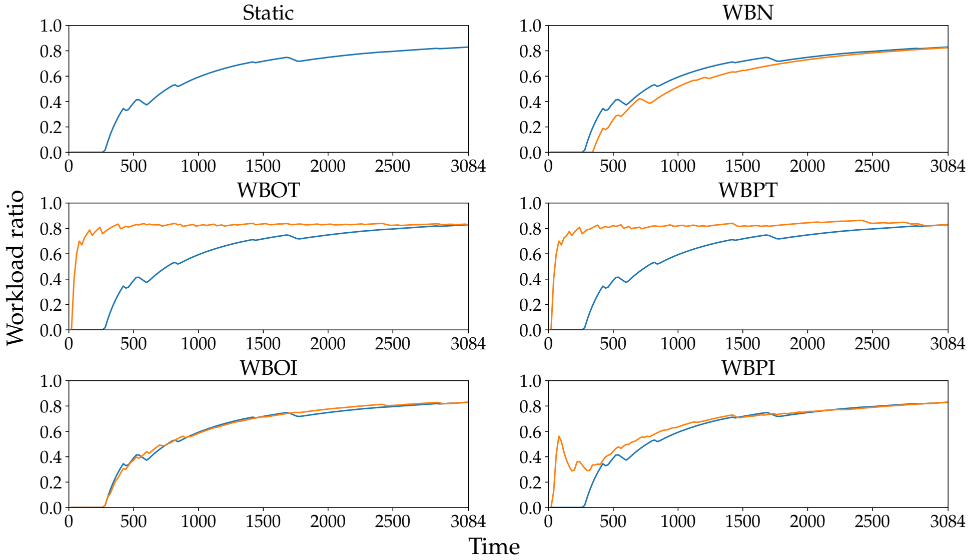
dynamic schedule's total workload was lower than in the static schedule. Therefore, the RTC approach could not accept all jobs during its execution. The interval approach WBOI followed the static machine utilization very closely; the small drop-off at the end of the line occurred due to the production's earlier completion. Furthermore, the illustration for the WBPI configuration indicates that the used definition of related instances might be too simple for this instance to avoid mixing different workload patterns. Figure 8.10 illustrates the workload ratios for a characteristic instance with relaxed time windows. The behavior of the different configurations executed on relaxed instances is identical to the behavior on tight instances. However, a much more pronounced over-utilization of machine time can be observed for the total workload-balancing configurations.

In general, the overestimation at the beginning forces early production of jobs during the execution of the RTC approach, which is more costly on the one hand but benefits the integration of future requests on the other, as more capacity is available later on. The number of rejected jobs does not drastically differ between the balancing approaches. However, there is a slight bias in favor of both total workload methods. It appears that the choices made to reduce the penalized machine under-utilization do not necessarily align with the scheduling decisions of the static schedule, and more rejections therefore occur compared to the total workload approaches.

Consider the displayed cost values of Table 8.12. The interpretation of the objective values and objective gaps is not straightforward. The option of the RTC approach to reject jobs whenever it does not establish a feasible job integration, positively influences the objective value, as it lowers the scheduling costs. A job rejection lowers the final schedule's objective value in two ways: One, each rejected job itself causes no holding and

Figure 8.10

Comparison of the Workload Ratios of the Different Balancing Configurations for a Selected Instance with Relaxed Time Windows



transportation cost in the final schedule, and two, rejection enables the dynamic solver to schedule other jobs more efficiently, as the machine does not need to allocate time to produce the rejected job.

For this reason, the table highlights the results of the WBN configuration executed on tight instances (bold numbers). On average, this configuration's objective value was almost 35% lower than the best-found objective values for tight instances. The total holding costs almost halved for tight instances with many rejected orders. Nevertheless, the results seem to be very promising on relaxed instances. In this setting, order integration was much simpler, even without any workload-balancing mechanism active. The displayed average objective gaps were 1.89% on the relaxed instances with high transportation costs and 3.31% with low transportation costs. Again, it must be noted that these gap values were not measured in an entirely fair manner, as 0.94 and 0.83 jobs were rejected on average.

The balancing approaches' performance on relaxed instances differs significantly compared to their performance on tight instances. First, the RTC approach generally solved tight instances with vastly lower objective gaps than the relaxed instances. Simultaneously, the number of rejected jobs was slightly higher on average. Second, the interval workload-balancing methods produced schedules with lower gaps but more rejections than the total workload configurations. Third, the predictive interval balancing method WBPI attained an average gap of 5.03% on tight instances with high transportation costs and an average gap of 11.49%. Note that this approach performed comparably to the WBOI approach. Taking these results into account, it appears that balancing methods are instrumental in tight instances while attaining relatively low gaps to statically generated schedules.

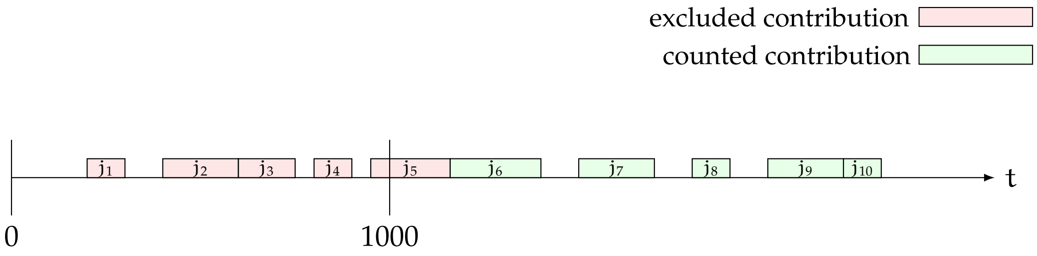
As already discussed, the option to reject jobs distorts the interpretation of the attained objective values. The drastic positive effect on the objective function of rejecting even a few orders mandated measuring the performance of the RTC approach differently. Section 8.7.2 thus proposes performance indicators that aim to interpret the solution quality of the RTC configurations with a different perspective than the one portrayed in Table 8.12.

8.7.2 Analysis with Different Performance Indicators

This section presents *three* performance indicators that evaluate the scheduling costs in different ways. The *first indicator* measures cost values for all solutions – static and dynamic – from a specified time onward and ignores earlier cost contributions. That is, the measurement excludes early scheduling costs from the total cost computation for comparison. This analysis is motivated by the fact that the initial scheduling phase is not necessarily representative in the dynamic context. The expectation is that the total workload predictions more accurately reflect the workload in the planning horizon's main part than at the start. The first indicator with a warm-up phase measures cost values for all solutions – static and dynamic – beginning at time $t = 1{,}000$. The specific threshold value was chosen from visually analyzing the workloads of the dynamic schedules that converge around this time with the static schedule workloads. The separation of jobs impacts the contribution of holding costs and transportation costs. Specifically, in a schedule $s(x)$ for instance x, holding costs for jobs $j \in N$ with start time $S_j < 1{,}000$ are excluded, and delivery assignments of these jobs are not counted. Hence, in a schedule where only jobs that start before $t = 1000$ form a delivery batch, this batch's transportation cost is excluded from the cost computation. Figure 8.11 illustrates this performance indicator and partitions jobs as counted and excluded jobs. Moreover, Table 8.13 reports the results of this performance indicator as average objective values and average objective gaps compared to the static solutions in the first two rows of a cell. Overall, using this performance indicator reduced the relative difference between static and dynamic solutions on relaxed instances compared to the values stated in Table 8.12. Therefore, the high total cost of the dynamic schedules was caused predominantly at the beginning of the dynamic schedules. This finding is in line with the two representative illustrations in Figures 8.9 and 8.10. The workload of the WBOT, WBPT, and WBPI approaches significantly differs from

Figure 8.11

Performance Indicator with Warm-Up



Note. Job j_5 completes in the counted-interval but starts during the warm-up phase and thus is not counted.

Table 8.13

Results for the First Indicator: Comparison of Different Workload-Balancing Configurations Applied to all 144 Dynamic Instances with 50% Dynamic Arrivals and Medium Time Settings with Costs Measured First After 1,000 TU

Instances		**Static**	**RTC configuration**				
			WBN	WBOT	WBPT	WBOI	WBPI
High	Relaxed	28830.03*	32210.81	30880.92	31601.22	33775.67	34345.92
		0.00	11.96	7.58	8.72	17.26	18.78
		37.94	39.44	34.11	33.83	38.00	37.25
		0.00	7.51	20.07	23.28	17.15	21.52
High	Tight	30102.03	26741.86	32574.39	31883.75	33158.61	32087.69
		0.00	-9.45	8.12	6.36	9.73	6.36
		32.14	31.81	30.47	30.42	31.19	30.86
		0.00	-8.28	14.04	12.46	13.15	11.01
Low	Relaxed	15888.53	18372.86	18074.25	19080.53	19281.06	20208.28
		0.00	16.86	16.22	23.01	24.06	30.83
		37.06	39.03	34.00	33.67	36.64	36.61
		0.00	10.81	27.08	37.92	25.70	33.48
Low	Tight	16508.14	13239.31	18816.39	17832.00	19035.58	18041.83
		0.00	-18.33	14.46	6.97	15.40	8.59
		31.89	31.56	30.58	30.19	31.22	30.92
		0.00	-17.21	19.26	12.96	17.99	12.27

* Total cost, objective gap to static objective, number of jobs evaluated, average gap per evaluated job

the initial statically scheduled workload. This indicator favors the early scheduling of jobs, as it excludes all jobs that started before $t = 1{,}000$ from the total cost sum. This design favors the two total workload methods WBOT and WBPT, as they overestimate the initially required workload. Furthermore, the rejection of orders still lowers the total cost contribution of the dynamic schedules. For this reason, the table includes the average number of evaluated jobs by the performance indicator as a third value in each cell. Accordingly, the fourth value in each cell is the average gap per evaluated job. These corrected values indicate a better performance of the interval approaches than the total workload approaches on relaxed instances and similar performance on tight instances. Only the WBOI configuration closely mimics the static schedule's workload profile; hence, the number of evaluated orders is relatively close to the number attained by static scheduling. Interestingly, the cost difference between the solution approaches mostly concerns the holding costs because, since on average, the transportation costs are similar between static and dynamic solutions.

The *second indicator* evaluates the jobs commonly scheduled in all configurations. Let $N(s(x))$ denote the jobs scheduled (accepted) in schedule s for instance x. Given the six generated schedules (Static, WBN, WBOT, WBPT, WBOI, and WBPI) $s_i(x)$ with $i \in 1, \dots, 6$ for instance x, the evaluation procedure first computes the intersection of scheduled jobs of the six schedules and then counts these job-contributions while excluding the contribution of the remaining jobs. As for the first indicator, the exclusion also affects delivery batches that do not comprise at least one intersection job.

Table 8.14 presents the results of applying this method and the results of combining the first two indicators. This third (composite) indicator only measures the cost contributions of intersection-jobs that start processing at or later than $t = 1{,}000$. In detail, the approach computes the intersection of scheduled jobs with or without warm-up ($t \in \{0, 1{,}000\}$) as follows:

$$N_t^{\cap}(x) = \bigcap_{i=1}^{6} \left\{ j \mid j \in N(s_i(x)), S_j \geqslant t \right\} .$$

The table displays the mean objective gaps to the static objective and the mean number of commonly scheduled jobs for the second (Time = 0) and third (Time = 1,000) indicators. The computed intersections were mostly determined by the WBN configuration, which rejected more jobs than the other dynamic configurations. By applying the second indicator, the interval approaches WBOI and WBPI seem to be superior to the total workload approaches WBN and WBOT. The jobs were generally scheduled more efficiently by the interval approaches. This result is in line with the relative performance between the dynamic configuration presented in Table 8.12. For the third indicator, the number of commonly scheduled jobs was lower. The total workload configurations slightly outperformed the interval configurations on relaxed instances, while the opposite was true on tight instances. As for the first indicator, the warm-up phase favored the early scheduling of jobs, which resulted in more efficient scheduling of later arriving jobs by the approaches WBOT and WBPT.

Table 8.14

Second and Third Indicators: Comparison of Different Workload-Balancing Methods Applied to all 144 Dynamic Instances with Half of the Jobs Known in Advance and Medium Time Settings with Costs Measured after 0/1,000 TU and for Common Jobs

Instances		**Time**	**Common**	**RTC configuration**				
				WBN	WBOT	WBPT	WBOI	WBPI
High	Relaxed	0	49.06	3.39*	39.33	42.29	15.16	23.06
	Tight		44.14	-24.70	11.44	11.01	4.46	5.35
Low	Relaxed		49.17	4.89	66.90	75.95	28.58	45.86
	Tight		44.89	-34.99	22.09	21.21	10.30	13.53
High	Relaxed	1000	29.83	8.77	12.09	14.98	16.28	18.23
	Tight		22.28	-8.79	8.46	5.58	6.48	5.47
Low	Relaxed		29.72	12.72	21.14	30.14	23.62	28.83
	Tight		23.22	-17.78	15.42	8.82	12.75	9.10

*objective gap to static objective measured after 0/1,000 TU for a common job set

Time = Earliest time of measurement; Common = Average number of evaluated jobs

8.7.3 Comparison with Static Schedules

This section presents a more elaborate method of evaluating the RTC approach compared to the previous indicators. The evaluation method presented below compares each dynamically generated schedule to an optimized schedule that comprises all accepted jobs and dismisses dynamically rejected jobs from schedule generation. Specifically, for each dynamic schedule $s^{d}(\check{x})$ with accepted jobs $N_0(s(\check{x}))$ for dynamic instance $\check{x}$ and its corresponding static instance x, the evaluation approach builds a new static instance x'. This instance is identical to x except for omitting the rejected jobs $N \setminus N_0(s(\check{x}))$. Afterward, the SFDDHT-GRASP solver computes a static solution $s(x')$ with a limit of 15,000 iterations. This solution acts as a benchmark for efficiently scheduling the accepted jobs, which is expected to outperform the dynamic schedules due to the full information reveal. The limit of 15,000 iterations turned out to always be sufficient to generate a better schedule. Table 8.15 displays the results of this comparison. The WBN configuration with balancing deactivated attained mean objective values that are only about 5% higher than the static objective value. This result indicates that the continuous re-optimization during the application of the RTC approach produces dynamic schedules of a high quality. In contrast, the solution quality suffered from applying the balancing methods. The left-shifting (i.e., earlier production) of jobs led to a significant increase in holding costs. Note that the interval balancing methods interfered less with the scheduling of jobs than the total balancing methods. The early machine allocations recommended by the WBOT and WBPT configurations led to remarkably high costs on relaxed instances. The applied evaluation method demonstrates the high impact on workload balancing for request integration. The workload overestimation for the total workload balancing configurations does not appear suitable in scenarios with high scheduling costs, compared to the downside of rejecting orders. Section 8.7.4 discusses the trade-off between scheduling costs and order rejection costs.

Table 8.15

Objective Gaps of the RTC Configuration's Statically Produced Solutions for the Set of Accepted Jobs by the Individual Configuration

Instances		**RTC configuration**				
		WBN	WBOT	WBPT	WBOI	WBPI
High	Relaxed	4.78*	37.64	39.94	13.97	21.64
		13.89	84.72	88.03	31.36	51.64
		-2.50	7.08	7.00	4.11	1.56
High	Tight	4.31	18.64	20.28	17.83	18.58
		7.17	47.42	44.53	42.69	42.39
		2.17	-3.61	0.75	-0.42	1.22
Low	Relaxed	6.08	64.22	72.69	27.58	44.11
		14.03	111.72	129.75	48.14	83.25
		-3.22	-0.61	-1.28	1.06	-4.00
Low	Tight	5.53	36.03	38.00	33.97	33.92
		10.47	54.86	59.67	54.72	55.61
		-0.64	-4.36	-4.53	-3.53	-5.31

* Total cost gap, holding cost gap, transportation cost gap

8.7.4 Analysis for Flexible and Inflexible Environments

The following analysis distinguishes between *flexible* and *inflexible environments*. In inflexible environments, order rejection must be avoided, as rejecting an order is extremely expensive due to contractual obligations or product expiration. In the latter case, expiration imposes additional opportunity and disposal costs. Ranking the configurations in this context is straight forward because solution quality can be measured hierarchically. The already presented analysis in previous sections leads to the conclusion that the total workload configurations perform best in this setting despite the high scheduling costs. The workload overestimation helps to ensure dynamic order acceptance.

The discussion regarding flexible environments is more nuanced than for inflexible environments. Flexible environments require the joint optimization of scheduling and rejection costs. The general assumption in this setting is that the products of rejected orders can be produced either in a subsequent planning horizon (e.g., day) or at additional monetary costs by, for example, using overtime capacity or outsourcing.

The following analysis measures the performance of the already generated dynamic schedules by adding to the attained objective function value a rejection penalty that depends on the number of rejected orders. Without discussing the costs of rejecting orders in detail, the subsequent analysis assumes that each order's rejection cost is proportional to the estimated average holding cost for a scheduled job. This value was calculated from the best-performing static schedule for each instance. On average, the holding costs per job for the four instance groups were as follows:

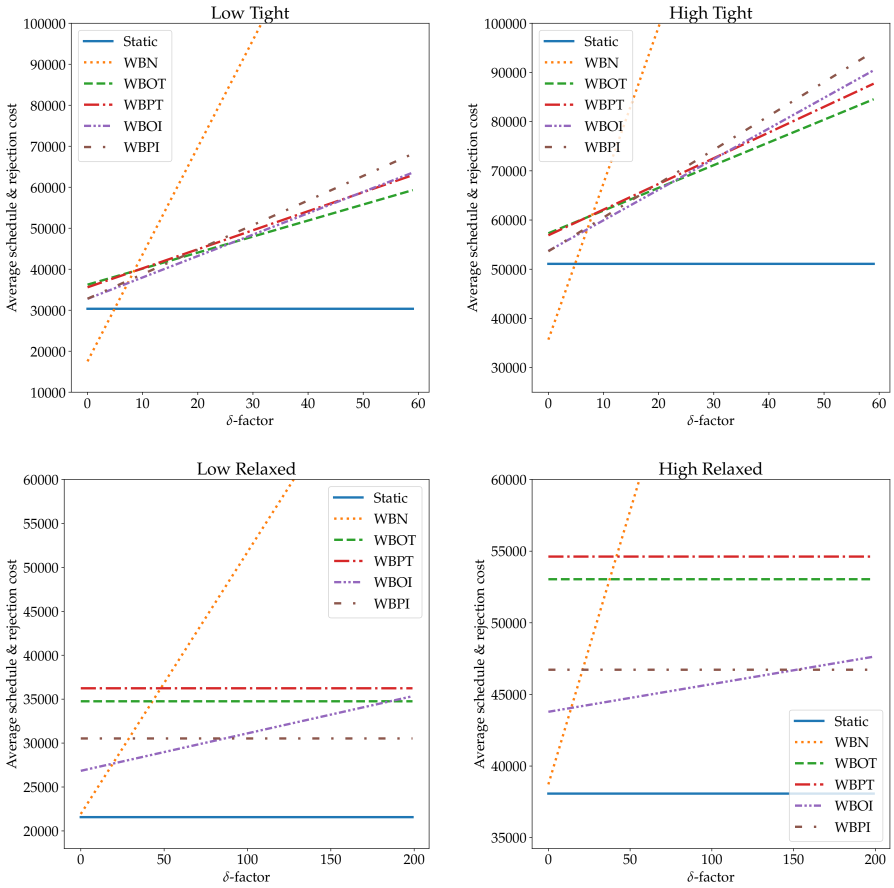
- (Low/Relaxed): 326.47;
- (High/Relaxed): 399.34;
- (Low/Tight): 490.18 ;
- (High/Tight): 545.14.

To analyze the obtained results in this context, Figure 8.12 displays the average sum of schedule and rejection costs for the four instance groups with an order rejection severity factor (δ) that scales the average holding cost value. For $\delta = 1$, a dynamically generated solution with n rejected orders is rated with the additional rejection cost value n-times the average holding cost value. For $\delta > 1$, the additional rejection cost value is δn-times the average holding cost value.

The cost values for all dynamic schedules for the five RTC configurations with δ-factors ranging from 0 to 200 were computed. Thereafter, the mean objective values for the 36 instances were computed for each of the four instance groups for each δ-factor. Figure 8.12 displays the costs for δ-factors up to 60 for the tight instances. Up to roughly a δ-factor of 10, the total cost of the WBN configuration was lower for *low tight* and *high tight* instances compared to the values of the workload-balancing approaches. Beyond this mark, rejection costs became increasingly prohibitive for the WBN configuration. The interval workload method performed better than the total workload method up to a δ-factor of about 20, as the scheduling cost was comparatively lower, but the rejection rate was slightly higher. Furthermore, on the instances with relaxed time windows, the WBN

Figure 8.12

Performance of the RTC Configurations with Rejection Costs



Note. Each graph displays the result for a combination of the time-window tightness (low/high) and the transportation costs (low/high). The plotted lines display the total costs that include the scheduling and rejection costs by the chosen δ-value (Delta).

configuration's performance was comparable to the instances with tight time windows. The only balancing configuration that rejected orders was the WBOI method. Hence, for exceptionally high rejection costs, the method performed worse than the other balancing approaches, although it attained the lowest scheduling costs on average.

From these results, one may draw conclusions regarding the applicability of the approaches for different scenarios. In environments with tight time windows and high rejection costs, balancing is required. Although the no-balancing configuration attained lower costs for low δ-factors, one must consider that the rejected jobs still had to be scheduled during subsequent planning horizons and, therefore would contribute additional costs. For instances with relaxed time-windows, the rejection rate is low for all approaches. Hence, deactivating balancing mechanisms when rejection costs are relatively low is a valid option. Similar to prior discussed results, the total workload configuration's schedule performance was relatively poor for this type of instance. As the WBPI configuration, like the total workload approaches, accepted all orders, it displayed superior performance for scenarios with high rejection costs. Overall, the WBOI configuration is the best-performing balancing configuration for relatively low rejection costs for tight and relaxed instances because of its low scheduling costs.

8.8 Comparison of Different Real-Time Control Time Settings

This section discusses the effect of varying the time settings of the RTC approach by presenting the results of the RTC approach executed on the 144 50% instances with small, medium, and large time settings and with optimization as well as both integration strategies active. Specifically, the RTC approach was also executed with the short and long settings defined in Section 8.6 to investigate the effect of shortening or lengthening the time available for optimization during an anticipation horizon. Table 8.16 summarizes the results of running the RTC approach with the different time settings and displays the objective gaps and the mean number of rejected jobs. The table does not indicate a significant difference in the average number of rejected orders or objective values for the different settings. This result is not surprising, since the results from Section 8.5 revealed that the SFDDHT-GRASP quickly finds its best solution. This result may not hold for larger instances, as the number of computable GRASP iterations decreases with an increase in

Table 8.16

Gaps and Rejected Jobs for Different Time Settings

Instances		opt.-time	RTC configuration				
			WBN	WBOT	WBPT	WBOI	WBPI
High	Relaxed	short	1.32	39.05	41.48	15.25	21.61
			1.06	0.00	0.00	0.06	0.00
		mid	1.89	38.93	41.95	15.31	22.77
			0.94	0.00	0.00	0.03	0.00
		long	1.91	39.16	41.11	14.90	21.97
			0.94	0.00	0.00	0.03	0.00
High	Tight	short	-28.28	12.06	10.71	5.39	5.86
			5.61	0.75	0.92	1.14	1.33
		mid	-28.46	11.86	11.24	5.34	5.03
			5.64	0.78	0.89	1.14	1.31
		long	-28.75	12.38	10.99	5.83	5.22
			5.58	0.72	0.86	1.17	1.28
Low	Relaxed	short	3.51	65.80	77.05	31.48	45.57
			0.92	0.00	0.00	0.06	0.00
		mid	3.31	66.64	75.32	29.02	46.03
			0.83	0.00	0.00	0.06	0.00
		long	2.90	67.54	77.05	30.72	44.92
			0.86	0.00	0.00	0.06	0.00
Low	Tight	short	-38.88	21.33	19.16	10.51	12.05
			5.00	0.72	0.89	1.06	1.08
		mid	-39.35	21.50	19.39	10.26	11.49
			4.89	0.69	0.89	1.06	1.17
		long	-39.22	21.70	19.04	10.13	11.63
			5.06	0.72	0.86	1.06	1.11

* objective gap, rejected orders

the number of jobs to schedule. Within the scope of the conducted experiments, one can conclude that the medium setting's optimization time is well suited.

8.9 Evaluation of the Real-Time Control Approach Executed on Instances with an Increased Number of Dynamic Job Arrivals

In addition to dynamically computing solutions for the instances with 50% dynamic arrivals, the program executed the five workload-balancing configurations for the problem instances with 76% dynamic orders. Similar to the previous experiments, the RTC was performed with optimization as well as both integration strategies active. Table 8.17 summarizes the results for this experiment. Each cell displays the average objective gap, the difference to the objective gap for the 50% instances in percentage points, the average number of rejected jobs, and the difference to the average number of rejected jobs for the 50% instances in percentage points. As expected, on average, The WBN configuration rejected more orders with the more difficult setting. The attained objective values also increased, except for the high/relaxed instances. In this case, the decrease in the mean cost value stemmed from a higher rejection rate. Overall, the performance of the workload-balancing methods did not drastically change for all instance types. Interestingly, the average number of rejected jobs decreased for the tight instance-set with

Table 8.17

RTC Results with 38 Dynamically Arriving Orders

Instances		RTC configuration				
		WBN	WBOT	WBPT	WBOI	WBPI
High	Relaxed	1.35*	40.85	43.13	16.58	22.5
		-0.54	+1.92	+1.18	+1.27	-0.27
		1.58	0.00	0.00	0.03	0.03
		+0.64	+0.00	+0.00	+0.00	+0.03
High	Tight	-29.39	12.00	11.68	6.84	5.57
		-0.93	+0.14	+0.44	+1.50	+0.54
		6.47	0.67	0.86	1.06	1.25
		+0.83	-0.11	-0.03	-0.08	-0.06
Low	Relaxed	4.83	67.59	77.41	33.99	46.56
		+1.52	+0.95	+2.09	+4.97	+0.53
		1.44	0.00	0.00	0.06	0.00
		+0.61	+0.00	+0.00	+0.00	+0.00
Low	Tight	-40.92	21.02	18.35	12.49	11.87
		-1.57	-0.48	-1.04	+2.23	0.38
		6.42	0.72	0.92	1.11	1.28
		+1.53	+0.03	+0.03	+0.05	+0.11

* objective gap, difference to the objective gap of the 50% instances in percentage points, rejected jobs, difference to the rejected jobs of the 50% instances in percentage points

** no rejections in the 50% solutions

high transportation costs, while the number slightly increased for tight instances with low transportation costs. As the difference is only marginal, this observation can be attributed to randomness. The results suggest that the RTC approach with workload balancing active is well suited for situations with a higher quantity of dynamic arrivals.

8.10 Evaluation of the Optimization Performance

The last experiment evaluated the applied optimization approach's effectiveness during the execution of the RTC. Table 8.18 compares the execution of the RTC approach in the WBPI configuration with three optimization variations executed on the 50% instances with medium time settings. The first RTC column (OPT+OI+LCI) displays the results with optimization and both integration strategies active, while the second column (OPT+LCI) shows the results with the optimized integration deactivated, and the last column (LCI) presents the results for the RTC approach without the optimizer (for integration and schedule optimization). The LCI configuration applies only the least-cost insertion procedure to integrate new requests. The results indicate that the optimized integration slightly outperformed the configuration without optimized integration – rejections were less frequent with this option activated. Without the optimizer, the integration of new job requests often failed. A consequent conclusion is that a simple schedule extension by insertion heuristics is unlikely to work in the considered dynamic environment. Instead,

Table 8.18

Results for Different Optimization Configurations

Instances		static	RTC (WBPI)		
			OPT+OI+LCI	OPT+LCI	LCI
High	Relaxed	38077.36	46724.33	46308.06	44899.92
		0.00	22.77	21.61	18.97
		19966.94	27948.69	27629.53	25839.56
		18110.42	18775.64	18678.53	19060.36
		0.00	0.00	0.06	0.89
High	Tight	51090.36	53642.50	54778.06	38286.89
		0.00	5.03	6.95	-23.52
		27256.78	29912.56	31094.50	17291.75
		23833.58	23729.94	23683.56	20995.14
		0.00	1.31	1.53	5.47
Low	Relaxed	21569.56	30525.97	30171.44	27115.14
		0.00	46.03	44.68	32.53
		16323.42	25404.83	25010.03	22029.61
		5246.14	5121.14	5161.42	5085.53
		0.00	0.00	0.00	1.08
Low	Tight	30350.36	32807.00	33500.56	20068.25
		0.00	11.49	13.78	-29.27
		24508.81	27183.28	27851.42	14707.22
		5841.56	5623.72	5649.14	5361.03
		0.00	1.17	1.33	5.19

** Total cost, objective gap to best-found static objective, holding cost, transportation cost, and rejected jobs* extensive rescheduling appears to be necessary to integrate new jobs and provide feasible plans. This result justifies the application of the real-time approach that utilizes the SFDDHT-GRASP to propose dynamic schedules.

8.11 Conclusion

This chapter presented the performed computational experiments for the developed approaches on specifically designed problem instances. This section summarizes the key findings of the study.

The proposed optimization algorithms SFDDHT-B&B and SFDDHT-GRASP and the RTC approach were implemented in C++ to perform on specifically designed problem instances. Furthermore, the CPLEX Solver was used to solve corresponding MILPs to provide a benchmark for the SFDDHT-B&B algorithm. The problem instances were designed to vary instance characteristics to evaluate whether they influence the approaches' performance. Overall, 1,224 problem instances were generated for the static problem SFDDHT. The computational study for the RTC approach modified the large static problem instances with 50 jobs to incorporate dynamic arrival times. The employed generation procedure enabled a comparison between dynamically generated and corresponding statically generated schedules such that the performance of the RTC approach could be evaluated in detail. The problem of dynamic order integration was investigated using five different RTC workload-balancing configurations, for which workload recommendations were generated from the static instances and best-performing schedules.

The developed SFDDHT-B&B algorithm attained encouraging results on the instances with tight time windows and decent results for the instances with relaxed time windows. The algorithm solved all instances with tight time windows and up to 40 jobs in seconds, and 77.78% of the 30 job instances with relaxed time windows. With the initially provided solution of the SFDDHT-GRASP, the algorithm computed feasible solutions for all 1,224 generated instances. The exact algorithm attained the best results with the configuration that applies LB II. The B&B algorithm benefited from its fast computation and high bounding values compared to LB I and III. Moreover, although LB III generally provided the weakest bound values of the three bounds, its fast computing enabled the B&B approach to obtain similar results as with the application of LB II. The application of LB I (also in combination with LB II) did not perform well. For large instances, the cubic computational complexity of the Hungarian method resulted in too long computational times compared to the attained bound quality.

The developed B&B algorithm clearly dominates the CPLEX solver (which solves the generated MILPs) in all aspects. Solving the MILP formulation within the time limit of 1 hour became challenging for instances with 14/16 jobs for relaxed/tight time windows. Furthermore, the solver could only provide feasible solutions for 46.16% of the generated solutions. Therefore, this solution method can be ruled out from practical usability.

The SFDDHT-GRASP delivered promising results in all aspects. The solution quality was high, with a mean gap to the best-found solution of under 1% with many instances solved optimally. Although the finding of feasible solutions is a complex task, the simple repair procedure proved to be highly effective. The heuristic was shown to quickly provide a large sample size of high-quality solutions, thus making the heuristic especially suitable for dynamic applications.

Based on the results of the RTC approach, it performed well in terms of dynamic order acceptance rate and scheduling costs. The two integration strategies coupled with the workload-balancing methods fulfilled their purpose of improving the order acceptance rate compared to no balancing. All *four* tested balancing methods reduced the average number of rejected orders successfully below *one* for instances with 25 dynamic orders. The results suggest that workload balancing with *correct* workload values is instrumental in integrating new orders and attaining reasonable scheduling costs. The RTC approach performed well with short anticipation horizons and an increasing proportion of dynamic orders that caused a more considerable degree of uncertainty.

The testing of the different workload balancing methods revealed that the usefulness of the different approaches depends on the instance characteristics. The total workload method appears to be sufficient when the workload remains relatively constant throughout the planning horizon. Therefore, this method attained decent results on tight instances. Moreover, deactivating workload balancing also performed well on tight instances. On the one hand, the mean workload of static schedules for the tight instances was 97% (see Table 8.3); that is, only 3% of machine time was not utilized. Therefore, balancing had a smaller effect on schedule generation in these cases. On the other hand, in relaxed instances, the interval methods WBOI and WBPI could utilize the more accurate information for fitting workload ratios. In these cases, the generated scheduling costs turned out to be acceptable when considering the high request acceptance rate.

Overall, the performance of the RTC approach highlights the need for powerful optimization frameworks in dynamic settings where order integration is not trivial due to time window constraints. The integration of dynamic requests by insertion and optimization successfully integrated dynamic orders in tight settings.

Chapter 9

Summary and Outlook

This chapter presents the conclusions of this dissertation. Section 9.1 contains a summary of the contents and highlights the attained results. Lastly, Section 9.2 discusses the limitations of the research and elaborates on future research possibilities.

9.1 Summary and Research Findings

Coordinating production and distribution processes in the supply chain is a topic of high importance and opens up immense opportunities for optimization. However, no widespread research is available that aims at the integration of production and distribution processes on an operative level. In particular, the detailed scheduling of products and their outbound distribution by 3PL providers offers many research opportunities due to the lack of developed models and solution approaches. The developed models and methods that were presented in this dissertation contribute to this research field.

The research aim of this dissertation, stated in Section 1.2, was to provide insights into the application of OR methods for the considered IPODS-FD that provide value as is and are easily extendable towards more complex applications. This aim was accomplished by developing the optimization methods described in Chapters 4 to 7. The proposed models and methods cover scenarios with hard time-window restrictions for the joint optimization of final product holding costs and batch delivery transportation costs. The proposed methods were explained in detail to allow for their implementation and application in a real-world system or extension for related applications. Chapter 8 analyzed the results of the conducted computational experiments in detail to indicate the strengths and limitations of the proposed approaches.

The contents and findings of the individual chapters are summarized next. Chapter 1 introduced the topic of IPODS with fixed delivery departure times. The research was categorized as a specific line of research within the field of OR. Furthermore, the chapter stated the aims and objectives of this work and provided a brief outline of the contents. Chapter 2 introduced the selected OR topics and the basic scheduling terminology and concepts.

Chapter 3 introduced IPODS-FD models and the current state of research. Moreover, a classification scheme was presented to categorize the different scheduling models. This also chapter discussed the differences and similarities between the developed models to date and their applications. Specifically, models can be differentiated not only in terms of the structure of delivery dates but also by their objective. From reviewing the existing literature, it is evident that on a grand scale, the detailed scheduling of production and distribution involving 3PL providers is still a niche topic with relatively few contributions. The contemporary research mostly focuses on deterministic single-machine scheduling problems, and therefore neglects not only uncertainty in dynamic settings but also more complex machine environments.

Chapter 4 presented a novel scheduling problem (the SFDDHT) that models the joint optimization of finished goods' holding costs and batch delivery transportation costs in a setting with fixed delivery departure times and hard time-window restrictions. The model is the first of its kind that combines these complex aspects into a single formulation. Its application areas are JIT production in MTO or BTO supply chains that usually try to avoid excessive inventories of components and materials and that rely on 3PL distribution. The formulation differs from already published scheduling models by having to optimize schedules with hard production and delivery time windows and by considering the presence of idle time in optimal schedules. These aspects are not considered by the related research of Li et al. (2017). Moreover, the modeling of delivery bundling extends Leung and Chen's (2013) research by also considering inventory holding costs. The combined consideration of the conflicting objectives results in special properties of optimal schedules that are newly discussed in this dissertation. Section 4.3 presented a MILP formulation for the SFDDHT. This formulation enables the solving of SFDDHT instances with available optimizer suites by translating the mathematical formulation to the applied optimizer syntax. The formulation unambiguously defines the problem and is a starting point for extending the SFDDHT towards problem variations. In particular, the objective function can easily be modified to capture different cost assumptions, as detailed scheduling and transportation decisions are already expressed by the defined decision variables. The **strongly $\mathcal{NP}$-hard** complexity status of the problem motivated the development of the B&B and GRASP approaches in the two subsequent chapters to provide optimal and high-quality heuristic solutions, respectively.

The developed B&B procedure presented in Chapter 5 utilizes several specific problem and dominance properties and lower bounds to reduce the explicitly searched subproblem search space during the B&B procedure. The optimality properties of schedules due to the objective function enabled the reduction of the searched subproblem space to (partial) canonical schedules, which in turn provided the basis to branch on a manageable set of combinatorial branching decisions. Furthermore, the cost structure and feasibility properties further enabled the construction of branch-reduction procedures in the form of preprocessing procedures, a dynamic feasibility test, and the exclusion of dominated delivery decisions. It must be noted that many of the results are applicable to other optimization methods, such as the GRASP presented in the subsequent chapter, to heuristically solve problem instances. The combination of all developed procedures into the B&B framework provides a superior alternative to the standard solver approach, which the conducted computational experiments clearly show. Many of the presented techniques concern only parts of the problem formulation, and are therefore reusable for related problems and direct extensions. Note that the proposed B&B approach is one of the few exact solution approaches for an $\mathcal{NP}$**-hard** IPODS-FD. The research by Mensendiek et al. (2015) (B&B) and Li et al. (2017) (CG) form the only other solution approaches of this type.

The SFDDHT-GRASP described in Chapter 6 effectively applies insertion and reinsertion operators in combination with the efficient canonical scheduling procedure to sample the solution space. A specifically designed repair procedure generated feasible solutions, which is a difficult task for instances with restrictive production time windows. Moreover, the parallel implementation with path relinking offers an easy-to-implement, scalable heuristic solution for the problem. The heuristic provided high-quality initial upper bounds for the B&B procedure that often turned out to be optimal. Similar to the B&B algorithm, most parts of the heuristic can be tailored to related optimization problems, as long as the sequencing and date assignment decisions, in combination with an efficient scheduling procedure, are sufficient to define problem solutions.

The D-SFDDHT with dynamic job arrivals was presented in Chapter 7. The proposed RTC approach to dynamically propose schedules uses the framework by Bock (2010). In this context, a transformation of the real-time situation to statically solvable instances was developed, as well as the fitting of the SFDDHT-GRASP to provide high-quality schedules quickly. The integration of new requests was handled by an insertion and optimization strategy. Further improvement of the order acceptance rate could be established by using specifically designed workload-balancing methods.

The conducted computational experiments in Chapter 8 used specifically designed instances to evaluate the performance of the designed approaches. The constructed B&B algorithm turned out to be a valid exact optimization approach for small-to medium-sized instances, especially for instances with tight time windows. The designed SFDDHT-GRASP was shown to be a powerful heuristic that attains solutions of high-quality in very little time. Both aspects make the heuristic an appropriate tool for dynamic optimization with small response times to changes in the environment. The RTC approach offers schedules of high quality and successfully integrates new order requests by applying workload-balancing methods.

9.2 Limitations and Future Research

The proposed models SFDDHT and D-SFDDHT primarily focus on the conflicting objectives of minimizing holding and batch delivery transportation costs. This simplified cost structure has its uses for problem domains in which both aspects primarily influence the total scheduling costs. However, in practice the transportation cost is often influenced by other factors, such as choosing different transportation modes or having different product weights and sizes. Furthermore, the proposed models assume transportation capacity to be sufficient or nonrestrictive. Modeling the mentioned aspects would enhance the applicability of generated solution approaches to numerous additional scenarios. However, many of the established results in Chapter 5 no longer hold in these settings, which might reduce the chances of generating optimal schedules in a reasonable amount of time.

In addition to extensions concerning the delivery of produced products, another possible future research direction is the modeling of more complex production environments. Most models in the literature (including the ones explored in this dissertation) are concerned with single-machine environments. Future models could hence cover more complex machine models, such as job-shop, flow-shop, and open-shop models or even assembly line sequencing.

The RTC approach described in Chapter 7 handles only dynamic job arrivals as the single source of dynamism. This limitation was purposefully imposed to have better control over the dynamic influence in order to evaluate the implemented approach. For practical applications, one must guarantee that a real-time approach handles many types of dynamic events. This includes, for example, the delay or cancellation of pickups and an extension of release dates due to delays in previous stages or because of material availability or machine breakdowns. Furthermore, the assumption that the proposed plan is followed without any variations is rather utopic.

The prediction of the required workload in the dynamic setting is a topic that was not covered in this work, although an accurate prediction of the workload turned out to be a critical aspect for applying the RTC approach during the computational study. In practice, it is unrealistic to know these values exactly; therefore, one must accurately predict these values. The simplistic WBPI approach that identifies *related* instances did not quite fulfill this requirement. For practical applications, a suitable approach might match the current situation to workload patterns identified from past historical data (c.f. Ferrucci and Bock, 2016). In this context, machine learning algorithms can be used to supplement dynamic optimization approaches.

The proposed RTC approach attempts to integrate new orders without considering the impact on the scheduling costs of producing an additional order′s products. As shown by the experiments in Chapter 8, rejecting orders reduces the scheduling costs immensely, foremost due to a reduction in holding costs. With additional information about the cost of rejecting orders, one could modify the approach to handle flexible environments such that the RTC procedure refuses to produce products of orders for cost reasons.

Lastly, while this dissertation contributes to the existing research, numerous not-yet-analyzed IPODS-FD applications still exist that are worth investigating.

Bibliography

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

Armstrong, R., Gao, S., and Lei, L. (2008). A zero-inventory production and distribution problem with a fixed customer sequence. *Annals of Operations Research*, 159(1):395–414.

Bachtenkirch, D. and Bock, S. (2022). Finding efficient make-to-order production and batch delivery schedules. *European Journal of Operational Research*, 297(1):133–152.

Baker, K. R. and Su, Z.-S. (1974). Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Research Logistics Quarterly*, 21(1):171–176.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716.

Bellman, R. (1953). An introduction to the theory of dynamic programming. Technical report, RAND CORP SANTA MONICA CA.

Belouadah, H. and Potts, C. N. (1994). Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Applied Mathematics*, 48(3):201–218.

Bilgen, B. and Ozkarahan, I. (2004). Strategic tactical and operational production-distribution models: a review. *International Journal of Technology Management*, 28(2):151–171.

Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., and Weglarz, J. (2019). *Handbook on scheduling*. Springer.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.

Bock (2004). *Echtzeitfähige Steuerung von Speditionsnetzwerken*. Deutscher Universitätsverlag.

Bock, S. (2010). Real-time control of freight forwarder transportation networks by integrating multimodal transport chains. *European Journal of Operational Research*, 200(3):733–746.

Bodin, L. (1981). *The state of the art in the routing and scheduling of vehicles and crews*, volume 1. Office of Policy Research, Urban Mass Transportation Administration.

Boysen, N., Briskorn, D., and Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, 258(1):343–357.

Boysen, N., Fliedner, M., and Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research*, 183(2):674–693.

Brucker, P. (2007). *Scheduling algorithms*, volume 3. Springer, Berlin, Heidelberg.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41.

Bruno, J., Coffman Jr, E., and Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17(7):382–387.

Carrera, S., Ramdane-Cherif, W., and Portmann, M.-C. (2010a). Scheduling problems for logistic platforms with fixed staircase component arrivals and various deliveries hypotheses. In *2nd International Conference on Applied Operational Research-ICAOR 2010*, volume 2, pages 517–528.

Carrera, S., Ramdane-Cherif, W., and Portmann, M.-C. (2010b). Scheduling supply chain node with fixed components arrivals and two partially flexible deliveries. *IFAC Proceedings Volumes*, 43(17):152–157.

Chand, S. and Schneeberger, H. (1988). Single machine scheduling to minimize weighted earliness subject to no tardy jobs. *European Journal of Operational Research*, 34(2):221–230.

Chen, B., Potts, C. N., and Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization*, pages 1493–1641. Springer.

Chen, Z.-L. (2004). Integrated production and distribution operations. In *Handbook of quantitative supply chain analysis*, pages 711–745. Springer.

Chen, Z.-L. (2010). Integrated production and outbound distribution scheduling: review and extensions. *Operations Research*, 58(1):130–148.

Chen, Z.-L. and Vairaktarakis, G. L. (2005). Integrated scheduling of production and distribution operations. *Management Science*, 51(4):614–628.

Chhajed, D. (1995). A fixed interval due-date scheduling problem with earliness and due-date costs. *European Journal of Operational Research*, 84(2):385–401.

Christopher, M. (2011). *Logistics & supply chain management*. Pearson Education, fourth edition.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press, third edition edition.

Crainic, T. (1993). *Parallel branch-and-bound algorithms: Survey and synthesis*. Centre for Research on Transportation= Centre de recherche sur les transports.

Dantzig, G. B. (1948). Linear programming in problems for the numerical analysis of the future. In *Proceedings of the Symposium on Modern Calculating Machinery and Numerical Methods, UCLA, July*, pages 29–31.

Dantzig, G. B., Wald, A., et al. (1951). On the fundamental lemma of neyman and pearson. *The Annals of Mathematical Statistics*, 22(1):87–93.

Devapriya, P., Ferrell, W., and Geismar, N. (2006). Optimal fleet size of an integrated production and distribution scheduling problem for a perishable product. In *IIE Annual Conference. Proceedings*, page 1. Institute of Industrial and Systems Engineers (IISE).

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159.

Eastman, W. L., Evan, S., and Isaacs, I. M. (1964). Bounds for the Optimal Scheduling of n Jobs on m Processors. *Management Science*, 11(2):268–279.

Elmaghraby, S. E. and Park, S. H. (1974). Scheduling jobs on a number of identical machines. *AIIE transactions*, 6(1):1–13.

Erengüç, Ş. S., Simpson, N. C., and Vakharia, A. J. (1999). Integrated production/distribution planning in supply chains: An invited review. *European Journal of Operational Research*, 115(2):219–236.

Fan, B. (2010). Scheduling with fixed delivery dates and temporary storage. In *2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM)*, volume 3, pages 1926–1929. IEEE.

Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71.

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

Ferrucci, F. and Bock, S. (2014). Real-time control of express pickup and delivery processes in a dynamic environment. *Transportation Research Part B: Methodological*, 63:1–14.

Ferrucci, F. and Bock, S. (2015). A general approach for controlling vehicle en-route diversions in dynamic vehicle routing problems. *Transportation Research Part B: Methodological*, 77:76–87.

Ferrucci, F. and Bock, S. (2016). Pro-active real-time routing in applications with multiple request patterns. *European Journal of Operational Research*, 253(2):356–371.

Ferrucci, F., Bock, S., and Gendreau, M. (2013). A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research*, 225(1):130–141.

Fortnow, L. (2009). The status of the p versus np problem. *Communications of the ACM*, 52(9):78–86.

Fu, B., Huo, Y., and Zhao, H. (2012). Coordinated scheduling of production and delivery with production window and delivery capacity constraints. *Theoretical Computer Science*, 422:39–51.

García, J. M. and Lozano, S. (2004). Production and vehicle scheduling for ready-mix operations. *Computers & Industrial Engineering*, 46(4):803–816.

García, J. M. and Lozano, S. (2005). Production and delivery scheduling problem with time windows. *Computers & Industrial Engineering*, 48(4):733–742.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*, volume 174. W.H. Freeman and Company, San Francisco.

Geismar, H. N., Laporte, G., Lei, L., and Sriskandarajah, C. (2008). The integrated production and transportation scheduling problem for a product with a short lifespan. *INFORMS Journal on Computing*, 20(1):21–33.

Gendreau, M., Hertz, A., Laporte, G., and Stan, M. (1998). A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. *Operations Research*, 46(3):330–335.

Gendreau, M. and Potvin, J.-Y. (2010). *Tabu Search*, pages 41–59. Springer US, Boston, MA.

Gendreau, M., Potvin, J.-Y., et al. (2010). *Handbook of metaheuristics*, volume 2. Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

Glover, F. (1997). Tabu search and adaptive memory programming–advances, applications and challenges. In *Interfaces in computer science and operations research*, pages 1–75. Springer.

Goetschalckx, M., Vidal, C. J., and Dogan, K. (2002). Modeling and design of global logistics systems: A review of integrated strategic and tactical models and design algorithms. *European Journal of Operational Research*, 143(1):1–18.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier.

Gunasekaran, A. and Ngai, E. W. (2009). Modeling and analysis of build-to-order supply chains. *European Journal of Operational Research*, 195(2):319–334.

Hall, N. G., Lesaoana, M., and Potts, C. N. (2001). Scheduling with Fixed Delivery Dates. *Operations Research*, 49(1):134–144.

Han, D., Yang, Y., Wang, D., Cheng, T., and Yin, Y. (2019). Integrated production, inventory, and outbound distribution operations with fixed departure times in a three-stage supply chain. *Transportation Research Part E: Logistics and Transportation Review*, 125:334–347.

Hillier, F. S. and Lieberman, G. J. (2012). *Introduction to operations research*. Tata McGraw-Hill Education.

Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–73.

Horowitz, E. and Sahni, S. (1976). Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM (JACM)*, 23(2):317–327.

Huang, P. Y., Rees, L. P., and Taylor, B. W. (1983). A simulation analysis of the japanese just-in-time technique (with kanbans) for a multiline, multistage production system. *Decision Sciences*, 14(3):326–344.

Ibaraki, T. (1976). Theoretical comparisons of search strategies in branch-and-bound algorithms. *International Journal of Computer & Information Sciences*, 5(4):315–344.

Ibaraki, T. (1977). The power of dominance relations in branch-and-bound algorithms. *Journal of the ACM (JACM)*, 24(2):264–279.

Ibaraki, T. (1978). Branch-and-bound procedure and state–space representation of combinatorial optimization problems. *Information and Control*, 36(1):1–27.

Ibarra, O. H. and Kim, C. E. (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468.

Ignall, E. and Schrage, L. (1965). Application of the branch and bound technique to some flow-shop scheduling problems. *Operations Research*, 13(3):400–412.

Jamili, N., Ranjbar, M., and Salari, M. (2016). A bi-objective model for integrated scheduling of production and distribution in a supply chain with order release date restrictions. *Journal of Manufacturing Systems*, 40:105–118.

Kao, G. K., Sewell, E. C., and Jacobson, S. H. (2009). A branch, bound, and remember algorithm for the $1|r_i|\sum t_i$ scheduling problem. *Journal of Scheduling*, 12(2):163.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *The Annals of Mathematical Statistics*, pages 338–354.

Knuth, D. E. (1997). *The Art of Computer Programming: Fundamental Algorithms. Fundamental Algorithms*. Addison-Wesley.

Kohler, W. H. and Steiglitz, K. (1974). Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *Journal of the ACM (JACM)*, 21(1):140–156.

Kolesar, P. J. (1967). A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Kumar, V. and Kanal, L. N. (1983). A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, 21(1-2):179–198.

Langley, C. J. and Infosys (2020). 2020 24th annual third-party logistics study: The state of logistics outsourcing.

Lawler, E. L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546.

Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719.

Lee, C.-Y., Danusaputro, S. L., and Lin, C.-S. (1991). Minimizing weighted number of tardy jobs and weighted earliness-tardiness penalties about a common due date. *Computers & Operations Research*, 18(4):379–389.

Lee, C.-Y. and Li, C.-L. (1996). On the fixed interval due-date scheduling problem. *Discrete Applied Mathematics*, 68(1-2):101–117.

Leung, J. Y.-T. and Chen, Z.-L. (2013). Integrated production and distribution with fixed delivery departure dates. *Operations Research Letters*, 41(3):290–293.

Li, F., Chen, Z.-L., and Tang, L. (2017). Integrated production, inventory and delivery problems: Complexity and algorithms. *INFORMS Journal on Computing*, 29(2):232–250.

Li, K., Ganesan, V., and Sivakumar*, A. (2005). Synchronized scheduling of assembly and multi-destination air-transportation in a consumer electronics supply chain. *International Journal of Production Research*, 43(13):2671–2685.

Li, K., Ganesan, V. K., and Sivakumar, A. I. (2006). Scheduling of single stage assembly with air transportation in a consumer electronic supply chain. *Computers & Industrial Engineering*, 51(2):264–278.

Liu, C.-H. and Hsu, C.-I. (2015). Dynamic job shop scheduling with fixed interval deliveries. *Production Engineering*, 9(3):377–391.

Ma, H., Chan, F. T., and Chung, S. (2013). Minimising earliness and tardiness by integrating production scheduling with shipping information. *International Journal of Production Research*, 51(8):2253–2267.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30.

Matsuo, H. (1988). The weighted total tardiness problem with fixed shipping times and overtime utilization. *Operations Research*, 36(2):293–307.

Mensendiek, A., Gupta, J. N., and Herrmann, J. (2015). Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness. *European Journal of Operational Research*, 243(2):514–522.

Meseguer, P. (1997). Interleaved depth-first search. In *IJCAI*, volume 97, pages 1382–1387. Citeseer.

Mitten, L. (1970). Branch-and-bound methods: General formulation and properties. *Operations Research*, 18(1):24–34.

Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109.

Morin, T. L. and Marsten, R. E. (1976). Branch-and-bound strategies for dynamic programming. *Operations Research*, 24(4):611–627.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38.

Nau, D. S., Kumar, V., and Kanal, L. (1984). General branch and bound, and its relation to a* and ao*. *Artificial Intelligence*, 23(1):29–58.

Nikolaev, A. G. and Jacobson, S. H. (2010). *Simulated Annealing*, pages 1–39. Springer US, Boston, MA.

Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417.

Padberg, M. (2013). *Linear optimization and extensions*, volume 12. Springer Science & Business Media.

Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer.

Reeves, C. R. (2010). *Genetic Algorithms*, pages 109–139. Springer US, Boston, MA.

Resende, M. G. and Ribeiro, C. C. (2005). Grasp with path-relinking: Recent advances and applications. In *Metaheuristics: progress as real problem solvers*, pages 29–63. Springer.

Resende, M. G. and Ribeiro, C. C. (2010). *Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications*, pages 283–319. Springer US, Boston, MA.

Ruf, N. and Schöbel, A. (2004). Set covering with almost consecutive ones property. *Discrete Optimization*, 1(2):215–228.

Russell, R., Chiang, W.-C., and Zepeda, D. (2008). Integrating multi-product production and distribution in newspaper logistics. *Computers & Operations Research*, 35(5):1576–1588.

Sahni, S. K. (1976). c. *Journal of the ACM (JACM)*, 23(1):116–127.

Sarmiento, A. M. and Nagi, R. (1999). A review of integrated analysis of production-distribution systems. *IIE Transactions*, 31(11):1061–1074.

Scholl, A. and Klein, R. (1999). Balancing assembly lines effectively–a computational comparison. *European Journal of Operational Research*, 114(1):50–58.

Schonberger, R. J. and Gilbert, J. P. (1983). Just-in-time purchasing: a challenge for us industry. *California Management Review*, 26(1):54–68.

Seddik, Y., Gonzales, C., and Kedad-Sidhoum, S. (2013). Single machine scheduling with delivery dates and cumulative payoffs. *Journal of Scheduling*, 16(3):313–329.

Stecke, K. E. and Zhao, X. (2007). Production and transportation integration for a make-to-order manufacturing company with a commit-to-delivery business mode. *Manufacturing & Service Operations Management*, 9(2):206–224.

Toregas, C. and Revelle, C. (1973). Binary logic solutions to a class of location problem. *Geographical Analysis*, 5(2):145–155.

Tyagi, N., Abedi, M., and Varshney, R. G. (2013). Single machine scheduling in a batch delivery system with fixed delivery dates. *International Journal of Engineering and Technology*, 5(4):3484–3488.

Wagner, T., Guralnik, V., and Phelps, J. (2003). Taems agents: Enabling dynamic distributed supply chain management. *Electronic Commerce Research and Applications*, 2(2):114–132.

Wang, Q., Batta, R., and Szczerba, R. J. (2005). Sequencing the processing of incoming mail to match an outbound truck delivery schedule. *Computers & Operations Research*, 32(7):1777–1791.

Wang, W., Xu, X., Jiang, Y., Xu, Y., Cao, Z., and Liu, S. (2020a). Integrated scheduling of intermodal transportation with seaborne arrival uncertainty and carbon emission. *Transportation Research Part D: Transport and Environment*, 88:102571.

Wang, W., Xu, X., Peng, Y., Zhou, Y., and Jiang, Y. (2020b). Integrated scheduling of port-centric supply chain: A special focus on the seaborne uncertainties. *Journal of Cleaner Production*, page 121240.

Webster, S. (1995). Weighted flow time bounds for scheduling identical processors. *European Journal of Operational Research*, 80(1):103–111.

Woeginger, G. J. (2000). When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (fptas)? *INFORMS Journal on Computing*, 12(1):57–74.

Xu, Z. and Xu, D. (2015). Single-machine scheduling with preemptive jobs and workload-dependent maintenance durations. *Operational Research*, 15(3):423–436.

Yang, X. (2000). Scheduling with generalized batch delivery dates and earliness penalties. *IIE Transactions*, 32(8):735–741.

Appendix A

Additional Results

Table A.1

Additional SFDDHT-BNB (LB I) Statistics

	n^N	Time	Nodes	Del	Block	Swap	Move	Batch	Table
Relaxed	8	0.00	221.44	169.46	30.31	24.04	0.72	0.04	13.76
	10	0.00	761.00	596.20	171.22	96.04	3.98	1.50	50.52
	12	0.00	2768.76	2231.91	646.19	381.87	26.69	1.06	188.78
	14	0.00	7059.65	5763.44	2317.87	1078.78	94.63	7.44	422.28
	16	0.46	75900.41	64892.39	19637.11	14381.56	1408.24	13.78	7317.35
	18	1.61	201495.09	173497.28	64296.54	30846.57	4129.91	35.20	17507.56
	20	44.17	2171091.71	1898265.56	412525.54	379294.99	27156.85	2118.78	554659.60
	30	2516.92	61061512.43	50583386.61	21920711.40	9990325.01	1004798.07	13643.26	12783900.94
	40	3600.00	28034695.31	20500829.47	11598111.90	4847247.18	333582.60	199.79	3711680.38
	50	3600.00	14075138.06	10391030.75	7175144.33	2340887.19	112556.79	53.29	762270.85
Tight	8	0.00	41.44	23.72	9.33	2.93	0.13	0.06	1.98
	10	0.00	44.87	24.20	12.19	2.52	0.31	0.04	3.07
	12	0.00	141.43	85.74	50.22	14.00	1.69	0.06	10.04
	14	0.00	199.63	125.65	72.80	23.15	1.74	0.15	13.76
	16	0.00	666.46	461.67	244.83	89.06	6.17	1.00	50.70
	18	0.00	1053.07	736.61	466.13	138.43	10.81	0.48	77.39
	20	0.00	2345.92	1713.29	568.26	428.15	27.32	1.85	427.19
	30	0.86	48198.18	38999.14	24479.61	5378.18	467.92	11.35	6587.82
	40	189.68	4544673.17	3936506.25	2085381.17	825592.56	80677.15	2786.61	934018.53
	50	1964.67	34194514.36	28755466.65	21618175.44	2677499.03	160852.62	1055.82	4065556.19

Time = average runtime in seconds;
Nodes = average number of created nodes;
Del = average number of deleted nodes;
Block = average number of dominated nodes by the Production Block Dominance Procedure;
Swap = average number of dominated nodes by the Permutation Block Dominance Procedure (Swap);
Move = average number of dominated nodes by the Permutation Block Dominance Procedure (Move);
Batch = average number of dominated nodes by the Delivery Batch Dominance Procedure;
Table = average number of dominated nodes by the Dominance Table

Table A.2

Additional SFDDHT-BNB (LB II) Statistics

	n^N	Time	Nodes	Del	Block	Swap	Move	Batch	Table
Relaxed	8	0.00	227.83	159.31	60.57	24.72	1.24	0.11	11.85
	10	0.00	878.43	640.76	327.20	124.50	5.98	0.30	44.54
	12	0.00	4605.57	3545.19	1991.22	650.04	47.17	1.48	213.26
	14	0.00	11655.19	9215.09	5733.41	1657.67	161.98	1.74	533.91
	16	0.00	92036.85	76831.39	50955.30	12900.56	1048.87	5.46	3531.39
	18	0.09	303190.83	254366.37	184104.13	32697.26	3086.46	4.98	14707.61
	20	1.58	748567.04	630661.62	186183.14	155370.89	8742.53	198.85	115999.06
	30	595.89	145662739.42	123239664.86	56788186.94	24373555.08	3334371.88	16267.25	26518910.54
	40	2141.65	389169218.07	306015672.47	143108570.85	92938543.29	9681129.46	8008.72	56597949.54
	50	2596.94	459827601.01	367955260.17	219914497.86	113818340.46	9515109.60	49.32	24702967.69
Tight	8	0.00	49.17	26.87	13.52	3.39	0.20	0.06	2.89
	10	0.00	49.20	25.91	14.83	2.80	0.31	0.04	3.83
	12	0.00	150.56	89.89	55.04	14.98	1.76	0.11	11.91
	14	0.00	214.80	133.44	79.85	25.19	1.80	0.15	14.50
	16	0.00	741.72	506.39	298.98	94.31	6.22	1.17	57.59
	18	0.00	1138.07	792.80	523.67	152.93	10.67	0.41	81.94
	20	0.00	2483.29	1794.14	618.54	458.76	29.17	2.15	471.12
	30	0.03	48199.50	38773.64	24727.93	5552.44	481.97	10.01	6480.60
	40	20.69	4557032.29	3953192.44	2089368.17	830659.31	81337.92	2447.00	937610.51
	50	1077.88	206182400.33	181572261.61	137456615.01	17654621.33	874002.25	1159.32	25460083.35

Time = average runtime in seconds;
Nodes = average number of created nodes;
Del = average number of deleted nodes;
Block = average number of dominated nodes by the Production Block Dominance Procedure;
Swap = average number of dominated nodes by the Permutation Block Dominance Procedure (Swap);
Move = average number of dominated nodes by the Permutation Block Dominance Procedure (Move);
Batch = average number of dominated nodes by the Delivery Batch Dominance Procedure;
Table = average number of dominated nodes by the Dominance Table

Table A.3

Additional SFDDHT-BNB (LB III) Statistics

	n^N	**Time**	**Nodes**	**Del**	**Block**	**Swap**	**Move**	**Batch**	**Table**
Relaxed	8	0.00	295.35	211.52	80.07	32.44	1.33	0.15	18.46
	10	0.00	1102.26	811.37	391.76	151.19	7.33	1.72	81.20
	12	0.00	6097.56	4730.50	2460.72	857.83	72.83	6.07	453.52
	14	0.00	16218.46	12943.50	6974.04	2715.93	261.35	5.31	1237.30
	16	0.04	150040.35	125318.20	66584.19	23854.83	2337.85	25.81	14011.96
	18	0.28	449947.81	377076.81	242328.35	57098.85	5901.76	64.35	36129.09
	20	4.35	1832410.99	1567498.99	438940.90	353979.33	24664.19	1380.25	386988.93
	30	809.29	198435790.07	168567984.71	80287453.10	32270484.99	4252045.35	56834.86	44379774.26
	40	2325.44	462901296.46	381403465.74	189786372.21	102072013.38	9225967.62	17935.39	79870981.71
	50	2741.18	550227434.31	464404731.74	286201837.92	128642715.36	7792355.46	2256.35	41765566.65
Tight	8	0.00	50.67	27.94	14.59	3.46	0.20	0.06	2.94
	10	0.00	49.41	25.98	15.09	2.76	0.31	0.07	3.89
	12	0.00	150.04	89.46	55.19	14.96	1.78	0.07	10.93
	14	0.00	213.28	132.33	80.39	24.94	1.76	0.11	15.02
	16	0.00	768.11	523.69	309.52	97.46	6.50	1.15	67.89
	18	0.00	1160.46	806.59	531.61	156.78	10.93	0.54	90.96
	20	0.00	2559.14	1848.31	657.76	474.03	29.39	2.97	515.74
	30	0.03	49869.31	40039.79	25200.68	5842.42	504.81	12.03	7046.47
	40	20.62	4625469.36	4011481.82	2117288.60	843794.78	82484.72	3061.28	959250.33
	50	1065.12	217481705.62	192399767.71	145415090.99	18136146.26	876012.26	1517.08	27618164.83

Time = average runtime in seconds;
Nodes = average number of created nodes;
Del = average number of deleted nodes;
Block = average number of dominated nodes by the Production Block Dominance Procedure;
Swap = average number of dominated nodes by the Permutation Block Dominance Procedure (Swap);
Move = average number of dominated nodes by the Permutation Block Dominance Procedure (Move);
Batch = average number of dominated nodes by the Delivery Batch Dominance Procedure;
Table = average number of dominated nodes by the Dominance Table

Table A.4

Additional SFDDHT-BNB (LB I+II) Statistics

	n^N	Time	Nodes	Del	Block	Swap	Move	Batch	Table
Relaxed	8	0.00	307.44	223.85	72.20	32.52	1.37	0.11	18.17
	10	0.00	1154.72	867.33	384.96	155.87	7.04	1.57	73.59
	12	0.00	5393.81	4192.35	2162.28	740.87	53.15	2.00	308.28
	14	0.00	12894.33	10249.48	6004.67	1869.85	180.41	7.70	703.13
	16	0.59	132154.35	111371.74	60098.24	20229.50	1823.89	14.46	9062.31
	18	2.35	390293.59	330385.72	207931.44	46816.48	5042.59	35.28	25261.74
	20	48.18	2255510.38	1961361.78	442840.83	400673.57	28208.90	2145.19	584716.26
	30	2630.76	64211365.99	52801471.53	24076096.36	10905084.60	1110925.72	12808.79	13665970.76
	40	3600.00	28710178.57	21375165.61	12662668.64	4900180.01	341104.36	191.81	3471020.75
	50	3600.00	14973110.49	11427857.71	8274186.76	2318776.60	110909.46	52.68	723932.21
Tight	8	0.00	49.94	27.28	13.67	3.44	0.20	0.06	2.96
	10	0.00	49.48	26.06	14.83	2.80	0.31	0.04	3.94
	12	0.00	151.20	90.28	55.22	15.06	1.76	0.11	12.50
	14	0.00	216.13	134.39	80.17	25.20	1.80	0.15	15.00
	16	0.00	755.80	516.78	302.54	96.37	6.31	1.24	60.48
	18	0.00	1145.72	797.69	526.70	154.07	11.04	0.48	84.57
	20	0.00	2533.69	1831.76	629.14	466.17	29.26	2.47	498.65
	30	0.93	49907.43	40162.12	25017.15	5826.42	499.12	11.38	7059.10
	40	194.72	4488654.94	3881902.78	2063511.64	813927.15	79355.35	2579.90	916837.74
	50	1979.22	40022889.12	34593146.81	27756377.88	2669937.50	172756.61	1060.51	3756772.43

Time = average runtime in seconds;
Nodes = average number of created nodes;
Del = average number of deleted nodes;
Block = average number of dominated nodes by the Production Block Dominance Procedure;
Swap = average number of dominated nodes by the Permutation Block Dominance Procedure (Swap);
Move = average number of dominated nodes by the Permutation Block Dominance Procedure (Move);
Batch = average number of dominated nodes by the Delivery Batch Dominance Procedure;
Table = average number of dominated nodes by the Dominance Table