Scheduling problems in rail transshipment yards and industrial production

Dissertation

zur Erlangung des Grades eines Doktors

rer. pol.

der Fakultät III - Wirtschaftswissenschaften, Wirtschaftsinformatik
und Wirtschaftsrecht der Universität Siegen

vorgelegt von

M.Sc. Xiyu Li

Erstgutachter: Prof. Dr. Erwin Pesch
Zweitgutachterin: Prof. Dr. Alena Otto
Drittes Mitglied der Kommission: Prof. Dr. Marc Goerigk

Datum der Disputation:
26.02.2021

Dekan der Fakultät III:
Prof. Dr. Marc Hassenzahl

Time flies, my slightly long doctoral career is coming to an end.

I would first like to thank my supervisor, teacher since my master's program, Prof. Dr. Erwin Pesch. I am very grateful for your support, help, and encouragement over the past ten years, and for the opportunity I was given to enter the field of operations research.

I would like to thank Prof. Dr. Alena Otto, the coauthor of all my papers, for your consistent support and valuable guidance throughout my studies. I learned a lot from you. It's my luck to be a colleague and friend with you.

Special thanks to my dearest parents for your unconditional tolerance, understanding and support for me. Finally, I could not have completed this dissertation without my beloved wife Lina. Thank you for your care in life, encouragement in spirit, comfort when trouble, smile when happy!

Xiyu Li

Contents

List of Figures

List of Tables

Chapter 1

Preface

"The line between disorder and order lies in logistics. . . " Sun Tzu said, who was a Chinese military strategist and lived approximately in the 6th century BC. It can be seen that the operations research (OR) thought has a long and rich history in human development. For nearly a century, OR has continuously developed and innovated in researching and solving complex practical problems. With technological progress and improved production efficiency, OR faces more new requirements and challenges in various fields. In the field of logistics, the customer's requirement for shorter lead time and on-time delivery increases continuously; in the field of manufacturing, the advancement of production technology allows companies to provide customized products to meet the different needs of customers, but at the same time they have to reduce costs to deal with price competition.

1.1 Structure of the present thesis

This thesis consists of three papers focusing on two topics (see Table 1.1).

Otto et al. (2017) and Li et al. (2019) address operations of rail container transshipment yards. The railway has played an important role since its inception. As an economical and environmentally friendly mode of transportation in medium and long-distance transportation, it plays a crucial role in many countries' transportation systems. Increase the proportion of rail transportation is also a goal of EU until 2050. Due to customers' stricter demand for accuracy and timeliness, transportation efficiency becomes more and more important. The optimization of the rail transshipment yard has also become one of the most important parts to improve rail transportation efficiency.

Otto and Li (2020) deals with the optimization of the production process for customized products. As a way to improve competitiveness, many companies have higher requirements for mass customization. They need to manufacture products of different

Table 1.1: A list of the published papers

	Co-Authors	*Journal*	*Year*
1.	Two-way bounded dynamic programming approach for operations planning in transshipment yards		
	A. Otto, E. Pesch	*Transportation Science*	*2017*
2.	Solving the single crane scheduling problem at rail transshipment yards		
	A. Otto, E. Pesch	*Discrete Applied Mathematics*	*2019*
3.	Product sequencing in multiple-piece-flow assembly lines		
	A. Otto	*Omega*	*2020*

specifications and high quality efficiently without a significant increase in cost. To achieve this goal, in addition to the improvement of information and production technology, the optimization of production and management processes can also play a very important role.

1.2 Overview of the papers

1.2.1 Two-way bounded dynamic programming approach for operations planning in transshipment yards - The paper of Otto et al. (2017)

In this paper, we consider the static crane scheduling problem (SCSP) at mixed container transshipment yards, where both rail-road container moves between trains and trucks and rail-rail container moves between trains are present. Gantry cranes are applied to load and unload trains and trucks. In SCSP, we determine working areas for every single crane and sequence the movement of containers simultaneously. The objective is to minimize the makespan, i.e., the time to serve a bundle of trains.

From a global perspective, we propose a new two-way bounded dynamic programming (TBDP) method. We can decompose this complex problem into simpler subproblems to quickly locate the bottleneck subproblem through this method. Specifically for the SCSP, we apply TBDP to decompose the SCSP into a single crane scheduling subproblem, and quickly identify the most time-consuming working area. Then we can only seek the optimal solution for the bottleneck subproblem, and for other non-bottleneck areas, a feasible solution that could be calculated by, for example, a heuristic without affecting the global optimal solution is good enough.

For the single crane scheduling subproblem, we also provide an exact algorithm Branch-and-Bound and a priority rule-based heuristic.

Finally, in the experiment, we illustrate the algorithms' effectiveness and efficiency in various situations simulated through various parameter combinations.

1.2.2 Solving the single crane scheduling problem at rail transshipment yards - The paper of Li et al. (2019)

In this paper, we investigated the single crane scheduling problem specified in the previous paper more deeply. First of all, given our objective makespan and our realistic conditions, such as that all trucks will load and unload containers in a relatively orderly manner within the agreed time windows, we propose a decomposition algorithm (DA). Through this method, we attempt to divide the initial problem into a set of jobs. For some sets, we may only need a feasible solution, not even a wise one. We only optimize the job sets that need to be optimized, to save a lot of calculation time.

Since the DA is a superstructure, we also integrate a branch-and-cut algorithm to solve the subproblems. We call the algorithm DBC as a whole. To the best of our knowledge, this is the first branch-and-cut algorithm for the single crane scheduling problem.

We demonstrate through the computational experiments that the DBC can solve real-world instances in a short time.

1.2.3 Product sequencing in multiple-piece-flow assembly lines - The paper of Otto and Li (2020)

This paper studies a product sequencing optimization problem on a multiple-piece-flow assembly line, which is a paced assembly line used to produce customized products. The traditional paced assembly line is designed initially to produce standard products. Workpieces stay on each station for the same amount of time, called cycle time, and then move to the next station. However, the modern market has higher requirements for customization. The processing time required for customized products at each workstation could vary wildly, and the products may have different due dates. In this way, the traditional single-piece assembly line will cause much idle time, increasing the product's cost significantly. The multiple-piece-flow assembly line is designed to solve this problem. The idle time can be reduced by combining several workpieces to a bundle and balancing the processing time they need at each workstation.

To the best of our knowledge, this paper is the first to discuss the optimization of multi-piece assembly lines. We proposed a mixed integer model, discussed its relationship and differences with other related classic optimization problems, and proposed some useful properties. By using these properties, we developed an iterative variable neighborhood heuristic (IVNH), which can be employed to optimize different goals. In this paper, we considered two objective functions: minimizing the idle times and weighted tardiness. Based on the real production data, we have done extensive simulation tests and a rolling-horizon planning simulation. The test results show that IVNH can find the optimal or near-optimal solution in short run times. For some small test instances with known optimal solutions, IVNH can find the most optimal solutions in less than one second, and for the remaining test instances, the gap to the optimal solution is almost negligible.

Chapter 2

Two-way bounded dynamic programming approach for operations planning in transshipment yards

We propose a two-way bounded dynamic programming (TBDP) approach to deal with situations, when it takes long to evaluate the value function in the state graph of dynamic programming. TBDP provides sharp bounds early in the solution process and identifies critical subproblems, i.e. states and transition arcs, for which the value function has to be estimated.

Based on the TBDP framework, we develop a heuristic and an exact algorithm TEMP for the static crane scheduling problem (SCSP). The SCSP refers to simultaneous yard partitioning into single crane areas and job sequencing at railway container transshipment yards, where both rail-rail and rail-road transshipments are present and rail-rail moves are short. TEMP solves instances of practically relevant size within acceptable time limits. The proposed heuristic finds optimal solutions in 90% of the cases. We recommend using the heuristic algorithm for planning very large transshipment yards, with more than five tracks a large number of container moves per crane.

2.1 Introduction

The value of world merchandise freight exports nearly tripled from 1998 to 2008 (U.S. Department of Transportation, 2010). In view of rapidly increasing freight transportation volumes, the issue of sustainability has become particularly important (de Jong et al., 2013; Kozan, 1997). Sustainable transport reduces the negative environmental impact and proposes economically efficient transportation solutions. Compared to trucks, freight trains emit only a quarter of carbon dioxide per tonne-kilometer and require significantly less energy to move freight (Strocko et al., 2014; Verband der Bahnindustrie in Deutschland (VDB) e.V., 2012). Therefore, one of the key goals of the EU's transport strategy until 2050 is to increase the share of freight transport performed by rail (EU Commission, 2011). The strategic focus of U.S. Department of Transportation is to strengthen the linkage between rail and other modes of transportation (U.S. Department of Transportation, 2013).

Until recently, firms mainly decided for transportation by rail when they had enough freight to set up a train from A to B. New innovative concepts have been designed to make rail transportation attractive also in case of geographically distributed customers and smaller freights. According to the "last mile concept", the most expensive "first miles" from the distributed customer locations to the rail yard and "last miles" from the rail yard to the distributed final destinations have to be performed by trucks. The intermediate distance will be covered by train. Container transportation decreases the handling costs significantly. Containers can be directly transferred from a train to a truck or between trains without time-consuming shunting of railcars. Lower handling costs and times enable bundling policies, e.g. hub-and-spoke, which promote rail transportation to low-to-medium-volume freights.

However, albeit significant political effort to promote the freight rail transportation, its share is decreasing in the EU from 15% in 1980 to 10% in 2010 (EU Commission, 2007), and remains about the same in Germany at 10% until 2030 (BMVI, 2014). In fact, rail logistics fails any competitive advantage. Notably, the actual average speed of freight trains is estimated to be 10-18 km/h (Kille and Schmidt, 2008; Verband der Automobilindustrie, 2006) and only 53% of freight trains reach their destination with less than a 30 minutes delay (EU Commission, 2007), mainly because of long processing and waiting times at shunting and transshipment yards. It has been widely recognized that even small improvements in yard operations, resulting in the decrease of train dwell time, will lead to large financial savings.

In this paper, we consider operations at mixed container transshipment yards, where both *rail-road* container moves (between trains and trucks) and *rail-rail* container moves (between trains) are present. Most prominent transshipment yards in this category are *hub* or *gateway* yards. Gateway yards connect domestic trains, on one side, and international trains that carry dedicated shipments to the next hub, on the other side (see Figure 2.1, DIOMIS, 2007). Inbound trains end and outbound trains originate at gateway yards. Note that international trains travel between their start and end destinations without interim stops. Although rare in US, gateway yards are common in Europe (DIOMIS, 2009), e.g. the transshipment yards Duisburg Ruhrport Hafen, Köln Eifeltor, Hamburg-

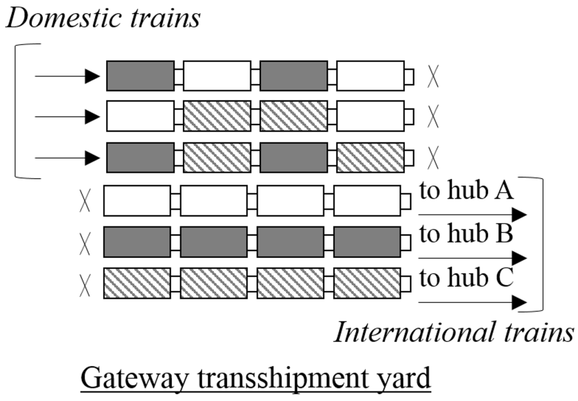


Figure 2.1: Illustration of the movement of trains at a gateway transshipment yard

Billwerder or München-Riem in Germany, to name only a few (see, e.g., Gaidzik et al., 2012). At gateway yards, gantry cranes unload and load trains by exchanging containers between domestic and international trains, trains and storage (*rail-storage* moves) as well as between trains and customer trucks. As a rule, rail-road transshipments comprise the majority of transshipment at gateway yards (cf. DUSS, 2010; Gaidzik et al., 2012). Note that at gateways usually all inbound and outbound containers are exchanged and specific drop-off container positions on the trains therefore do not exists. The flexible container placement at international trains favors short rail-rail moves at gateways in order to keep the cycle time (makespan), i.e., the time of transshipment, low. Indeed, situations where cranes move a container once across the yard are rare and often can be eliminated by improving planning routines of container assignment to railcars of the trains (see also Section 2.2).

A typical transshipment yard, including gateway yards, consists of multiple tracks, a driving lane and a parking lane for trucks, as well as of a storage area, see Figure 2.2 (cf. Ballis and Golias, 2002; Boysen et al., 2013). According to conventional policies, trains are served in bundles, i.e. they arrive and depart largely simultaneously (see Bostel and Dejax, 1998; Boysen et al., 2011; Rotter, 2004). Multiple gantry cranes typically share a common track for horizontal movements that have to be coordinated in order to avoid collisions. A common strategy in real-world transshipment yards is to define operational areas for each crane (Boysen and Fliedner, 2010; Kellner et al., 2012). This strategy is organizationally simple and has low costs of management. Especially crane operators find this strategy most acceptable, because they have their own area of responsibility. The subject of this paper is *assigning and sequencing container moves to cranes.* We call the resulting problem *Static Crane Scheduling Problem* (SCSP) to distinguish it from the dynamic case, where cranes are not restricted to operate within their defined operational areas.

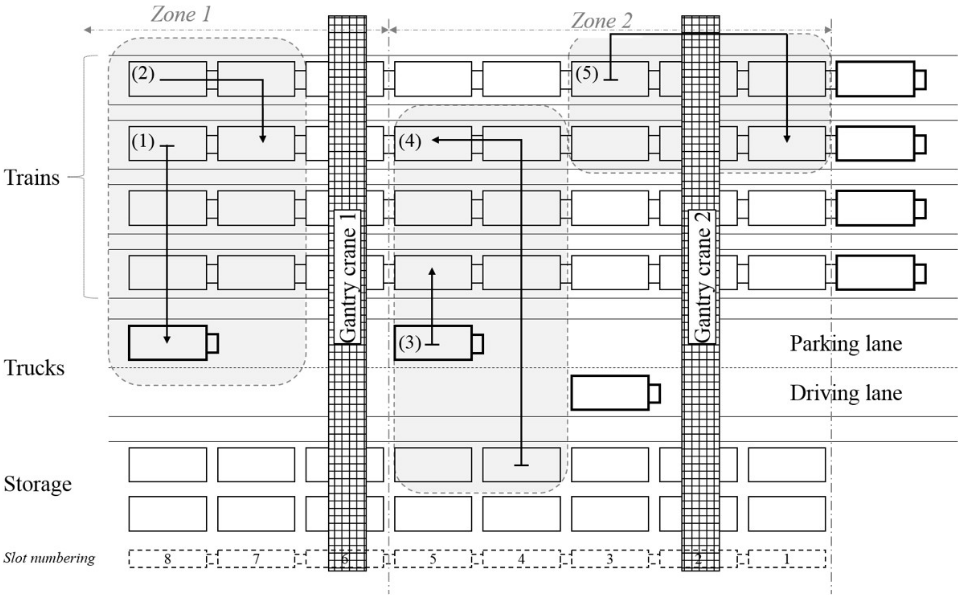


Figure 2.2: Illustration of a mixed container transshipment yard.
Spatially disjoint sets of jobs (see Section 2.2) are marked with grey rectangles

We refer to Boysen et al. (2013) for a detailed description of sequencing and scheduling problems at transshipment yards. The computational intractability has motivated Boysen et al. (2011) and Boysen and Fliedner (2010) to handle assignment and sequencing of crane moves as separate, hierarchically connected planning problems. A dynamic programming approach is proposed in Boysen and Fliedner (2010) for assigning container moves to cranes but neither their sequencing nor setup times or time windows have been included. Priority rules, metaheuristic and local search solution approaches for assignment or sequencing of crane moves are proposed in Kozan (1997), Souffriau et al. (2009), Montemanni et al. (2010) and Froyland et al. (2008). Alicke (2005) contains a constraint satisfaction based approach to find a feasible schedule of crane moves for a rail-rail transshipment yard, where trains have different arrival and departure times.

To the best of our knowledge, we are the first to propose an exact solution procedure for the *simultaneous* assignment and sequencing of crane moves at rail transshipment yards. We propose an exact as well as a heuristic solution approaches and we illustrate the expected benefits of the simultaneous planning by our computational experiments. We also investigate a realistic case of time windows for each container move. Time windows emerge, for example, as a service promise for customer vehicles (cf. Section 4.2).

A further contribution of our paper is a new decomposition approach called *two-way bounded dynamic programming procedure* (TBDP). TBDP enables to get sharp lower and upper bounds early in the solution process, as well as identifies critical subproblems within the decomposition that have to be handled first.

We proceed as follows. Section 2.2 provides a problem description. The TBDP framework is presented in Section 2.3. In Sections 2.4, we describe upper and lower bounds used in our solution procedure. Afterwards, we outline a heuristic and an exact solution approach in Section 2.5. Section 2.6 presents the results of our computational experiments. Finally, Section 2.7 concludes the paper and suggests directions for future research.

2.2 The static crane scheduling problem

In the static crane scheduling problem (SCSP), container moves are assigned to R gantry cranes and sequenced for each crane, whereby each crane operates within its defined operational zone.

Note that within the planning framework (see Boysen et al., 2013), assignment and sequencing of container moves are downstream planning tasks. So that container positions on trains and in the storage area, as well as train-to-track assignments have to be performed beforehand.

Table 2.1 summarizes parameters of the SCSP. We refer to an operation of picking up a container and bringing it to its final position as to a *jo*b $j \in J$. The time a gantry crane needs to move from the *final* container position of the current job i to the *final* container position of the next job j is called *setup time* s_{ij}. Observe, it is the minimum time needed to get from the drop-off position of job i's container to the pickup position of the container that defines job j added by the time to fix the crane's grip hooks at this container and lift and move it to its drop-off position. Thus the setup s_{ij} includes the processing time of job j. Notice that the triangular inequality is valid: $s_{ij} \leq s_{ik} + s_{kj}$. We differentiate among rail-road, rail-storage and rail-rail jobs.

Table 2.1: Notations of SCSP

J	Set of jobs
s_{ij}	Setup and transportation time between the final position of job i to the final position of job j
τ_j^d	Deadline of job j
τ_j^e	Release time of job j (the earliest time, when the container may be brought to its final position)
E	Set of precedence relations between jobs
P_j	Set of jobs preceding job j: $P_j = \{i \vert (i,j) \in E\}$
F_j	Set of jobs following job j: $F_j = \{i \vert (j,i) \in E\}$
$SK = \{1, ..., K\}$	Set of spatially disjoint sets
$a_{kj} \in \{1, 0\}$	Parameter, showing whether job j belongs to the spatially disjoint set k or not
$V = \{1, ..., R\}$	Set of cranes
D	Set of dummy jobs

Note that for rail-road jobs trucks do not block each other and can always park at the *most convenient* slot of the parking lane to minimize the processing time of the operating crane. Indeed, trucks arrive from the parking area outside the transshipment yard and leave the parking lane immediately after a gantry crane has processed the corresponding job.

Time windows arise as a service promise to customer vehicles. One of the highest priorities of railway companies is to serve customer vehicles in a short timespan after their arrival. Modern surveillance systems (e.g. BLU, which is implemented, for example, at yards Köln Eifeltor, Hamburg-Billwerde and Leipzig-Wahren) enable incorporation of service time windows into planning approaches. They register customer vehicles far in advance of their arrival, carefully predict their arrival times and route them to their position of service. Further reason for using time windows is the necessity to transship containers by trucks between several transshipment areas (e.g., a modular terminal, see Rotter, 2004, for details). Therefore job $j \in J$ has a release time, or earliest start time τ_j^e and a deadline τ_j^d.

Cranes $r \in V = \{1, ..., R\}$ operate within defined zones. We divide the yard into several crane handling zones, such that each container move remains *within* its zone.

Thus having dedicated zones is reasonable if rail-rail moves are short which is typical for gateways. It is additionally achieved by exceptionally introducing rail-storage moves and by selecting suitable parking positions of trains (e.g. Kellner et al., 2012), separating trains into bundles (e.g. Boysen et al., 2011; Nossack and Pesch, 2014) and by targeted selection of container positions at trains (e.g. Bostel and Dejax, 1998; Bruns and Knust, 2012; Cichenski et al., 2017; Corry and Kozan, 2006, 2008). Note that especially at gateway yards, container positions are flexible at international trains commuting between the hubs, because all the containers have the same destinations. Additionally, types and sizes of containers in international traffic, e.g. at gateway yards, are rather uniform compared to the domestic traffic.

In a preprocessing, the set of jobs can be separated into *spatially disjoint* sets $k \in SK = \{1, ..., K\}$. A parameter $a_{kj \in \{0,1\}}$ shows whether job j belongs to spatially disjoint set k ($a_{kj} = 1$) or not ($a_{kj} = 0$). In order to avoid container transshipments between zones, jobs from the same spatially disjoint set have to be assigned to the same zone. Figure 2.2 contains three spatially disjoint sets, $K = 3$, and two cranes, $R = 2$. Jobs 1 and 2 belong to the first spatially disjoint set, jobs 3 and 4 to the second one and job 5 belongs to the third one. Therefore, if crane one performs jobs 1 to 3, it has to perform job 4 as well.

Notice that precedence relations may emerge between jobs i and j if the respective container has to be picked from or dropped to the same railcar at a train. Precedence relations may arise only between jobs i and j that belong to the *same* spatially disjoint set, i.e., if $\sum_{k=1}^{K} a_{ki}k = \sum_{k=1}^{K} a_{kj}k$. The set of precedence relations E contains *all* pairs of jobs $(i,j) \in E$, where job i has to be performed before job j.

The objective is to minimize makespan, i.e. the time to serve a bundle of trains. Such an objective is of high importance in practice. Due to high competitive pressure, rail companies struggle for shorter train processing times at transshipment yards. Shorter train processing times at rail yards enable to be more flexible in negotiations with a network provider about renting travelling slots and to decrease travel times of customer containers in the mid- and long-run. In the short-term, this objective also allows to find a feasible solution, which meets the scheduled train departure times.

Binary decision variables x_{ij} indicate, whether job i is performed before job j by the same crane ($x_{ij} = 1$) or not ($x_{ij} = 0$). Decision variable q_k defines the crane, which performs jobs from a spatially disjoint set k. A starting time for each job j is determined by variable τ_j^s.

$D = \{0, n+1, n+2, ..., n+R\}$ are dummy jobs with release times of zero and no processing times. $\{0\}$ is a dummy source job with $\tau_0^d = 0$, job $\{n+r\}$ is a dummy sink job for crane $r \in V$. Then the SCSP can be modelled as follows:

$$\min \quad A \tag{2.1}$$

$$s.t. \quad \sum_{j \in \{1,...,n+R\} \setminus i} x_{ij} = 1 \qquad \forall\, i \in J, \tag{2.2}$$

$$\sum_{i \in J \cup \{0\} \setminus j} x_{ij} = 1 \qquad \forall\, j \in J \cup D \setminus \{0\} \tag{2.3}$$

$$\sum_{j=1}^{n+R} x_{0j} \leq R \tag{2.4}$$

$$\tau_i^s + s_{ij} \leq \tau_j^s + M(1 - x_{ij}) \qquad \forall\, i \in J \cup \{0\}, \forall\, j \in J \cup D \setminus \{0\}, i \neq j \tag{2.5}$$

$$\tau_j^e \leq \tau_j^s \leq \tau_j^d \qquad \forall\, j \in J \cup \{0\} \tag{2.6}$$

$$\tau_i^s + s_{ij} \leq \tau_j^s \qquad \forall\, (i,j) \in E \tag{2.7}$$

$$0 \leq \sum_{k=1}^{K} (q_k \cdot a_{kj} - q_k \cdot a_{ki}) \leq (R-1) \cdot (1 - x_{ij} - x_{ji}) \qquad \forall\, i,j \in J,\ such\, that \sum_{k=1}^{K} a_{ki}k \leq \sum_{k=1}^{K} a_{kj}k \tag{2.8}$$

$$r \cdot x_{i,n+r} \leq \sum_{k=1}^{K} q_k \cdot a_{ki} \leq r + R \cdot (1 - x_{i,n+r}) \qquad \forall\, r \in V, \forall\, i \in J \tag{2.9}$$

$$\tau_j^s \leq A \qquad \forall\, j \in D \setminus \{0\} \tag{2.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i,j \in J \cup D \tag{2.11}$$

$$q_k \in \{1,...,R\} \qquad \forall\, k \in \{1,...,K\} \tag{2.12}$$

Objective (2.1) is to minimize the makespan A. Constraints (2.2) - (2.4) require that jobs are arranged in R chains, that end with sink dummy jobs and start with source dummy job $\{0\}$. Each job $j \in J$ has exactly one predecessor and exactly one successor.

The time for the required setup has to be observed before starting the next job (constraints (2.5)). Note, that constraints (2.5) also eliminate possible cycles in a sequence of jobs for each crane. Constraints (2.6) ensure that jobs are performed within their time windows. Further, precedence relations have to be taken into account (constraints (2.7)).

Constraints (2.8) to (2.9) assign jobs to cranes. Thereby, constraints (2.8) require that a "chain of jobs" is assigned to the same crane. We assume that spatially disjoint sets are increasingly numbered from 1 to K. In the same way, cranes are numbered increasingly from 1 to R. Therefore we may assume that $q_{k'} \leq q_k$ for $k' \leq k, k, k' \in SK$. (2.9) require each crane to perform its dummy sink job.

Constraints (2.10) calculate the makespan of the bundle of trains. Binary constraints are imposed in (2.11) and values of q_k are restricted in (2.12).

Table 2.2: Illustrative example of a SCSP problem instance with two cranes and no precedence relations

Job $j =$	1	2	3	4	5
τ_j^e	1	3	4	5	1
τ_j^d	5	20	10	20	5
a_{1j}	1	1	0	0	0
a_{2j}	0	0	1	1	0
a_{3j}	0	0	0	0	1

	setup time s_{ij}				
	j				
i	1	2	3	4	5
0	1	2	1	2	1
1	-	5	3	4	7
2	3	-	4	4	6
3	5	7	-	5	5
4	4	4	3	-	4
5	7	6	5	4	-

In our illustrative example of Table 2.2, there are two cranes $R = 2$, five jobs and we assume three spatially disjoint sets $K = 3$ containing job sets $\{1,2\}, \{3,4\}$ and $\{5\}$. A solution might look like that: crane 1 performs first job 1 and then job 2 and crane 2 performs first job 5, then 4 and last job 3. The makespan equals to 8. Starting times τ_j^s are 1, 6, 8, 5 and 1 for jobs 1 to 5, respectively. Since crane 2 performs jobs from spatially disjoint subsets two and three, $q_1 = 1, q_2 = 2$ and $q_3 = 2$. Our model introduces three dummy jobs $D = \{0, 6, 7\}$. Further, $x_{01} = x_{05} = x_{12} = x_{26} = x_{54} = x_{43} = x_{37} = 1$ and other variables x_{ij} are equal to zero.

To solve the SCSP on the set of jobs J, we apply a hierarchical decomposition that splits the problem into a *yard partitioning problem* and subsequent *job scheduling problems* $\Pi(k', k)$ *for single cranes* each on the set of jobs $J' \subseteq J$. In the yard partitioning problem, we limit the operating area for each crane by assigning sets of jobs $k \in SK$ to crane zones. Based on the resulting yard partition, suppose a crane has to perform a set of jobs J' consisting of the disjoint sets $\{k', ..., k\}$. Then the single crane problem $\Pi(k', k)$ is to find a sequence of the jobs $j \in J'$, so that the makespan is minimized, i.e., $\Pi(k', k)$ is equivalent to the SCSP with $R = 1$ and $J = J' = \{j | \sum_{f=k'}^{k} a_{fj} = 1\}$.

In the next Section 2.3, we introduce an innovative concept for the overall decomposition which is two-way dynamic programming (TBDP). This framework allows us to economize on computational time by examining only promising yard partitions. To indicate promising yard partitions, we rely on upper and lower bounds for job scheduling problems for single cranes $\Pi(k', k)$. First we describe the general framework and then we sketch upper bounds, lower bounds as well as a branch and bound algorithm for single crane scheduling problem $\Pi(k', k)$ in Section 2.4. However, note that Sections 2.3 and 2.4 are written independently from each other, so that a reader may decide to start first with Section 2.4. Finally, in Section 2.5, we develop a heuristic and an exact algorithm for the SCSP based on the two-way dynamic programming framework.

2.3 Two-way bounded dynamic programming framework

The single crane scheduling problem $\Pi(k', k)$ (see Section 2.2) is a generalization of the asymmetric travelling salesman problem and NP-hard in the strong sense. Therefore, we should avoid solving too many of the problems $\Pi(k', k)$ with dynamic programming.

Suppose we know the defined crane zones, i.e. we are given a yard partition. Let us further assume an oracle gave us a bottleneck crane zone, i.e. the one with the largest makespan in the given yard partition. Then, we would focus on finding a sequence of jobs with a minimal makespan only for this crane zone, i.e. we would solve $\Pi(k', k)$ to optimality only for this zone. For all remaining zones, feasible job sequences of sufficient solution quality would suffice. Hence, a good solution procedure implicitly generates all possible yard partitions. Lower and upper bounds to partial solutions identify promising yard partitions and so-called bottleneck zones. Bottleneck zones are crane zones that are crucial for the calculation of a *global upper bound (GUB)* or a *global lower bound (GLB)* to the optimal objective value over all partitions. An improved sequence of jobs for the bottleneck zones may narrow the gap between the global lower and upper bound.

The procedure may continue until either a satisfactory solution has been achieved or the solution has been proved to be optimal because the two global bounds are identical. Hence, a good solution procedure must identify promising yard partitions early as well as bottleneck zones that are pivotal for the value of the objective function. The two-way dynamic programming provides such navigation.

We introduce and motivate the concept of the two-way dynamic programming (TBDP) (Section 2.3.1). Then we explain TBDP on the example of the static crane scheduling problem SCSP (Section 2.3.2). Finally in Section 2.3.3, we provide a summary of TBDP.

2.3.1 Outline of the two-way bounded dynamic programming framework

TBDP is based on dynamic programming. In dynamic programming, the overall optimization problem (w.l.o.g., we refer to minimization problems) is decomposed into computationally easier subproblems, or *states*, that are recursively nested inside the initial optimization problem (Bellman, 1954). The subproblems are evaluated (solved to optimality) by the *value function.* All these nested subproblems, as well as relations between them are summarized by a state graph. States represent nodes of the *state graph* and arcs are referred to as *transition arcs.*

Two-way bounded dynamic programming (TBDP) is developed for cases, when the state graph is of moderate size, but it takes a lot of time to calculate the value function for each state exactly. Our framework combines the strengths of the approximate dynamic programming of Bertsimas and Demir (2002) (see also Powell, 2010) as well as of the relaxed dynamic programming procedure of Christofides et al. (1981). In the former, values of the value function are calculated approximately in order to receive a good GUB for the problem instance. The later relaxes states and thus bounds the values of the value function from below, to come up with a GLB.

Apart from GLB and GUB, TBDP provides further valuable information. On the one hand, we can easily track the *bottleneck points* of the value function, which are pivotal for the values of the current GLB and GUB. Improving bounds of these few point values of the value function will directly improve the GLB or the GUB. On the other hand, we can identify non-influential points of the value function, which are sufficiently approximated/bounded, i.e. tighter bounds on these point values will neither affect the GLB nor the GUB. We ignore these points, i.e. *prune* the corresponding states and transition arcs from the state graph.

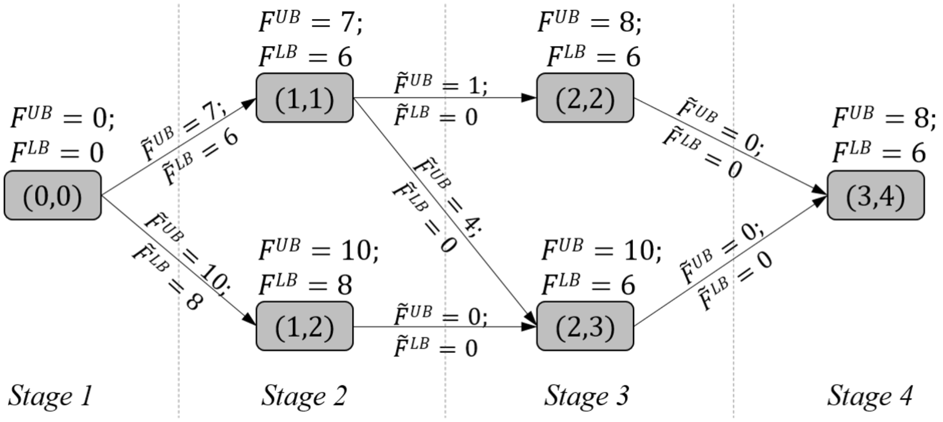


Figure 2.3: Illustrative example based on Table 3: state graph for dynamic bottleneck crane procedure

2.3.2 Illustrating TBDP for the static crane scheduling problem

Let the sets of jobs $k \in SK = \{1, ..., K\}$, as well as cranes $r \in V = \{1, ..., R\}$ be numbered in increasing order from left to right according to their position. In the state graph of TBDP, states are partial yard partitions, i.e. yard partitions with $r \leq R$ crane zones. A transition arc from state a to state b means, that in partial yard partition b the crane zone is added to a that is adjacent to the last (most right) zone in partial yard partition a. TBDP has $R+1$ stages, notice that at the initial state at stage 1 no crane is assigned. At each subsequent stage, the next right unassigned crane will be added to the current yard partition. The different states of a stage differ in their job sets. Thus, the nodes of the state graph (see Figure 2.3) are partial yard partitions (r, k) for r cranes (or zones) and for jobs from spatially disjoint sets $\{1, 2, ..., k\}$. Transition arcs $((r-1, k'), (r, k))$ connect nodes, or states, of neighboring stages and represent valid transitions between states. Note that w. l. o. g., our formulation of the SCSP assumes $s_{0j} \leq \tau_j^r, \forall j \in J$. We demand that $k' < k$ because we won't miss an optimal solution by requiring each crane zone to contain at least one spatially disjoint set of jobs.

We use two labels for each state which correspond to an upper bound $F^{UB}(\cdot)$ and a lower bound $F^{LB}(\cdot)$ of the value function. Let $LB(\Pi(k', k))$ be a lower bound and $UB(\Pi(k', k))$ be an upper bound of the corresponding single crane scheduling problem $\Pi(k', k)$, which can be computed as described in Section 2.4. Then we assign two labels to *arc* $((r-1, k'), (r, k))$ that show the increase in the upper (or lower) bound of the makespan of the current partial yard partitioning, if we add the next crane that serves jobs from sets $\{k'+1, ..., k\}$:

$$\tilde{F}^{UB}(((r-1, k'), (r, k))) = \max\{0, UB(\Pi(k'+1, k)) - F^{UB}(r-1, k')\} \tag{2.13}$$

$$\tilde{F}^{LB}(((r-1,k'),(r,k))) = \max\{0, LB(\Pi(k'+1,k)) - F^{LB}(r-1,k')\} \tag{2.14}$$

$$\tilde{F}^{LB}(((r',k'),(r,k))) = \infty \qquad \forall\, r',r \in R | r' \neq r-1, \forall\, k',k \in SK \tag{2.15}$$

$$\tilde{F}^{UB}(((r',k'),(r,k))) = \infty \qquad \forall\, r',r \in R | r' \neq r-1, \forall\, k',k \in SK \tag{2.16}$$

$$\tilde{F}^{LB}(((r',k'),(r,k))) = \infty \qquad \forall\, r',r \in R, \forall\, k',k \in SK | k' \geq k \tag{2.17}$$

$$\tilde{F}^{UB}(((r',k'),(r,k))) = \infty \qquad \forall\, r',r \in R, \forall\, k',k \in SK | k' \geq k \tag{2.18}$$

Node labels refer to the length of the shortest path in the state graph from state $(0,0)$ and are computed by the recurrence equations:

$$F^{UB}(r,k) = \min_{k'<k}\{F^{UB}(r-1,k') + \tilde{F}^{UB}(((r-1,k'),(r,k)))\} \tag{2.19}$$

$$F^{LB}(r,k) = \min_{k'<k}\{F^{LB}(r-1,k') + \tilde{F}^{LB}(((r-1,k'),(r,k)))\} \tag{2.20}$$

The last stage contains only one node (R,K). $F^{UB}(R,K)$ is the global upper bound (GUB) and $F^{LB}(R,K)$ is the global lower bound (GLB) for the SCSP.

We use a forward dynamic programming algorithm and initialize the recursion by:

$$F^{UB}(0,0) = F^{LB}(0,0) \tag{2.21}$$

$$F^{UB}(0,k) = F^{LB}(0,k) = \infty \qquad \forall\, k \in SK \tag{2.22}$$

$$\begin{aligned} &\tilde{F}^{UB}(((0,0),(1,k))) = UB(\Pi(1,k)) \\ &\tilde{F}^{LB}(((0,0),(1,k))) = LB(\Pi(1,k)) \qquad \forall\, k \in SK \end{aligned} \tag{2.23}$$

Example. Consider a situation where we have three cranes, $R = 3$, and four spatially disjoint sets, $K = 4$, with upper and lower bounds for *single crane* scheduling problems $\Pi(k',k)$ assumed as provided by Table 2.3. Note that we require that each crane performs at least one job. In this way, we further reduce the problem size without missing an optimal solution. The state graph with $R+1 = 4$ stages of TBDP is provided in Figure 2.3. We see, for example, that labels for arc $((1,1),(2,2))$ are $\tilde{F}^{UB}(((1,1),(2,2))) = \max\{0, 8-7\} = 1$ and $\tilde{F}^{LB}(((1,1),(2,2))) = \max\{0, 5-6\} = 0$. Labels of node $(2,3)$ are $F^{UB}(2,3) = \min\{F^{UB}(1,1) + \tilde{F}^{UB}(((1,1),(2,3))), F^{UB}(1,2) + \tilde{F}^{UB}(((1,2),(2,3)))\} = \min\{7+4, 10+0\} = 10$ and $F^{LB}(2,3) = \min\{6+0, 8+0\} = 6$. In our example, $GUB = F^{UB}(3,4) = 8$ and $GLB = F^{LB}(3,4) = 6$.

The computational complexity of TBDP for SCSP heavily depends on the complexity

Table 2.3: Example for four spatially disjoint sets

	$UB(\Pi(k',k))\backslash LB(\Pi(k',k))$			
k' \ k	1	2	3	4
1	7\6	~~10\8~~		*
2		8\5	11\6	
3		*	~~5\2~~	8\6
4				8\1

of the algorithms for upper and lower bounds of $\Pi(k', k)$, which have to be run repeatedly. But the number of arcs in the state graph is rather low and is bounded by $O(K^2 \cdot R)$.

With TBDP, it is easy to trace bottleneck yard partitions and bottleneck single crane scheduling problems. We call a yard partition *ub-bottleneck (lb-bottleneck)*, if the best known makespan (largest lower bound on the makespan) of this yard partition equals to the current GUB (GLB). A zone, containing the corresponding $\Pi(k', k)$, is *ub-bottleneck* if it is a part of an ub-bottleneck yard partition *and* if its best known makespan equals to the current GUB. In a similar way, we define an *lb-bottleneck* zone, the corresponding $\Pi(k', k)$ is called lb-bottleneck as well.

At each stage, we can store some best found (with respect to the upper bound) partial yard partitions for each state and thus receive several ub-bottleneck yard partitions in the final state (R, K). Note that the upper bound for *each* ub-bottleneck zone in *at least one* ub-bottleneck yard partition has to be improved in order to improve (decrease) the GUB.

Similarly, we can mark zones that are lb-bottleneck in *all* the lb-bottleneck yard partitions. Note that to increase the GLB, the lower bound for *at least one* lb-bottleneck zone in *each* of the lb-bottleneck yard partitions has to be improved.

Further, we can prune $\Pi(k', k)$ if the makespan exceeds the global upper bound ($GUB \leq LB(\Pi(k', k))$). Similarly, we need not solve $\Pi(k', k)$ optimally if $UB(\Pi(k', k)) \leq GLB$; i.e. we mark them as "sufficiently approximated".

Given the moderate size of the state graph, we perform the *bounding iteration* (i.e. determine shortest paths with respect to the upper and lower bound values in the state graph again, see Figure 2.4) each time, when GLB and GUB get updated. So that TBDP navigates to the currently ub- and lb-bottleneck zones.

Example ***(continuation)***. In the example in Table 2.3 and Figure 2.3, we can prune subproblems $\Pi(1, 2)$ and $\Pi(3, 3)$. Indeed, $UB(\Pi(1, 2)) > LB(\Pi(1, 2)) \geq GUB = 8$ and $UB(\Pi(3, 3)) < GLB = 6$. Further, there is only one ub-bottleneck zone partitioning $\{(1, 1), (2, 2), (3, 4)\}$; it contains two ub-bottleneck zones $(2, 2)$ and $(3, 4)$. To decrease GUB, we have to decrease both $UB(\Pi(2, 2))$ and $UB(\Pi(3, 4))$. On the other hand, there are three lb-bottleneck zones: $(1, 1)$, $(3, 4)$ and $(2, 3)$. However to increase GLB, it suffices

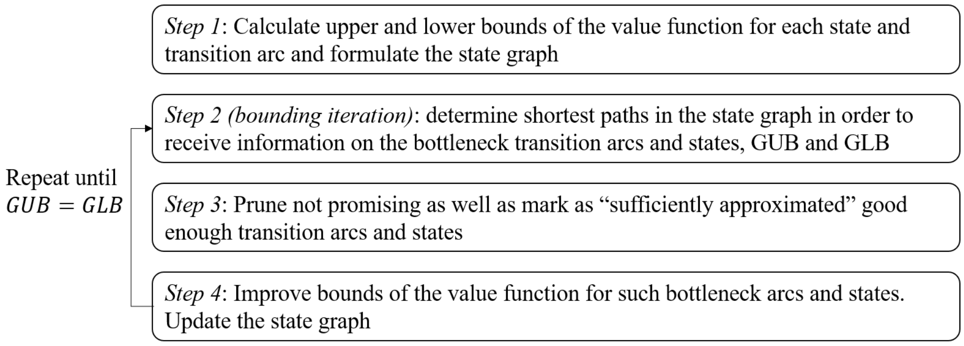

Step 1: Calculate upper and lower bounds of the value function for each state and transition arc and formulate the state graph
Step 2 (bounding iteration): determine shortest paths in the state graph in order to receive information on the bottleneck transition arcs and states, GUB and GLB
Step 3: Prune not promising as well as mark as "sufficiently approximated" good enough transition arcs and states
Step 4: Improve bounds of the value function for such bottleneck arcs and states. Update the state graph
Repeat until $GUB = GLB$

Figure 2.4: Outline of the two-way bounded dynamic programming (TBDP) framework

to improve the lower bound *only* of $\Pi(1,1)$, because zone $(1,1)$ is contained in all the lb-bottleneck yard partitions (these are $\{(1,1),(2,2),(3,4)\}$ and $\{(1,1),(2,3),(4,4)\}$).

2.3.3 Summary of the TBDP framework

In TBDP (see Figure 2.4) we calculate an upper and a lower bound for the value function of each state with generally more than one crane and for each transition arc. Thereby we rely on the objective value bounds for the single crane problems. Afterwards, we prune transition arcs (and states) in the state graph, for which estimations of the value function do not have to be improved. We also identify bottleneck transition arcs (and states), for which the estimation of the value function is currently pivotal for the GUB or GLB. We improve bounds of the value functions for such bottleneck arcs and states. If the GUB or GLB improve, we update the state graph and re-run the *bounding iteration* in order to receive actual information on the bottleneck transition arcs and states. We repeat the procedure, until a solution of desired quality has been found.

Depending on the stopping criterion, the TBDP framework may serve as an effective heuristic or as an exact solution procedure.

2.4 Bounds and an exact algorithm for the single crane scheduling problem

Bounds for the single crane scheduling problem $\Pi(k',k)$ play an important role in our dynamic programming approach introduced in Section 2.3. Therefore, we outline lower bounds (Section 2.4.1) and an upper bound (Section 2.4.2) for the single crane scheduling problem. Afterwards we describe a branch and bound algorithm for the job set J' defined by $\Pi(k',k)$ (Section 2.4.3).

2.4.1 Lower bounds for single crane scheduling problem

In this section, we additionally use p_j to define the processing time of job j, i.e., it is the time needed when the crane's grip hooks are fixed to pick the container until it is dropped at the final position. Setup times and release times without including the processing time are defined as $\vartheta_{ij} = s_{ij} - p_j$ and $e_j = \tau_j^e - p_j$, respectively.

LB1. To compute LB1, we relax the single crane scheduling problem and build a single machine scheduling problem $1/e_i/C_{max}$. The computational complexity of LB1 is $O(|J'|^2)$. Details on LB1 can be found in Appendix A.

LB2. LB2 is based on the relaxation to the linear assignment problem (LAP, Burkard et al., 2009). In the LAP we assign to each job i an immediately following job j, $i \in \{0\} \cup J'$, $j \in J' \cup \{d\}$, where $\{d\}$ is a dummy sink job. The assignment costs c_{ij} of i to j are equal to s_{ij}, $c_{ii} = M$ and $c_{0i} = \max\{\tau_i^e, s_{0i}\}$. M is a sufficiently large number, for example, $M = \max_{i \in \{0\} \cup J'}\{\tau_i^e\} + \sum_{i,j \in \{0\} \cup J' \cup \{d\}} s_{ij}$. We modify some assignment costs in order to:

- Require that $\{0\}$ is at the "beginning" of the resulting assignment: $c_{0d} = M$, $i \in J'$.
- Take into account precedence relations: $c_{ij} = M$ if $(j, i) \in E$.

LB2 can be computed, for example, by the Hungarian algorithm in $O(|J'|^3)$ steps.

LB3. In LB3, we apply a relaxation to a vehicle routing problem with time windows (*VRP-TW,* Desrosiers et al., 1984) and compute its lower bound by a column generation approach.

In VRP-TW, customers can be served by an unlimited number of vehicles, where each vehicle has to start at the source depot and to finish at the sink depot. Customers are visited within the specified time windows. We interpret jobs $j \in J'$ as customers, dummy source job $\{0\}$ as the source depot and dummy sink job $\{d\}$ as the sink depot. We set the arc costs in VRP-TW equal to s_{ij}. Note that the objective function of VRP-TW is not the makespan, but total costs over all vehicles, i.e. possible waiting times of vehicles are neglected.

In the column generation approach, we rely on the model formulation of Desrosiers et al. (1984). We use the algorithm of Cunha and Swait (2000) for the restricted shortest path problem to generate the routes of single vehicles.

2.4.2 Upper bound for the single crane scheduling problem

To compute an upper bound, we start with *preprocessing steps.* Afterwards, we use construction heuristics: a *multistart feasible nearest neighbor heuristic* as well as *priority* *rules.* For each received feasible sequence of jobs, we apply *local search procedures* of node reinsertion and of two node exchange.

Preprocessing. We apply preprocessing steps, described by Ascheuer et al. (2001), to tighten time windows of jobs and construct new precedence relations between the jobs.

Release rule takes into account that job $j \in J'$ cannot start earlier than the earliest possible arrival time of the crane from a preceding job:

$\tau_j^e = \max\{\tau_j^e, \max_{i \in P_j}\{\tau_i^e + s_{ij}\}\}$.

Deadline rule takes into account that job $j \in J'$ cannot finish later than the latest possible departure time of the crane to a succeeding job:

$\tau_j^d = \min\{\tau_j^d, \min_{i \in F_j}\{\tau_i^d - s_{ji}\}\}$.

The preprocessing steps have computational complexity of $O(|J'|^2)$.

Multiple start feasible nearest neighbor heuristic (MSFNH). Feasible nearest neighbors are determined by taking time windows and precedence relations into account (cf. Ascheuer et al., 2001). The computational complexity of MSFNH is $O(|J'|^3)$.

Priority rules. We apply *composite* priority rules, as described by Otto and Otto (2014a). In this construction heuristic, jobs $j \in J'$ are assigned one after another to build a sequence. At each step, a job with the highest priority value (*priovalue*) is selected.

Elementary priovalues for job j are calculated based on its deadline (earliest deadline priovalue EDD_j), on its release time (earliest release time priovalue ERT_j), on its setup time from the currently last job in the sequence (smallest setup priovalue S_j) or on the size of its time window (time window priovalue TW_j).

In composite priority rules, the *final priovalue* of job j is received by a weighted sum of (in our case two) elementary priovalues. Composite rule (EDD_j, S_j, w) means, for example, that the final priovalue of job $j \in J'$ is computed as $[w \cdot EDD_j + (1-w) \cdot S_j]$. Thereby priovalues are *scaled* to make them comparable to each other. We apply composite priority rules (EDD_j, S_j, w), (ERT_j, S_j, w) and (TW_j, S_j, w) with weights $w \in \{0, 0.1, 0.2, ..., 1\}$, which makes $3 \cdot 10 + 1 = 31$ composite priority rules in total (since the composite rules are identical for $w = 0$).

Let job $l \in J'$ be the last job in the currently built sequence of jobs and τ_l^s is the starting time of this job l. Then, the scaled priovalues are computed as follows:

$$EDD_j = -\frac{\tau_j^d}{\sum_{f \in J'} \max_{i \in J' | (i,f) \notin E}\{s_{fi}\}};$$

$$ERT_j = -\frac{\tau_j^e}{\max_{i \in J'}\{\tau_i^e\}};$$

$$TW_j = -\frac{\tau_j^d - \tau_j^e}{\sum_{f\in J'} \max_{i\in J'|(i,f)\notin E}\{s_{fi}\}};$$
$$S_j = -\frac{\max\{\tau_j^e, \tau_l^s + s_{lj}\} - \tau_l^s}{\max_{f,i\in J'|(i,f)\notin E\{s_{fi}\}}}.$$

Example. Let us apply composite priority rule $(EDD_j, S_j, 0.5)$ on a single crane scheduling instance with four jobs $J' = \{1, 2, 3, 4\}$ and parameters as provided in Table 2.2. We assume that there are no precedence relations. In the first assignment step, we select job 1 because it has the highest final priovalue and fix job 1 to be first in the job sequence (see Table 2.4). We set $\tau_1^s = \max\{\tau_1^e, s_{01}\} = max\{1, 1\} = 1$. In the second assignment step, we select job 3 and set $\tau_3^s = \max\{\tau_3^e, \tau_1^s + s_{13}\} = \max\{4, 1 + 3\} = 4$ and so on. The final sequence is $(1, 3, 4, 2)$ with the makespan of 13.

The computational complexity of a composite priority rule is $O(|J'|^2)$.

Table 2.4: Application of priority rule $(EDD_j, S_j, 0.5)$

	Job	1	2	3	4
	Priovalue EDD_j	-0.25	-1	-0.5	-1
Step 1	Priovalue S_j	-1/7	-3/7	-4/7	-5/7
	Final priovalue	**-0.20**	-0.71	-0.54	-0.86
Step 2	Priovalue S_j	-	-5/7	-3/7	-4/7
	Final priovalue	-	-0.86	**-0.46**	-0.79
Step 3	Priovalue S_j	-	-1	-	-5/7
	Final priovalue	-	-1	-	**-0.86**

Local search. Node-reinsertion and two-node exchange local search procedures are adapted from Ascheuer et al. (2001). The computational complexity of the local search procedure is $O(|J'|^2)$.

2.4.3 Branch and bound algorithm for the single crane scheduling problem

Our overall branch and bound procedure for the single crane scheduling problem $\Pi(k', k)$ is based on a depth-first search. The root node P_0, which is located at *depth* $\delta = 1$ of the branch and bound tree, is the initial single crane scheduling problem $\Pi(k', k)$ on the set of jobs J'. In each node P_r which corresponds to a single crane scheduling problem of a set of jobs J'_{P_r}, branching leads to several subproblems. A *child* node P_l at depth δ' distinguishes from its *parent* node P_r at depth $(\delta' - 1)$ by assigning some job $j \in J'_{P_r}$ to the next position in the sequence. In order to strengthen the preprocessing and lower bounds, we consider in P_l only the scheduling problem for the jobs $J'_{P_l} = J'_{P_r} \setminus \{j\}$ with setup times from the source (job) equal to $s'_{oi} = \tau^s_j + s_{ji}$, $\forall i \in J'_{P_l}$.

In the *root node* P_0, we compute an upper bound $UB(\Pi(k', k))$ of the single crane scheduling problem according to Section 2.4.2. Note that we check the reduction rule (see below) for each generated feasible sequence. We take the maximum of LB1, LB2 and LB3 (Section 2.4.1) as lower bound $LB(\Pi(k', k))$.

In all *other nodes* P_r we run a preprocessing step (Section 2.4.2) and calculate a lower bound $LB(P_r)$ as the maximum of LB1 and LB2. The computational complexity of LB3 forces to limit its use for nodes up to depth $\delta = 2$.

We fathom node P_r if its lower bound exceeds the upper bound, i.e. $LB(P_r) \geq UB(\Pi(k', k))$ or if there is no feasible solution.

Reduction rule. Consider a feasible sequence of jobs where $[f]$ denotes the job at position f and let $\tau^s_{[f]}$ be the starting time for job $[f] \in J'$ in this sequence. If for some $[f] \in J'$, releases of all subsequent jobs are late enough $(\tau^e_{[f']} \geq \tau^s_{[f]} + s_{[f],[f']}, \forall f' > f, [f'] \in J')$, then the subsequence $([1], [2], ..., [f])$ does not have to be improved anymore. Therefore we reduce the original single crane scheduling problem and set $J' := J' \setminus \{[1], [2], ..., [f]\}$.

In the reduction rule, we check each job in a feasible sequence starting from the last one. Its computational complexity is $O(|J'|^2)$.

2.5 Algorithms for the static crane scheduling problem

In this section, we combine solution elements introduced above and develop two solution algorithms for the SCSP, a heuristic (Section 2.5.1) and an exact algorithm (Section 2.5.2). The algorithms are based on the two-way bounded dynamic programming framework (Section 2.3).

2.5.1 The TBDP heuristic for the SCSP

In the TBDP heuristic, we perform only steps 1 and 2 (see Figure 2.4). Thereby single crane scheduling problems $\Pi(k', k)$ are solved heuristically, by computing only an upper bound (see Section 2.4.2).

We can check the quality of the received solution, by computing a global lower bound within the TBDP framework based on lower bounds for single crane problems (Section 2.4.1). In case of insufficient solution quality, the planner can re-solve ub-bottleneck single crane scheduling problems of the stored most promising yard partition(s) with more powerful algorithms, e.g. with a truncated branch and bound algorithm (e.g. Section 2.4.3) or beam search.

2.5.2 TBDP-based exact method for the SCSP (TEMP)

TEMP is an exact solution procedure based on the TBDP framework (cf. Figure 2.4). In TEMP, we run the TBDP procedure until an optimal solution is found and its optimality is proven. TEMP has an *integrated* branch and bound (B&B) algorithm to solve job scheduling problems for single cranes. The B&B algorithm (see Section 2.4.3) in TEMP is applied to improve lower bounds of the lb-bottleneck zones.

In step 1 of TEMP (see Figure 2.4), upper bounds $UB(\Pi(k', k))$ are computed as described in Section 2.4.2 and lower bounds $LB(\Pi(k', k))$ are set to the maximum of LB1 and LB2 (see Section 2.4.1). In step 4, we apply a branch and bound algorithm (see Section 2.4.3) to improve the lower bound of the *lb-bottleneck* zone with the highest priority. We improve lb-bottlenecks rather than ub-bottleneck zones, because the GLBs were less close to the optimal value than the GUBs in our experiments. We stop the B&B, when an optimal solution is found or an improvement of the lower bound is achieved.

We put the highest priority to lb-bottleneck zones, which are a part of *all* the lb-bottleneck crane partitions. Recall that it suffices to improve the lower bound only of one of such zones in order to raise the GLB. Thereafter, priority is given to lb-bottleneck zones (k', k), which contain the *least* number of spatially disjoint sets $[k - k' + 1]$. Indeed, if $k''' \leq k'$ and $k \leq k''$ and we were able to raise $LB(\Pi(k', k))$ so that $LB(\Pi(k', k)) > LB(\Pi(k''', k''))$, then we can raise $LB(\Pi(k''', k''))$ as well.

2.6 Computational Experiments

Yard layout. For our computational experiments we generated instances that closely mimic typical German transshipment yards. A typical terminal segment is about 700 meters long (Boysen et al., 2010; DUSS, 2015). We divide it into slots that correspond to a standard railcar of 14 meter length and 7 meters vertical distance between the tracks. Similar parameters were used by Boysen et al. (2010) and Boysen and Fliedner (2010). Further, a terminal segment typically consists of about two to four parallel tracks (Ballis and Golias, 2002; DUSS, 2015) and contains up to four cranes (Boysen et al., 2013; DUSS, 2015). Therefore we designed our data sets as follows. To investigate the impact of the number of parallel tracks, we randomly generated data sets with four cranes and two, three and four parallel tracks called $C4T2$, $C4T3$ and $C4T4$, respectively. To study the impact of the number of cranes, we look at instances with two parallel tracks and two, three and four cranes called $C2T2$, $C3T2$, $C4T2$, respectively. Each data set contains 25 instances, so that we have $25 \cdot 5 = 125$ instances in total.

Trains. The train lengths may vary. We have drawn them from a normal distribution

$N(\mu_{train}, \sigma_{train})$ with expected train length of $\mu_{train} = 43$ slots, which corresponds to about 600 meters (see Ballis and Golias, 2002). To mimic the situation that yards differ in the length of tracks and thus in their capacity to handle long trains, we randomly set the standard deviation σ_{train} to 2, 4, 6 or 8 for each instance (cf. Boysen and Fliedner, 2010). Without loss of generality and following a common practice in the literature, we let each train to park from the first slot.

Since freight trains also transport empty railcars and some trains end at the rail-yard, the number of jobs n is a fraction of the number of possible container positions determined by the total length of trains. We set this fraction at $Share = 40\%$ on average (cf. Boysen and Fliedner, 2010). We generated the parameter Share from a normal distribution with expectation of 40% and standard deviation of 5.

Set-up times. In our data set, we mimic technical crane parameters similar to studies of Boysen et al. (2010) and Boysen and Fliedner (2010). Independent engines enable cranes to move a container simultaneously in both directions, parallel to the tracks (horizontal) as well as orthogonally to the parallel tracks (vertical) whereby the whole crane is progressing in the horizontal direction, the steering cab carrying the spreader is moving in the vertical direction. We assume horizontal and vertical velocities of the crane carrying a container at $v^l = 2$ meters per second. When the crane is not loaded, its horizontal and vertical velocities increase to $v^u = 3$ meters per second. Picking and dropping a container requires fixing (unfixing) the gripping hooks as well as a careful positioning of the spreader. This precision task takes about 45 seconds on average. For example, the setup time s_{25} between job 2 and job 5 in Figure 2.2 involves travelling of the crane 1 slot vertically and 4 slots horizontally; picking container of job 5; moving it 2 slots horizontally and 1 slot vertically as well as dropping the container: $s_{25} = \max\{4 \cdot 14\,m/(3\,m/sec); 1 \cdot 7\,m/(3\,m/sec)\} + 45\,sec + \max\{2 \cdot 14\,m/(2\,m/sec); 1 \cdot 7\,m/(2\,m/sec)\} + 45\,sec = 122.7\,sec.$

Jobs. As common in a typical transshipment yard (Rotter, 2004), in each instance, about 2/3 of the jobs are generated as rail-road and the rest are rail-rail (25% of jobs) and rail-storage jobs (10% of jobs). For each job, its pickup horizontal positions were determined randomly from a uniform distribution. For rail-road jobs, we realistically assume that vehicles park at the most convenient parking position (cf. Figure 2.2). For rail-rail and rail-storage jobs we assume that container positions have already been optimized after arranging trains to bundles and selecting suitable parking positions. Therefore we limit the difference in *horizontal positions* for rail-rail and rail-storage jobs and generate it randomly from $\{-1, 0, 1\}$. For example, in Figure 2.2 job 2 starts in slot 8 and ends in slot 7, so that the *difference in horizontal positions* is $7 - 8 = -1$. Note as European freight trains are one-story (due to low height of electric wires), we do not allow to pick more than one container from the same slot.

To imitate the current situation, time windows are kept wide. Let us define p_j as processing time of job j, i.e., it is the time needed when the crane's grip hooks are fixed to pick the container until it is dropped at the final position. The size of the time window $(\tau_i^d - \tau_i^e)$ is uniform distributed in the interval $[RV/2; 2 \cdot RV]$ where $RV = \sum_{i \in J} p_j \cdot \gamma / R$ is a reference value and measures the total processing time per crane with some allowance $(\gamma = 1.5)$ for setup times. Note that narrow time windows will further favor to jointly consider yard partition and job scheduling together, as we promote in this paper, i.e., the advantage of using the TBDP framework is more significant for small time windows. Moreover, run times for the exact algorithm will decrease. We randomly select the release time τ_i^e among four options: $0, RV/4, 2 \cdot RV/4$ and $3 \cdot RV/4$. We generated time windows for rail-road jobs (i.e., for about two thirds of the jobs).

Descriptive statistics of our data sets is provided in Table 2.5. We performed tests on a PC with Intel i5-3450 3.1GHz CPU and 8.0 GB RAM. The implementation of the algorithms is in C++, utilizing the library of IBM ILOG CPLEX 12.6.

Table 2.5: Descriptive statistics of the data sets

Data set	No. of cranes R	No. of tracks	No. of spatially disjoint sets K		No. of jobs $\lvert J\rvert$	
			Avg.	[min, max]	Avg.	[min, max]
$C4T2$	4	2	21	$[15, 27]$	32	$[20, 45]$
$C4T3$	4	3	24	$[17, 31]$	52	$[41, 68]$
$C4T4$	4	4	25	$[19, 30]$	63	$[49, 75]$
$C3T2$	3	2	21	$[14, 28]$	34	$[28, 42]$
$C2T2$	2	2	20	$[15, 26]$	32	$[26, 37]$

In Experiment 1 (Section 2.6.1) we investigate the performance of the TBDP heuristic and of TEMP. A detailed additional value analysis of the proposed TBDP framework for SCSP, is studied in Experiment 2 (Section 2.6.2).

The proposed solution methods, TEMP and TBDP, are designed for the case, when no container moves are performed across crane zones. This policy is reasonable if there is a sufficient number of spatially disjoint sets of jobs and it requires a low-to-medium share of rail-rail moves, low variety of container sizes, which is common for international traffic, and optimized container positions resulting in short container moves. Such conditions, for example, can be found in gateway traffic, described in the Introduction. In Appendix 2.B, we examine the expected number of spatially disjoint sets of jobs for different operational parameters of the transshipment yard.

2.6.1 Experiment 1: Performance of the TBDP heuristic and of TEMP

TBDP heuristic. TBDP heuristic found optimal solutions in 90% of the total 125 instances (see Table 2.6). For instances not solved to optimality, the average relative deviation from the optimal objective value (*average performance*) was about 1% and never exceeded 2.5% (*worst performance*). We conclude that the TBDP heuristic performs sufficiently well for many practical purposes.

Table 2.6: Performance of TBDP heuristic and of exact algorithm TEMP

Data set	$C4T2$	$C4T3$	$C4T4$	$C3T2$	$C2T2$
Performance of the TBDP heuristic					
No. of cases, where optimum was found	24	23	19	23	24
Avg. performance of instances not solved optimally (%)	1.3	0.2	0.3	1.3	1.1
Worst performance (%)	1.3	0.2	0.7	2.5	1.1
Performance of exact algorithm TEMP					
Avg. computational time (sec.)	2	10	113	4	27
Performance of TEMP after the first bounding iteration					
No. of cases with $GUB = GLB$	9	2	2	3	1
Avg. relation of GLB to optimum (%)	99	99	99	98	99
Performance of TEMP compared to the current routines in operational practice					
Avg. relative improvement to practice by TEMP (%)	23	28	27	22	18
Performance of a standard solver (IBM ILOG CPLEX 12.6, 1 hour time limit)					
No. of cases, where a feasible solution was found	5	0	0	4	14
No. of cases, where optimum was found	3	0	0	2	4
No. of cases, where optimum was proven	2	0	0	1	1
Avg. relation of GLB to optimum (%)	91	87	86	89	91

TEMP. TEMP took 30 seconds per instance on average. As the B&B algorithm for the single crane scheduling problem is the most computationally expensive part of TEMP, run times differ a lot among the data sets (see Table 2.6). Overall, the more jobs (or tracks) a crane handles the larger is the run time. We were able to solve all the instances to optimality within the run time limit of 30 minutes. The maximal run time per instance comprised 23 minutes.

In practice, an early estimation of the solution quality is much appreciated. Therefore we analyzed the results of TEMP *after the first bounding iteration* (cf. Section 2.3.2). We compared, how the gap GLB/GUB is related to the gap GLB/optimum. Overall for only 15% of the instances, for which the optimum was actually found (but not necessarily proven), both bounds were the same, i.e., $GUB = GLB$ (see Table 2.6). However, the GLB provides a good approximation of the optimal solution. On average, the relation of GLB to the optimal objective function value comprised 99% and never dropped below 96% of the optimal value. Notice that if TEMP is stopped after the first bounding iteration, it is equivalent to the TBDP heuristic.

Routines in practice. It is interesting to compare the proposed solution methods to the routines used currently in practice (cf. Boysen and Fliedner, 2010; Boysen et al., 2010). According to practitioners, the yard is partitioned in zones of about equal size. The crane operator determines a sequence of jobs. The operator's decision criterion resembles in most cases the earliest deadline rule (EDD_j). Compared to these planning routines, TEMP was able to reduce the resulting makespan by 18% to 28% on average (see Table 2.6).

Standard solver. In these computational tests, we introduce redundant constraints into the SCSP model (see Section 2.2) in order to improve the performance of the standard solver. IBM ILOG CPLEX 12.6 was able to find a feasible solution for only 23 (or 18% of) instances within a one hour run time. On average, the GLB of CPLEX was about 89% of the optimum value and dropped even to 79% of the optimal value in the worst case (see Table 2.6).

Large instances. Although large transshipment yards with more than four tracks are rare, we also compared the performance of the TBDP heuristic for 6 tracks and 2 to 6 cranes $C2T6$, $C3T6$, $C4T6$, $C5T6$ and $C6T6$ with the current situation in practice. Each data set contains 25 instances generated in the same way as described above. We applied the TBDP heuristic, because B&B for a single crane instance often takes more than an hour if there are more than 20 jobs per crane. Table 2.7 shows that the TBDP was able to reduce the makespan by 13% to 26% on average compared to planning routines in practice. For some instances the improvement reached 45%.

Table 2.7: Performance of TBDP heuristic compared to the current routines in use for large transshipment yards

	$C2T6$	$C3T6$	$C4T6$	$C5T6$	$C6T6$
Avg. relative improvement in % to results in practice	13	15	20	25	26

2.6.2 Experiment 2: Additional value of TBDP

It is interesting to illustrate and measure the advantages of TBDP as a part of our computational experiments.

Firstly, the TBDP indicates *critical crane zones* (see our discussion in Section 2.3.1) early in the solution process. For example, in TEMP, a single crane job scheduling problem (step 4 in Figure 2.4) had to be solved only for few critical crane zones. Indeed, B&B algorithm for a single crane problem was on average only called twice and never more than eight times per instance.

Further, the TBDP framework allows to receive *strong lower bounds* early in the

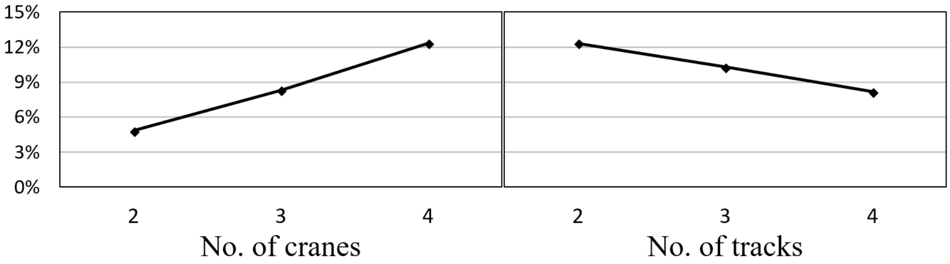


Figure 2.5: TBDP provides sharp lower bounds: relative improvement of GLB after the first bounding iteration, which is based solely on LB2, called GLB^{LB2} compared to $LB2^{SCSP}$

solution process. While constructing a lower bound, TBDP takes additional information into account, on which combinations of the crane zones are possible.

For example, consider the lower bound based on the linear sum assignment problem. We can apply this bound to the SCSP assuming that there was only one crane. Therefore we add R dummy source jobs and R dummy sink jobs and apply same transformations as in case of LB2 (see Section 2.4.1). The value of the optimal solution of the resulting linear sum assignment problem provides us an estimation from below of the *total* time to be spend by all the cranes. If we divide this value by the number of cranes R we get a lower bound, which we call $LB2^{SCSP}$.

Alternatively, we apply the relaxation idea to the single crane job scheduling problems. Notice that this is the idea behind LB2. Then we aggregate via the introduced dynamic programming procedure (step 2 in Figure 2.4), we get a sharp GLB. We call this GLB as GLB^{LB2}.

We compared the results of $LB2^{SCSP}$ to GLB^{LB2} (Figure 2.5). TBDP helped to improve the lower bound by 5% to 12% on average. For some instances, GLB^{LB2} was impressive 35% higher than $LB2^{SCSP}$. We also observe that in line with intuition, TBDP returns a sharper lower bound with an increasing number of cranes. The difference between GLB^{LB2} and $LB2^{SCSP}$ slightly decreases with an increasing number of tracks. The reason is as follows. A larger number of tracks usually corresponds to a larger number of jobs per crane and to more "balanced" makespans of single crane zones with the same number of jobs. So that the average time per crane, which is the main idea behind $LB2^{SCSP}$, gets closer to the optimal makespan of SCSP.

Thirdly, TBDP enables to *consider complex computational planning problems*: yard partitioning and job scheduling - *together*. While considering possible yard partitions, we anticipate possible schedules of jobs in advance and therefore improvve the quality of planning as well as avoid infeasible solutions.

A hierarchical planning of yard partitioning and job scheduling (cf. Boysen et al., 2013), without anticipating the results of job scheduling, needs to evaluate each crane

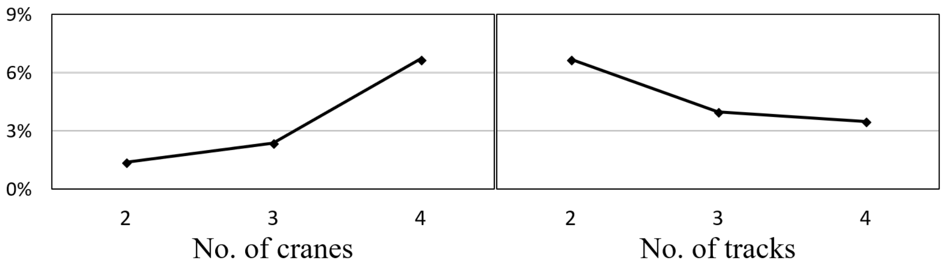


Figure 2.6: TBDP vs. decomposition approaches: exact algorithm TEMP vs. hierarchical planning approach HA

zone by some surrogate value function, for example, by a simple sum of job processing times $\sum_{j\in J'} p_j$ (cf. Boysen and Fliedner, 2010; Boysen et al., 2010). Such a hierarchical approach (HA) results in efficiency losses and complicates promising tight service times to priority customer vehicles, because a HA may be unable to find available feasible solutions.

We implemented the hierarchical approach as follows. Yard partitioning, which is a *first-tier planning problem*, is solved by a dynamic programming approach. Nodes and transition arcs of the state graph are formulated in the same way, as in Section 2.3. However, the value function $F()$ is different. The recursive formula for the value function, which is a counterpart of formulas (2.19) and (2.20), is $F(r,k) = \min_{k'<k}\{F(r-1,k') + \sum_{j\in J'} p_j\}$, where $J' = \{j \in J | a_{fj} = 1, f \in \{k'+1, k\}\}$. When the yard partitioning is obtained using dynamic programming, we perform the scheduling of jobs as a *second-tier* planning problem. Thereby, scheduling of jobs is performed using our branch and bound procedure (see Section 2.5).

Figure 2.6 shows that hierarchical planning of yard partitioning and job scheduling in HA leads to 1.5% to 7% deterioration in the received makespan. The worst observed deterioration in the makespan comprised 14%. The disadvantage of a hierarchical decomposition is even more obvious with the increasing number of cranes and with the decreasing number of jobs per crane.

2.7 Conclusion

The proposed two-way bounded dynamic programming procedure is an effective tool, in case it takes long to compute the value function within dynamic programming. TBDP provides not only sharp initial bounds, but also navigates to critical states and transition arcs within the state graph, for which the value functions have to be computed foremost. As a consequence, TBDP enables to solve hard instances, for which hierarchical planning decomposition previously had to be applied.

We propose an exact algorithm TEMP and a heuristic approach, both based on TBDP.

TEMP solves instances of practically relevant size within 30 minutes time limit and an average from 2 seconds to 2 minutes. We recommend using the proposed heuristic in cases of very large transshipment yards, with more than five tracks and more than about 30 jobs per crane. The TBDP heuristic finds an optimum in about 90% of the cases; its solution does not exceed the optimal makespan by more than 2.5% in our experiments.

Overall, we highly recommend performing yard partitioning and job scheduling together. Hierarchical decomposition leads to efficiency losses of up to 14%. Thereby the losses become more pronounced, if more cranes or less jobs per crane are present.

Several topics remain for further research. First of all, we assumed that no container transshipment between the crane zones occurs. Certainly, at least variable costs may be reduced by introducing sorter vehicles, which are carrying out such transshipments. In this case, our model serves as a baseline to determine the additional value of such sorters. Secondly, although static crane zones are currently preferred because of organizational simplicity, the dynamic policies, i.e. when cranes do not have to stay within their defined zones, may lead to a better makespan. Therefore, it is useful to investigate this case in context of railway transshipment yards. Overall, with progressing automation and better data processing at transshipment yards, the importance of holistic planning approaches is growing. Holistic planning approaches combine several planning steps and make the trade-offs between single decisions transparent. For example, container placement that minimizes distances of container moves leads to time and cost savings in the container transshipment operations. On the other hand, we may place containers to minimize the wind resistance of the train in order to prolong the servicing life of brakes and to reduce the energy consumption. A holistic planning approach, which combines container placement with job-to-crane assignments and jobs sequencing compares these trade-offs to enable an optimized decision.

2.A Appendix A. LB1

Since all the jobs have to be assigned in a sequence, the makespan is at least $LB1 = \sum_{i \in J'} p_i + \sum_{i \in J'} \min_j \{\vartheta_{ij} | i, j \in J', (j,i) \notin E\} - \max_i \min_j \{\vartheta_{ij} | i, j \in J', (j,i) \notin E\}$. Obviously, LB1 can be calculated in $O(|J'|^2)$ time as each factor of the lower bound requires linear time, if the minimum for each job i has been determined in an earlier step which also requires only linear time.

LB1 can be further improved by relaxing the single crane scheduling problem to the problem $1/e_i/C_{max}$ which can be solved in polynomial time by sequencing the jobs in non-decreasing order of their releases e_j. Thus assume all jobs of set J' are ordered in non-decreasing order of their release times with job $[J']$ being the last job in this ordering.

We assign the jobs sequentially one after another and assume $[J']$ completes at time $C_{[J']}$. The time for processing is assumed to be $p'_i = p_i + \min_j\{\vartheta_{ij}|i,j \in J', (j,i) \notin E\}$ for all jobs $i \in J'$. Additionally define $\vartheta^{max} := \max_i \min_j\{\vartheta_{ij}|i,j \in J', (j,i) \notin E\}$. If $e_{[J']} + p_{[J']} \leq C_{[J']} - \vartheta^{max}$ then LB1 is equal to $C_{[J']} - \vartheta^{max}$, otherwise LB1 is equal to $e_{[J']} + p_{[J']}$.

In the example with four jobs $J' = \{1,2,3,4\}$ and parameters from Tables 2.2 and 2.A1, jobs are assigned in the sequence $(1,2,3,4)$ and $\vartheta^{max} = 3$. LB1 equals to 12 (modified processing times $p'_{[f]}$ are shown in Table 2.A1).

Notice that in general $\min_j\{\vartheta_{ij}|j \in J'\}$ is not equal to $\min_j\{\vartheta_{ji}|j \in J'\}$. So, we can reformulate the aforementioned calculation with "incoming" setup times and choose the larger lower bound as LB1.

Table 2.A1: Additional parameters for the illustrative problem instance given in Table 2.2

Job $j =$	1	2	3	4
p_j	1	2	1	2
e_j	0	1	3	3
p'_j	3	4	4	4

	setup time $\vartheta_{ij} = s_{ij} - p_j$			
	j			
i	1	2	3	4
1	-	3	2	2
2	2	-	3	2
3	4	5	-	3
4	3	2	2	-

2.B Appendix B. Terminal Settings and the Number of Spatially Disjoint Sets of Jobs

In this section we report on results of two computational studies. We focus on a general case and quantify the criteria, when container moves across crane zones are not reasonable, which implies that our solution approach can be applied without any changes. As a rule of thumb in case of two to four cranes, we require at least 8 to 10 spatially disjoint sets of jobs.

In a first computational study, we generated instances with 50 slots and, to examine the least favorable case, assumed that all containers are to be moved. Notice that this setting is different from our simulation in Sections 2.6.1 and 2.6.2, where we limited the number of jobs to 40% of the number of railcars in a train in order to account for trains being compiled at this terminal, empty railcars or containers not being transshipped at this terminal. We varied the number of tracks, the fraction of rail-rail and rail-storage jobs θ as well as the maximal horizontal distance in number of slots between the pickup

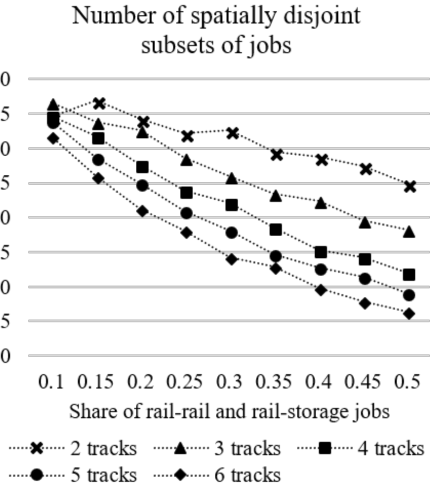


Figure 2.B1: Number of spatially disjoint sets of jobs for 50 slots and a horizontal distance to the new container position of at most 1 slot

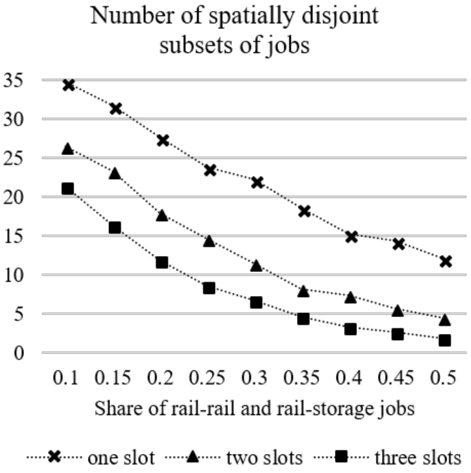


Figure 2.B2: Number of spatially disjoint sets of jobs for four tracks and 50 slots. Curves visualize the maximal distance to the new horizontal container position in number of slots

and drop-off position of a container. We generated 20 instances for each setting (see Figure 2.B1 and 2.B2). Note that there are spatially disjoint sets with rail-rail and rail-storage jobs, as well as single-slot sets of rail-road jobs. To provide a full picture, we also report the number of spatially disjoint sets for different values θ excluding single-slot sets rail-road jobs (see Table 2.B1). As a result Figure 2.B1 shows that even if the share of rail-rail and rail-storage jobs raises up to 50%, we still get more than 10 spatially disjoint sets of jobs for a terminal with four tracks. Even in case of very large terminals with six tracks, we get more than eight spatially disjoint sets for $\theta \leq 0.45$. Not surprisingly the maximal horizontal distance in number of slots to the new container position has a profound impact on the number of spatially disjoint sets (see Figure 2.B2). For a distance of up to 3 and $\theta < 0.25$ we receive more than 8 spatially disjoint sets. In a common situation where $\theta = 0.35$ there are more than 10 spatially disjoint sets even for distances of up to two slots.

Table 2.B1: Number of spatially disjoint sets excluding single-slot sets of jobs in case of four tracks and 50 slots

	Share of rail-rail and rail-storage jobs θ								
	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
Max. distance in slots to the new position									
1	10.9	14.0	15.9	16.7	17.4	17.2	16.5	16.6	15.7
2	9.7	11.4	12.1	11.6	11.2	10.1	9.1	8.0	7.1
3	8.1	9.0	8.9	7.6	7.1	6.4	5.3	4.5	4.1

In the second computational study, we assume containers and railcars of varying sizes. Especially in the national traffic, sizes of railcars and containers may differ. For example, trains to or from ports regularly carry about 50% 20-foot and 50% 40-foot containers (cf. EU Commission, 2015). A European transshipment terminal may handle up to 23 types of containers or swap bodies and railcars are of lengths between 40 and 104 feet (12.2 to 31.7 meter) (see Bruns and Knust, 2012). The corners castings of containers are fixed to the railcars with specialized pins. There are three possible fixation types of containers and swap bodies with the following distances between the corner castings: 20 feet ($5,853 \pm 3\,mm$), 30 feet ($8,918 \pm 4\,mm$) or 40 feet ($11,985 \pm 5\,mm$). Note that the positions of pins at the railcar and the fixation type of the container restrict possible loading patterns of the railcar.

In our computational study, we randomly generated railcars of three types with lengths 13, 23 and 32 meters, and three load patterns for each railcar type depending on the fixation type of the containers (see Table 2.B2). For example, load pattern $\{3.5, 9.5\}$ consists of two containers with midpoints at 3.5 meters and 9.5 meters from the left edge of the railcar, respectively.

Table 2.B2: Load patterns for railcar and fixation types used in the computational study

	Fixation type 1	Fixation type 2	Fixation type 3
13-meter railcar	$\{3.5, 9.5\}$	$\{7.5\}$	$\{6.5\}$
23-meter railcar	$\{4.5, 11.5, 18.5\}$	$\{7.5, 16.5\}$	$\{14.5\}$
32-meter railcar	$\{4.5, 11.5, 18.5, 25.5\}$	$\{7.5, 16.5, 25.5\}$	$\{10, 23\}$

We randomly generated railcars to compile trains of up to 650 meters. For each railcar, we draw a random load pattern. Following our experimental settings in Sections 2.6.1 and 2.6.2, we generated instances with 25% of rail-rail, 10% of rail-storage jobs and 65% of rail-road jobs. We set the number of tracks to 2, 3 or 4. Since only a fraction of containers is moved at a transshipment yard, we also vary the probability of a container to be moved between 5% and 95%. We call this parameter the "fraction of container movements" in Figure 2.B3. We examined 20 instances for each setting, which makes $3 \cdot 19 \cdot 20 = 1140$ instances in total.

Note that we cannot measure the horizontal distance of a container's current position to its new position because of varying container and railcar sizes. Therefore we generated pickup and drop-off positions of rail-rail and rail-storage jobs as follows. Given a random pickup position, we set the drop-off position to the horizontally nearest available location with the same fixation type, which does not belong to the same train.

Because of safety reasons, we do not allow that two containers (of two jobs, respectively) with the distance between the midpoints less than 10 meters belong to different

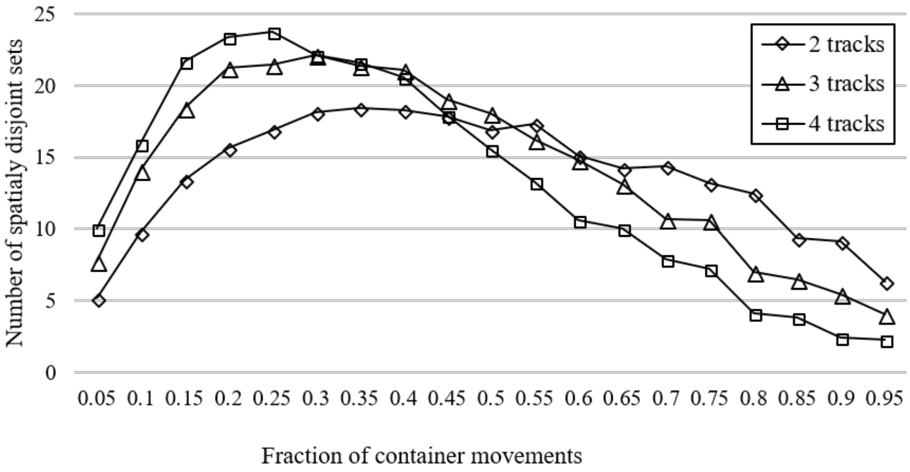


Figure 2.B3: Number of spatially disjoint subsets in case of varying types of railcars and containers

spatially disjoint sets.

Figure 2.B3 reports the computational results. Note that if very few jobs are available, then almost every single job forms a spatially disjoint set. Therefore, the relation of the number of spatially disjoint sets to the fraction of container movements has an concave U-shape. In line with the intuition and our previous results, the curve for 4 tracks is the steepest to rise and the fastest to decline, so that the number of tracks is an important determinant of the number of spatially disjoint sets. Overall, we get more than 10 spatially disjoint sets unless the fraction of container movements exceeds 0.60.

Our computational experiments show that limiting container moves to their crane zones is reasonable for a wide range of settings provided that moves are short, e.g., moves to neighbor slots. In case of moderate container moves over distances of up to 3 slots, the fraction of rail-rail and rail-storage moves to the number of railcars should not exceed 25-35% depending on the number of tracks.

Chapter 3

Solving the single crane scheduling problem at rail transshipment yards

We consider single-crane scheduling at rail transshipment yards, in which gantry cranes move containers between trains, trucks and a storage area. The single-crane scheduling problem arises at single-crane transshipment terminals and as a subproblem of the multiple-crane scheduling problem. We consider a makespan objective function, which is equivalent to minimizing the train dwell time in the yard, and introduce time windows for container moves, for example, as a customer service promise. Our proposed decomposition algorithm with integrated dynamic branch-and-cut or dynamic programming solves practically relevant instances within short time limits.

3.1 Introduction

Freight container transportation has been growing steadily in the last years. The amount of goods transported in containers increased by 50% in Germany in the ten years from 2004 to 2014 (www.destatis.de). In comparison to freight railcars, containers are standardized transport units and allow significant savings on handling cost and time, especially during consolidation at transshipment yards. Optimization of yard operations enables to exploit this time and cost reduction potential of containers to its maximum.

The layout of a typical rail terminal for the container transshipment includes four to six tracks, two lanes for trucks – a driving lane and a parking lane –, and a storage area (see Figure 3.1). One or several gantry cranes transport containers between trains (rail-rail transshipments), between trains and trucks (rail-road transshipments) or between trains and the storage (rail-storage transshipments). In this paper, we consider scheduling of a *single crane*. Single-crane terminals are quite widespread and can be found, e.g., at

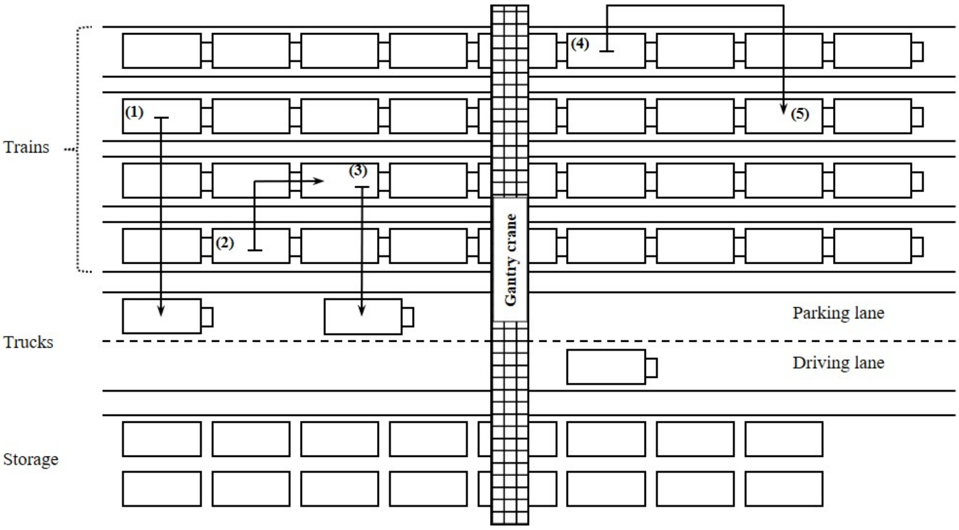


Figure 3.1: Illustration of a typical rail terminal

transsshipment yards in Göttingen, Beiseförth, or Heilbronn. The single-crane scheduling problem also arises as a subproblem of crane scheduling problems with several cranes (eg. Froyland et al., 2008; Nossack et al., 2018; Otto et al., 2017; Souffriau et al., 2009).

We define J to be a set of n jobs each representing a container move. The processing time p_j of *job* $j \in J$ is the duration of a container move that consists of a series of operations like fixing the crane's grip hooks at the target container, lifting it, and moving it to its target position where the grip hooks are released. The time for moving the crane from the position where the current job j has been completed (drop-off position of the container) to the starting position of the next job j' (pick-up position of the next container) may be considered as a *setup time* $\vartheta_{jj'}$ between the two jobs. Cranes have to load and unload the customer trucks fast, within the promised time windows. Therefore we specify a time window $[r^o_j, d_j]$ for each job $j \in J$ where the release time r^o_j is the earliest possible starting time of job j and the deadline d_j specifies the time when j is supposed to be finished at the latest. The first and the last job to be processed from set J are dummy jobs and determine the starting and the desired final position of the crane and will be addressed as source job a and sink job z, respectively. Therefore, we set $p_a = p_z := 0$, $r^o_a = r^o_z := 0$, $d_a := 0$, $d_z := \infty$.

In some cases, it might happen that the container of j has to replace the container of job j', as for example in case of job 2 and job 3 of Figure 3.1. Then job j' has to precede job j described by the order $(j', j) \in E$, where set E is a set of precedence constraints between jobs. We enforce the source job to be the first and the sink job to be the last in any feasible sequence of jobs by introducing precedence constraints $(a, j) \in E$ and $(j, z) \in E$ for all $j \in J \setminus \{a, z\}$.

To simplify the notation, we define *modified* setup times as $s_{ij} = \vartheta_{ij} + p_j$ and modified release times as $r_j = r_j^o + p_j$. Observe that the triangular inequality is satisfied for the setup times, i.e. $s_{jk} + s_{kl} \geq s_{jl}, \forall\, j, k, l \in J$. We introduce the completion time of job j as decision variable C_j.

A *bundle* of trains arrives together at the yard for container transshipment and departs from it jointly (see Bostel and Dejax, 1998; Boysen et al., 2011; Rotter, 2004). Specifically, for safety reasons trains are not allowed to move while a crane is working. Thus minimizing the makespan leads to a short handling time of each bundle and accelerates the throughput. The first exact methods on partitioning trains into the bundles and on scheduling arrival of the bundles are described in Boysen et al. (2011, 2012) as well as in Barketau et al. (2013).

Therefore the *Single-Crane Scheduling Problem (SgCSP)* is to find a schedule with a *minimal makespan* C_z, i.e. an optimal sequence $P^* = (j_1, j_2, ..., j_n)$ of the n jobs starting with source job a and finishing with sink job z and completion times C_j for each job $j \in J$, such that:

- time windows constraints are satisfied: $r_j \leq C_j \leq d_j, \forall\, j \in J$,
- precedence constraints are satisfied: $\forall\, (j_k, j_l) \in E : k < l$,
- setup times are observed: $C_{j_k} \geq C_{j_{k-1}} + s_{j_{k-1}j_k}, \forall\, k \in \{2, ..., n\}$.

In the next steps, we develop exact solution approaches for the SgCSP.

Articles on the crane scheduling at transshipment yards mostly treat the single-crane scheduling problem as a part of a more general planning problem and solve it heuristically. For example, Alicke (2005) examines the multiple-crane scheduling problem at transshipment yards with a sorter system, which is to assign jobs to cranes and to determine the sequence of jobs. He formulates a constraint satisfaction problem and develops different heuristics to minimize maximum lateness. Souffriau et al. (2009) consider the two-crane scheduling problem without time windows and the objective to minimize the makespan. They simultaneously determine container positions on trains, assign the jobs to cranes and decide on the sequence of the jobs. The authors solve the last subproblem, which is similar to the SgCSP, with a variable neighborhood search procedure. Otto et al. (2017) consider the multiple-crane scheduling problem, where cranes are working in separate areas. They solve the arising SgCSP subproblem with a branch-and-bound algorithm. Barketau et al. (2015, 2016) focus on the assignments of the containers to the drop-off positions in case there is no unique position predefined for each container. They prove that the new assignment problem is NP-hard in the strong sense and derive a best approximation result. Cichenski et al. (2017) introduce the first holistic model that solves simultaneously the train bundling, train-to-track assignment and the selection of the container drop-off positions for instances relevant in practice. Kress et al. (2015) allow the pick-up positions and therefore the assignment of containers to cranes be a part of the optimization. Stephan and Boysen (2017) analyze computational complexity of different formulations of the single-crane scheduling problem. A detailed survey on the container processing at rail transshipment yards is provided by Boysen et al. (2013).

Several articles focus on single crane scheduling in the context of port container terminals. Thus, Ng and Mak (2005) address the single-crane scheduling problem with the objective to minimize the sum of job waiting times. They neither include precedence constraints nor deadlines of jobs. The authors propose a branch-and-bound algorithm, which solves the generated test instances with up to 25 jobs. Montemanni et al. (2010) propose a local search algorithm and an ant colony metaheuristic for the single-crane scheduling problem without time windows and the objective of minimizing the total cost. In other papers the single-crane scheduling problem is part of a more general framework. For example, Kim and Park (2004) propose a mixed-integer model for a multiple-crane scheduling problem with the objective to minimize makespan. They develop a branch-and-bound algorithm to solve small instances to optimality and a customized heuristic algorithm for instances of practically relevant size. Alsoufi et al. (2015) suggest a mixed-integer model that combines crane scheduling with the berth allocation problem. They solve small-scale instances with off-the-shelf software CPLEX. Sha et al. (2017) propose a mixed-integer model for crane scheduling to minimize the energy consumption. Fedtke and Boysen et al. (2017) set up an optimization problem for simultaneous crane and shuttle car scheduling problem in rail-rail transshipment yards. The authors solve single-crane scheduling subproblems with a beam search heuristic based on the dynamic programming algorithm of Held and Karp (1962). Dik and Kozan (2017) design a hybrid metaheuristic based on the variable neighborhood and tabu search for the multiple crane scheduling problem. We also refer the reader to the recent surveys on crane scheduling with interference by Boysen et al. (2017). The classification scheme, complexity results and close to optimal approximation algorithms for twin cranes operating on the same level can be found in Kovalyov et al. (2018) and the fast branch and cut algorithm for dual cranes operating on different levels are proposed by Nossack et al. (2018). Barketau and Pesch (2016) represent the single crane scheduling problem without time windows as a special case of the asymmetric traveling salesman problem and design an approximating algorithm with an approximation guarantee of 3. More about container processing at seaports can be found in Stahlbock and Voß (2008) as well as Bierwirth and Meisel (2010, 2015).

Notice that the SgCSP is an extension of the asymmetric traveling salesman problem with time windows (ATSPTW) where we additionally allow precedence constraints. The asymmetric traveling salesman problem (ATSP) with the makespan objective function, in the literature also referred to as the *minimum completion time problem* (MCTP), has a more general variant, the *minimum tour duration problem* (MTDP), with a flexible starting time of the tour. According to this terminology, the SgCSP is equivalent to the MCTP with time windows and precedence constraints. Although a plentitude of solution approaches has been developed for the ATSP and its variants, the MCTP and the MTDP have rather rarely been investigated. To the best of our knowledge, the available exact solution approaches for the MCTP and the MTDP are dynamic programming or branch-and-bound approaches (e.g. Baker, 1983; Christofides et al., 1981; Cire and van Hoeve, 2013; Langevin et al., 1993; Tilk and Irnich, 2017).

The SgCSP is also related to the *single-machine scheduling problem* with sequence-dependent setup times, precedence relations, time windows, and the makespan objective function, or $1/r_j, d_j, prec, s_{ij}/C_{max}$ in notation of Graham et al. (1979). Literature reviews on the machine scheduling literature with sequence-dependent setup times can be found in Allahverdi et al. (1999, 2008) and Allahverdi (2015). Machine scheduling problems are often motivated by manufacturing or hospital applications, so they mostly contain a number of specific features and constraints, such as time-dependent setup times, rate-modifying activities, or scheduling of maintenance work (cf. Ivanov et al., 2009; Nesello et al., 2018; Pei et al., 2017; Silva et al., 2018; Stecco et al., 2008). A few papers propose exact solution methods - branch-and-bound, dynamic programming, and constraint programming algorithms - for $1/s_{ij}, \cdot/\cdot$ with different objective functions (makespan: Bianco et al. (1988); sum of weighted completion times: Chou et al. (2009); tardiness-related objectives: Bigras et al. (2008); Kim and Lee (2009); Luo and Chu (2007); Luo et al. (2006); Luo and Chu (2006); Tanaka and Araki (2013)). However, most of these algorithms neglect time windows. Time windows significantly increase the intractability of the SgCSP because waiting times may have to be included into a schedule (Bigras et al., 2008; Kim and Lee, 2009; Luo and Chu, 2007; Luo et al., 2006; Luo and Chu, 2006; Tanaka and Araki, 2013). To the best of our knowledge, none of these algorithms can be readily applied to the SgCSP.

In this paper, we design a decomposition algorithm (DA), which is a general approach to decompose the initial problem for a set of jobs into smaller and easier to solve problems for subsets of jobs. The DA is a superstructure that requires an additional algorithm to solve the resulting SgCSP problems. Therefore, we propose a decomposition algorithm where subproblems are solved by a dynamic branch-and-cut procedure (DBC). To the best of our knowledge the designed DBC is the first branch-and-cut algorithm for the SgCSP and the closely related problem of the MCTP.

Next, we discuss alternative problem formulations of the SgCSP, that will be used in our solution approaches. Section 3.3 and Section 3.4 describe the DBC and the DA (and its combination), respectively. We report on extensive computational experiments in Section 3.5 and finally conclude with an outlook in Section 3.6.

3.2 Problem formulations for the single-crane scheduling problem

In Section 3.2.1, we provide an illustrative example on the SgCSP, introduce some additional notation, and formulate an observation that we will actively use in our solution procedures.

To solve the SgCSP with a branch-and-cut algorithm (see Section 3.3), we will use a problem formulation with a surrogate objective function – minimize total cost. This problem formulation, which we call the SgCSP-Mincost, has a rather tight LP-relaxation and therefore allows to speed up the branch-and-cut algorithm significantly according our preliminary evaluations. In Section 3.2.2, we describe the SgCSP-MinCost and its relation to the SgCSP.

Observe that since the SgCSP closely relates to machine scheduling as well as to routing problems (see Section 3.1), we use suitable terminology from both research fields. For instance, terms "paths" and "Hamiltonian paths" enable us to explain, how we apply effective path-elimination-constraints in our branch-and-cut procedure, whereas we use the term "active schedule" to succinctly explain how we can reduce the solution space and schedule jobs given their sequence.

3.2.1 Some observations on the single-crane scheduling problem

Consider the example of Table 3.1 with six jobs, where job 3 has to precede job 2: $E = \{(3,2),(a,1),(a,2),(a,3),(a,4),(1,z),(2,z),(3,z),(4,z)\}$. Figure 3.2 shows an optimal sequence of jobs $(a,1,3,2,4,z)$ with makespan 16 and completion times C_j of 0, 3, 8, 10, 16 and 16, respectively.

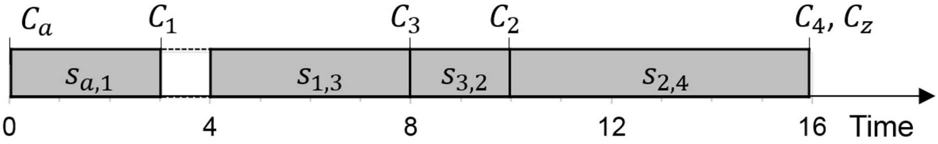


Figure 3.2: Illustration of an optimal solution for the example in Table 3.1

Observe that the constructed schedule of jobs will remain optimal if job 1 is rescheduled to finish anywhere between 3 and 4. Indeed, for a given sequence of jobs, we will be

Table 3.1: Example of a SgCSP problem instance

Job $j =$	a	1	2	3	4	z
r_j	0	3	6	8	16	0
d_j	0	6	10	14	18	∞

	setup time s_{ij}					
	j					
i	a	1	2	3	4	z
a	–	3	1	2	2	0
1	∞	–	2	4	7	0
2	∞	5	–	2	6	0
3	∞	6	2	–	6	0
4	∞	10	7	7	–	0
z	∞	∞	∞	∞	∞	–

able to construct a plentitude of feasible job schedules, i.e., schedules where precedence constraints among jobs are observed and the jobs are scheduled within their time windows. In the following we show that it is sufficient to examine active schedules (see the definition below) for the SgCSP and we reformulate our model accordingly.

We define a labeled directed graph $G = (J, A)$ with vertex set J corresponding to the set of jobs. Two labels for each vertex $j \in J$ describe release time r_j and deadline d_j. Arc set $A = \{(i, j) | i \, may \, precede \, j, \; i, j \in J\}$ describes all possible job pairs, in particular, $E \subseteq A$. Let $P = (j_1, j_2, ..., j_m)$ be a *path* in G that includes any vertex of G at most once. The makespan, i.e., the *earliest completion time* $M(P)$ of the jobs of path P is recursively defined through a simple forward calculation as follows:

$$
\begin{aligned}
&M(j_1) := r_{j_1} \\
&M(j_1, ..., j_k) := \max\{r_{jk}, M(j_1, ..., j_{k-1}) + s_{j_{k-1}, j_k}\} \, for \, k \in \{2, ..., m\}.
\end{aligned}
$$

A feasible *schedule* of jobs, i.e. a schedule that does not violate any precedence constraints and where each job is scheduled within a time interval defined by its release time and deadline, is *active*, if each job in the sequence finishes at its earliest possible completion time, i.e. no job can start earlier without delaying another one. For instance, the schedule in Figure 3.2 is active.

Observation 3.2.1. If an instance of the SgCSP has a feasible schedule then there exists an optimal schedule which is active.

Consequently, we can *restrict our attention to active schedules*. We call a path P *feasible* if it has an active schedule based on P.

3.2.2 On the single-crane scheduling problem with the objective to minimize total cost

In order to find improved bounds within our solution procedure we make use of a mixed-integer formulation of the single-crane scheduling problem with a modified objective. In this problem formulation, we will omit hard-to-deal-with waiting times from the objective function and will consider just the total sum of the incurred setup times (or total "cost").

We denote this problem as *SgCSP-Mincost* and a path defining an optimal solution as $\widetilde{P^*}$. The *cost* $C(P)$ of path $P = (j_1, j_2, ..., j_m)$ is the sum of its setup times, i.e. $C(P) = \sum_{k=2}^{m} s_{j_{k-1},j_k}$. Additionally, we introduce a set $\Theta(J, A, GUB)$ of *prohibited* paths. A path P' is prohibited if one of the following is true:

- it violates time window or precedence constraints,
- or if the *makespan* $M(P')$ of P' is equal to or higher than the current *global upper bound (GUB)*, i.e. the makespan of the currently best solution of the SgCSP.

In the following, we refer to the set of prohibited paths simply as Θ for simplicity.

Given GUB, the SgCSP-Mincost(J, A, GUB) is the problem to find a Hamiltonian path $\widetilde{P^*} \notin \Theta$ in G with minimal cost $C(\widetilde{P^*})$.

Let us denote the set of ingoing arcs incident to vertex i as $\delta^-(i) = \{(j,i) \in A | j \in J\}$ and the set of outgoing arcs as $\delta^+(i) = \{(i,j) \in A | j \in J\}$. Binary decision variables x_{ij} define whether job i is performed immediately before job j ($x_{ij} = 1$) or not ($x_{ij} = 0$). We denote the sum of decision variables corresponding to the arcs of set A' by $x(A') = \sum_{(i,j)\in A'} x_{ij}$. Thus for a path P it is $x(P) = \sum_{k=2}^{m} x_{j_{k-1},j_k}$. Let $A(K)$ be the set of all arcs connecting any two vertices from a set $K \subseteq J$.

We adapt the model proposed by Ascheuer et al. (2001) and formulate a mixed-integer model for the SgCSP-Mincost as follows:

$$\min \quad \sum_{i\in J\setminus\{z\}} \sum_{j\in J\setminus\{a\}} s_{ij} \cdot x_{ij} \tag{3.1}$$

$$s.t. \quad x(\delta^+(i)) = 1 \qquad \forall\, i \in J \setminus \{z\} \tag{3.2}$$

$$x(\delta^-(i)) = 1 \qquad \forall\, i \in J \setminus \{a\} \tag{3.3}$$

$$x(A(K)) \leq |K| - 1 \qquad \forall\, K \subset J,\ 2 < |K| < n \tag{3.4}$$

$$x(P) \leq m - 2 \qquad \forall\, path\, P = (j_1, j_2, ..., j_m),\ P \in \Theta \tag{3.5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in J \tag{3.6}$$

Constraints (3.2) and (3.3) require that each job has exactly one successor and exactly one predecessor job. Subtours are excluded by constraints (3.4). The paths elimination constraints (3.5) eliminate all prohibited paths. The number of constraints of type (3.4) and (3.5) can be exponential in n.

Notice that any feasible solution of SgCSP-Mincost(J, A, GUB) is also a feasible solution for the original problem SgCSP(J, A), therefore $M(P^*(J, A)) \leq M(\widetilde{P^*}(J, A, GUB))$. Moreover, the optimal objective value of problem $SgCSP-Mincost(J, A, GUB)$ provides a lower bound to the corresponding objective value of problem $SgCSP(J, A)$. Therefore the following observation holds.

Observation 3.2.2. $C(\widetilde{P^*}(J, A, GUB)) \leq M(P^*(J, A)) \leq M(\widetilde{P^*}(J, A, GUB))$.

3.3 Dynamic branch-and-cut algorithm for the SgCSP

We apply the following *dynamic branch-and-cut (DBC)* procedure to the SgCSP: we iteratively solve the SgCSP-Mincost(J, A, GUB) problem and update GUB dynamically during the solution process. For instance, whenever we find a currently optimal solution $P := \widetilde{P^*}(J, A, GUB)$, we update the global upper bound as
$GUB := \min\{GUB, M(\widetilde{P^*}(J, A, GUB))\}$, path P enters set Θ of prohibited paths and we add a cut to exclude the solution. We proceed, until GUB equals a *global lower bound* (GLB), so that we find an optimal solution of the problem SgCSP(J, A) or the set of feasible solutions of SgCSP-Mincost(J, A, GUB) is empty.

Recall that the major idea of the branch-and-cut is to start with a relaxed model and to gradually strengthen this relaxation during the solution process by introducing valid cuts and enforcing integrality conditions on the decision variables. In the DBC, we extend this basic idea by using a non-restrictive initial value of an important parameter (GUB) and by "strengthening" this value during the solution process.

In Section 3.3.1, we provide an overview on the overall branch-and-cut procedure. Subsequently, in Section 3.3.2, we describe new separation routines designed for the SgCSP-Mincost problem.

3.3.1 Branch-and-cut solution procedure

We start the DBC with some preprocessing steps in order to narrow time windows and to reveal new implicitly existing precedence constraints imposed by the time windows. Additionally we compute a global lower bound GLB and a global upper bound GUB for the SgCSP(J, A). In the following, we explain the main idea. We refer an interested reader to Otto et al. (2017) for a detailed discussion.

We compute a lower bound as an optimal solution to the assignment problem (3.1)-(3.3), and continuous variables in (3.6). An alternative lower bound is obtained from an optimal solution of the single machine scheduling problem $1/r'_j/C_{max}$ with job release time r'_j and a makespan objective. The processing times p'_j and release times r'_j of the jobs of this single machine scheduling problem are either defined as $p'_j := \min_{i\in J|(i,j)\in A}\{s_{ij}\}$ and $r'_j := r_j - \min_{i\in J|(i,j)\in A}\{s_{ij}\}$ or as $p'_j := p_j + \min_{k\in J|(j,k)\in A}\{\vartheta_{jk}\}$ and $r'_j := r^o_j$.

We compute a global upper bound GUB for the SgCSP(J, A) through several heuristics, such as multiple start feasible nearest neighbor heuristic, priority rules as well as local search procedures with vertex reinsertion and two vertex exchange (see Otto et al., 2017, for details on preprocessing routines, GLB and GUB).

Starting with initial values for GLB and GUB, empty sets K and Θ in (3.4) and (3.5) as well as continuous variables in (3.6), we adapt cuts and separation routines in our branch-and-cut procedure as described by Ascheuer et al. (2001). We modify the separation routines for the *path elimination constraints* (PECs, see Section 3.3.2). The cuts include besides subtour elimination constraints (3.4), path elimination constraints for prohibited paths, inequalities for the sequential ordering problem, (π, σ)-inequalities to take precedence constraints into account, as well as inequalities for the one-machine scheduling problem with time windows and inequalities for the traveling salesman problem. We skip the details here and refer the interested reader to the paper of Ascheuer et al. (2001). We apply the cuts in the same order as in Ascheuer et al. (2001), but we include PECs immediately after the subtour elimination constraints in order to exclude long paths as early as possible.

In each node of the branch-and-cut tree, we consider one class of cuts at a time. We branch when we have included all the classes of cuts or when we have detected tailing-off. In each search tree node φ of the branch-and-cut procedure we consider a partial sequence of already scheduled jobs. Let job k (initially $k = a$) be the incumbent last job of the sequence in node φ. Binary branching in φ leads to two descendant nodes λ and μ where we either append a next job k' from the set of unscheduled jobs $J(\varphi)$ available in this node ($x_{k,k'} := 1$) or k' is not going to be the first job scheduled from the set $J(\varphi)$, i.e., $x_{k,k'} := 0$.

Job k' is excluded from the set of unscheduled jobs, i.e., $J(\lambda) := J(\varphi) \setminus \{k'\}$, and release times are updated as $r_i(\lambda) := \max\{C_{k'} + s_{k'i}, r_i(\varphi) | i \in J(\lambda)\}$ where $r_i(\alpha)$ is the release time of job i in search tree node α. These transformations allow reducing time windows and including implicitly existing precedence constraints by applying some simply preprocessing steps for the subproblem in node λ (for details see Otto et al., 2017). The search is repeated for the LP-relaxation of problem SgCSP-Mincost$(J(\lambda), A(J(\lambda)), GUB)$ in node λ.

Fixing variable $x_{k,k'}$ to 0 in the other node μ is rather weak and usually neither allows powerful preprocessing steps nor tight cuts. Therefore, we decided not to add any cuts.

We fathom a node when the lower bound exceeds the best upper bound or the set of feasible solutions is empty.

In order to improve efficiency of the branch-and-cut, we use the following simple *dominance rule.* Node λ dominates node η if $J(\lambda) = J(\eta)$ and $r_i(\lambda) \leq r_i(\eta)$ for all $i \in J(\lambda)$. In this case, the same sets of jobs have to be assigned in λ and in η, but the next unscheduled job may start earlier in node λ.

We perform the following variation of a breadth search. We always resume branching at a node with the smallest number of the already assigned jobs. Observe that this kind of branching eases application of the dominance rule. Among not yet assigned jobs in the incumbent node φ and given a possibly non-integer solution $x(\varphi)$ in this node, we heuristically select job k' as the job with $k' = \arg\max\{x_{k,i}(\varphi) | i \in J(\varphi),\ 0 < x_{k,i}(\varphi) < 1\}$ where k is the last job scheduled.

Each time we find a feasible *Hamiltonian* path in the branch-and-cut search, we compute the makespan of the corresponding active schedule and update GUB. A smaller GUB allows to eliminate more paths based on the path-elimination constraints (PECs) as described in Section 3.3.2. If a Hamiltonian path is optimal for the incumbent relaxed version of the SgCSP-Mincost problem, then we exclude this path by using a path-elimination constraint (3.A1) (see Appendix 3.A) in all active nodes of the branch-and-cut tree.

In the following, $[S]$ denotes a generic permutation of the jobs in set S, which does not violate precedence constraints. For example, $([S], j)$ describes a path, where a permutation of jobs in set S is immediately followed by job j. Further, $\{P\}$ denotes the set of all jobs of path P.

3.3.2 Extension of path elimination constraints

We extend the separation routine for path elimination constraints (PECs), originally used by Ascheuer et al. (2001), by introducing precedence constraints tests (see Section 3.3.2.1) and waiting time tests (see Section 3.3.2.2). The *precedence constraints tests* exclude infeasible paths violating precedence constraints. Notice that precedence constraints are also enforced by other classes of cuts with computationally more expensive separation routines (e.g., (π, σ)-inequalities), which are tackled following the separation for PECs during our branch-and-cut procedure. Therefore, our new precedence constraints tests may speed up the solution process, as will be demonstrated in our computational experiments in Section 5.2. The *waiting time tests* eliminate paths with large makespans.

To generate PECs, we consider all the paths $P = (j_1, j_2, ..., j_m)$, $m > 1$, satisfying condition $\sum_{k=2}^{m} \tilde{x}^*_{j_{k-1},j_k} > m-2$ for a (possibly non-integer) solution $\tilde{x}^*$ of the incumbent LP-relaxation of the SgCSP-Mincost problem (3.1)–(3.4), (3.6). Note that because of constraints (3.2) and (3.3), there are only polynomially many such paths. In contrast to Ascheuer et al. (2001), besides checking whether these paths violate time window constraints, we additionally apply our precedence constraints and waiting time tests. Detected infeasible or extra-long paths are eliminated through cuts. For convenience, we provide a list of PECs in the Appendix 3.A.

Observe that the logics of our tests in Sections 3.3.2.1-3.3.2.2 is somewhat related to the concept of the local-search domination of Bianco et al. (1988) and Jamal et al. (2017). Given some partial infeasible or extra-long path P, we identify and prohibit further similar infeasible or extra-long paths. In this way, we effectively prune whole sub-trees in the solution space.

3.3.2.1 Precedence constraints tests

Precedence constraints test 1. Consider an infeasible path $P = (j_1, j_2, ..., j_{m-1}, j_m)$ obtained during the search procedure where $(j_m, j_1) \in E$. Then, we eliminate all paths of the form $(j_1, [\{P\} \setminus \{j_1\}])$ where j_1 is the first job, by PEC-constraints (3.A4) (see Appendix 3.A) and all paths $([\{P\} \setminus \{j_m\}], j_m)$ where j_m is the last job, by PEC-constraints (3.A3) (see Appendix 3.A).

Precedence constraints test 2. Assume there is a job j_l to be scheduled between jobs j_1 and j_m, i.e., $(j_1, j_l) \in E$ and $(j_l, j_m) \in E$. Further assume j_l is not a part of the infeasible path $P = (j_1, j_2, ..., j_{m-1}, j_m)$. Then we exclude any path containing a consecutive permutation of jobs $\{P\}$,i.e., $[\{P\}]$ by constraint (3.A2) (see Appendix 3.A).

3.3.2.2 Waiting time tests

Let $w(P)$ be a lower bound on the waiting time of an active (partial) schedule, i.e., of any feasible path P in graph G. Additionally, let $\widetilde{LB}^0$ be a lower bound on the cost of feasible Hamiltonian paths in the current node φ of the branch-and-cut tree. We compute $\widetilde{LB}^0$ as the solution of the relaxed SgCSP-Mincost$(J(\varphi), A(\varphi), GUB)$ problem (see Section 3.3.1). We set lower bound $\widetilde{LB} := \widetilde{LB}^0 + C_k(\varphi)$, where $C_k(\varphi)$ is the completion time of the last job in the sequence of assigned jobs in node φ. If $\widetilde{LB} + w(P) \geq GUB$, we prohibit path P. Recall that $J(\varphi)$ does not include the assigned jobs (cf. section 3.3.1).

We develop waiting time tests for three overlapping classes of paths: paths that start with the source job (class 1), paths that end with the sink job (class 2) and the rest of the paths (class 3).

Observe that the waiting time in any active schedule between two consecutive jobs

j_k followed by j_l is at least $w_{j_kj_l}(C_{j_k}) := \max\{r_{j_l} - C_{j_k} - s_{j_k,j_l}, 0\}$ because j_k finishes at $C_{j_k} \leq d_{j_k}$ and j_l cannot start earlier than r_{j_l}.

Waiting time test 1. Consider a path $P = (a, j_1, j_2, ..., j_m)$ where the last job finishes at its release time, i.e. $C_{j_m} = r_{j_m} = M(P)$ and $w(P) = M(P) - C_k(\varphi) - C(P)$. If $\widetilde{LB} + w(P) \geq GUB$, we prohibit:

- any path $[\{P\}]$ starting in a, and which is a permutation of jobs of path P, by constraints (3.A2),
- paths of the form $[(\{P\} \cup \{j_k\}) \setminus \{j_m\}]$ by constraints (3.A2) for any $j_k \notin \{P\}$ with $w_{j_{m-1}j_k}(C_{j_{m-1}}) \geq w_{j_{m-1}j_m}(C_{j_{m-1}}) > 0$, where $C_{j_{m-1}}$ is the completion time of job j_{m-1} in the active schedule of path P.

The intuition behind the first statement is the following. Any path P' starting in a and consisting of only jobs from P has a makespan $M(P') \geq M(P)$ as job j_m cannot finish earlier than at its release time $r_{j_m} = M(P)$ possibly followed by additional jobs that are sequenced in P before job j_m. Let j_k be the last job in P' and assume H is a complete schedule consisting of the jobs of P' followed by a sequence P'' of jobs starting with j_k and all jobs from $J \setminus \{P'\}$. Then $M(H) \geq M(P') + C(P'') \geq M(P) + C(P'') = w(P) + C_k(\varphi) + C(P) + C(P'') \geq w(P) + \widetilde{LB} \geq GUB$.

As for the second statement consider the case where j_k replaces the last job j_m in P. $w_{j_{m-1}j_k}(C_{j_{m-1}}) > 0$ implies that job j_k is not available at time $(C_{j_{m-1}} + s_{j_{m-1}j_k})$ and therefore $C_{j_k} = r_{j_k}$. From $w_{j_{m-1}j_k}(C_{j_{m-1}}) \geq w_{j_{m-1}j_m}(C_{j_{m-1}})$ we conclude that $w(P') \geq w(P)$ where $P' = (a, j_2, j_3, ..., j_{m-1}, j_k)$ which immediately implies $\widetilde{LB} + w(P') \geq GUB$. Applying the first part of the waiting time test 1 proves the statement.

Waiting time test 2. Consider a path $P = (j_1, j_2, ..., z)$ from class 2. A possible lower bound on the waiting time is to set $w(P) := \max\{\widetilde{LB} - C(P) - d_{j_1}, 0\}$ because j_1 may finish as late as d_{j_1} and the local lower bound on the costs $\widetilde{LB}$ is also a local lower bound on the makespan. If $\widetilde{LB} + w(P) \geq GUB$, then we prohibit

- any path $[\{P\}]$ ending in z, and which is a permutation of jobs of path P, by constraints (3.A2),
- paths of the form $[(\{P\} \cup \{j_k\}) \setminus \{j_1\}]$ where $w_{j_kj_2}(d_{j_k}) \geq w_{j_1j_2}(d_{j_1}) > 0$ by constraints (3.A2).

The reasoning behind the first statement is the following. As $\widetilde{LB} < GUB$ (otherwise we would have fathomed the search tree node) path P has a positive waiting time, i.e. $\widetilde{LB} - C(P) > d_{j_1}$. Consider a feasible complete schedule where the jobs of path $[\{P\}]$

occupy the last positions. Assume the schedule consists of the jobs of a path P' ending with j_1 followed by a sequence P'' of jobs starting with j_1. As $C(P')+C(P) \geq \widetilde{LB}-C_k(\varphi)$, we get $d_{j_1} < \widetilde{LB} - C(P) \leq C(P') + C_k(\varphi)$ which contradicts that all jobs of P' finish until d_{j_1}.

As for the second statement let us consider the case where j_k replaces j_1 in P. Positive waiting time $w_{j_k j_2}(d_{j_k}) > 0$ implies that job j_2 is not available at time $(d_{j_k} + s_{j_k j_2})$ and therefore $C_{j_2} = r_{j_2}$. From $w_{j_k j_2}(d_{j_k}) \geq w_{j_1 j_2}(d_{j_1})$ we conclude that $w(P') \geq w(P)$ where $P' = (j_k, j_2, j_3, ..., j_{m-1}, z)$ which immediately implies $\widetilde{LB}+w(P') \geq GUB$. An application of the first part of waiting time test 2 proves the statement.

Waiting time test 3. Let $P = (j_1, j_2, ..., j_m)$ be a path of class 3. Observe that its first job cannot be scheduled later than its deadline, therefore we may consider the following waiting time lower bound $w(P) = \min\{M(P) - C(P) - d_{j_1}, 0\}$. If $\widetilde{LB} + w(P) \geq GUB$, then we prohibit

- path P by constraints (3.A1),
- paths P' received from P by replacing job j_1 with j_k such that $w_{j_k j_2}(d_{j_k}) \geq w_{j_1 j_2}(d_{j_1}) > 0$ by constraints (3.A1),
- paths P'' received from P by replacing job j_m with j_l such that $w_{j_{m-1} j_l}(C_{j_{m-1}}) \geq w_{j_{m-1} j_m}(C_{j_{m-1}}) > 0$ by constraints (3.A1), where $C_{j_{m-1}}$ is the completion time of job j_{m-1} in the active schedule of path $P \setminus \{j_m\}$, i.e. $C_{j_{m-1}} := M(j_1, ..., j_{m-1})$.

Let us consider the second statement. Since $w_{j_k j_2}(d_{j_k}) > 0$ and $w_{j_1 j_2}(d_{j_1}) > 0$, we receive that $r_{j_2} > d_{j_k} + s_{j_k j_2}$ and $r_{j_2} > d_{j_1} + s_{j_1 j_2}$. Therefore, in both schedules j_2 completes at its release time r_{j_2} and $M(P) = M(P')$. The validity of the second statement can be shown by observing that $M(P') - C(P') - d_{j_k} \geq M(P) - C(P) - d_{j_1}$.

In the third statement, observe that $M(P) = r_{j_m}$ and $M(P'') = r_{j_l}$ because $w_{j_{m-1} j_m}(C_{j_{m-1}}) > 0$ and $w_{j_{m-1} j_l}(C_{j_{m-1}}) > 0$. Therefore $M(P'') - C(P'') - d_1 = r_{j_l} + (r_{j_m} - r_{j_m}) + (C_{j_{m-1}} - C_{j_{m-1}}) - (C(P) - s_{j_{m-1} j_m} + s_{j_{m-1} j_l}) - d_1 = r_{j_m} + (w_{j_{m-1} j_l}(C_{j_{m-1}}) - w_{j_{m-1} j_m}(C_{j_{m-1}})) - C(P) - d_1 \geq M(P) - C(P) - d_1$. So that $w(P'') = \max\{M(P'') - C(P'') - d_1, 0\} \geq w(P)$ and we can prohibit path P''.

In our preliminary tests we observed that excluding paths using the waiting time tests made the solver generate slightly modified new paths where the waiting times are also too large. Therefore, we apply waiting tests 1, 2 and 3 first. As soon as we prohibit a path P we construct similar paths by job insertion and examine them with the waiting time test 4.

In the following, we define $w_{j_k j_l} := w_{j_k j_l}(d_{j_k})$ if path P belongs to classes 2 and 3 and

$w_{j_kj_l} = w_{j_kj_l}(C_{j_k})$ if path P belongs to class 1, where C_{j_k} is the completion time of job j_k in the active schedule of path P.

Waiting time test 4. Let jobs j_k and j_l be scheduled consecutively in some path P and $w_{j_kj_l} > 0$. If $\widetilde{LB} + w_{j_kj_l} - (s_{j_kj_c} + s_{j_cj_l} - s_{j_kj_l}) \geq GUB$ for some job $j_c \in J \setminus \{P\}$, then we prohibit path

- $P' = (..., j_k, j_c, j_l, ...)$ obtained from path P by inserting job j_c between j_k and j_l by constraint (3.A1).

3.4 Decomposition Algorithm

In the DA, we try to solve the original problem by decomposing it into easier to solve subproblems. Effectively, we try to partition the set of jobs J into two subsets J_1 and J_2 such that jobs J_1 precede jobs J_2 in an optimal solution. Observe that such introduction of additional precedence relations may favorably change the problem structure and significantly speed-up the solution process (cf. Klindworth et al., 2012; Montemanni et al., 2013).

Before initiating the DA we perform some preprocessing steps to tighten time windows and to introduce subsequently additional precedence constraints implicitly forced by the time windows as well as to compute a global upper bound GUB and a global lower bound GLB for problem SgCSP (cf. Section 3.3). If necessary we initialize $GUB := \infty$.

The initialization of the DA starts with the best feasible solution $P = (a, j_2, j_3, ..., j_{n-1}, z)$ generated during the preprocessing. We heuristically look for a decomposition of the job set J into $J^1 \cup J^2$, i.e., we assign a job j_k together with all the subsequent jobs to J^2 if:

- $r_{j_k} \leq r_{j_l}$, $k \leq l \leq n$ (j_k has the earliest release time among all the following jobs), and

- $C_{j_k} = r_{j_k}$ (j_k starts at its release time).

For the remaining jobs we assign $J^1 := J \setminus J^2$. If set J^2 is empty, we proceed solving the original problem SgCSP(J, A), e.g., using the DBC (see Section 3.3).

Otherwise, we start looking for a solution with the minimum makespan for jobs in J^2. We call the first job in this solution $j^c \in J^2$ as *connecting job* because it should immediately follow jobs from J^1 in our decomposition. If we can find a feasible schedule for jobs in J^1, so that for the last job of J^1, say j_k, satisfies $C_{j_k} \leq r_{j^c} - s_{j_kj^c}$, then the concatenation of both sequences gives us a best possible (and potentially optimal) solution with jobs from J^1 preceding jobs from J^2 and we stop the decomposition algorithm.

Otherwise, we complement the available job sequence for J^1 to a feasible solution for the SgCSP(J, A) (if such solution exists) with a minimum makespan and update the global upper bound GUB; we prohibit the examined job j^c to be a connecting job and proceed with a new iteration of the DA.

Observe that the decomposition algorithm will not find an optimal solution for the original problem SgCSP(J, A), if no optimal solution of the SgCSP(J, A) with jobs J^1 preceding jobs J^2 exists. Even in such a case a DA-based improved upper bound GUB often reduces the solution time significantly, as we will show in our computational experiments. Observe that the decomposition algorithm may also improve our initial global lower bound because the makespan of an optimal schedule for J^2 provides a lower bound for problem SgCSP(J, A).

We refer to the decomposition algorithm that uses the DBC to solve the resulting SgCSP problems as DBC&DA.

3.5 Computational experiments

Our computational experiments were run on a laptop with Intel i5-3450 3.1GHz CPU. The algorithms are implemented in C++, utilizing the library of IBM ILOG CPLEX 12.7. Next we describe our data sets in Section 3.5.1. We report on the results of our simulation of single-crane transshipment yards in Section 3.5.2. Section 3.5.3 presents the performance of the algorithms and their constituent elements on various test sets in more detail, including well-established benchmark data sets.

3.5.1 Data sets

We conducted computational experiments on three data sets, which we call for brevity $DS1$, $DS2$ and $DS3$. $DS1$, which contains 90 instances, is based on a careful simulation of single-crane transshipment yards. $DS2$ is the benchmark data set of Ascheuer (1996), which we downloaded from http://ftp.zib.de/pub/mp-testdata/tsp/atsptw/index.html. It contains 50 instances describing scheduling of a stacker crane in an automatic storage system, the instances are based on real data from practice. $DS3$ is an artificially generated well-established benchmark data set of Dumas et al. (1995) and contains 135 instances.

Instances of $DS1$ simulate a 700-meter-long single-crane transshipment terminal consisting of two tracks and of $n = 15$, 20 and 25 jobs with *wide* time windows. We generate instances in nine settings with different numbers of jobs and of (nontrivial) time windows. With 10 instances per setting, $DS1$ contains $9 \cdot 10 = 90$ instances in total. For brevity, we only sketch the most important instance generation parameters and refer the interested reader to Otto et al. (2017) for further details. We compute setup times and processing times based on typical horizontal and vertical velocities of a gantry crane, thereby we differentiate between velocities of a loaded and of an unloaded crane. Observe that precedence constraints arise when pickup and drop-off positions of containers are identical (e.g. job 3 and job 2 in Figure 3.1). We generate time windows for rail-road jobs, which account for about 65% of all jobs (cf. Rotter, 2004), to reflect a service promise for customer trucks. For instances with $n = 20$ jobs, we consider settings with 0, 2, 4, 6, 8, 10 and 13 (i.e. for all rail-road jobs) time windows. For instances with $n = 15$ and $n = 25$, we only look at settings where time windows are generated for all rail-road jobs. In order to see the influence of the time windows on the job sequence we report a ratio between the average time window length and the average setup time s_{ij} (not including dummy jobs). For $DS1$ this ratio is $\frac{180}{1}$, in other words time windows are wide and do not significantly restrict job positions.

$DS2$ is a well-established data set of Ascheuer (1996) originally describing instances of an asymmetric travelling salesman problem with time windows containing up to 233 jobs. In the original data set, Ascheuer (1996) sets the release time of the dummy sink job to a large number as part of the preprocessing procedure. Therefore, we change the release time of the dummy sink jobs to 0 in order to adapt the instances of Ascheuer (1996) to a makespan objective function of the SgCSP. The time windows in $DS2$ are rather wide as well. The ratio between the average size of the time windows and the average setup time varies significantly between $\frac{16}{1}$ and $\frac{291}{1}$. It is about $\frac{82}{1}$ on average.

$DS3$ describes the data set of Dumas et al. (1995), which is a well-established benchmark data set for the traveling salesman problem with time windows. In $DS3$, there are no precedence constraints, setup times are symmetric ($s_{ij} = s_{ji} \, \forall \, i, j \in J$) and time windows are narrow. The ratio between the average size of the time windows and the average setup times s_{ij} is about $\frac{2}{1}$ for the instance settings with the largest time windows. $DS3$ contains 135 instances spread over 27 settings with $n = 20$, 40, 60, 80, 100, 150 and 200 jobs. There are up to 5 time window settings for each size. Each setting specifies an interval from which the release times and the deadlines of jobs are randomly drawn. For small instances with $n \leq 60$, the data set of Dumas et al. (1995) contains five settings of time windows. For larger instances, it contains two to four settings of time windows with smaller intervals.

In our experiments, we solved all the instances of $DS1$, $DS2$ and $DS3$ to optimality.

We report the performance of our decomposition algorithm as a part of two solution procedures: the DBC&DA and the DP&DA. The DP&DA uses the dynamic programming algorithm (DP) of Dumas et al. (1995) to solve (sub-)problems of the SgCSP. DP was originally developed for the traveling salesman problem with time windows (TSPTW) with a total cost minimizing objective. It is one of the most successful currently available algorithms for the TSPTW, which can straightforwardly be extended to deal with a makespan objective function and precedence constraints.

In all the tests, we set a time limit to 600 seconds per instance. For the two-stage algorithms DBC&DA and DP&DA, we set a time limit to 600 seconds both for the first part and for the overall algorithm, including the first part. Recall that the first part examines all the solutions with a heuristically determined set of jobs $J^2 \neq \emptyset$ succeeding the set of jobs $J^1 = J \setminus J^2$. The second part solves the original SgCSP(J, A), if the first part of the algorithm could not find an optimal solution.

3.5.2 Simulation results

Table 3.2 summarizes our results on the instances of $DS1$. Both algorithms, the DBC&DA and the DP&DA solved all the instances of $DS1$ to optimality with an average run time of about 1.5 and 3.0 seconds, respectively. The run times of the DBC&DA and the DP&DA are generally lower if a decomposition is possible (i.e. if the set of jobs J^2 is not empty, cf. Section 3.4). This is the case for 67 out of 90 instances. The computational time of the DBC&DA does not exceed 63 seconds per instance and the computational time of the DP&DA does not exceed 91 seconds per instance. The DBC&DA usually dominates DP&DA for instances with only a small number of time windows.

Insight 1: We conclude that the proposed methods DBC&DA and DP&DA successfully solve instances of practically relevant size.

Planners often use algorithms that minimize the total cost even if they are actually interested in the makespan objective. In other words, they expect a waiting time of 0 in a solution with minimal cost. Table 3.3 illustrates that such an approximation results in about 30-40% increase in the makespan at transshipment terminals.

Insight 2: Solving a total cost minimization problem (and neglecting the makespan) may lead to a significant increase in the makespan.

The ability to warrant a service promise, i.e. a service time window, is one of the important instruments to attract customers and additional order volumes. However, the more jobs with time windows exist and the tighter time windows are, the larger is the makespan. Figure 3.3 illustrates that the makespan increases from about 33 minutes in case of no time windows to about 44 minutes (by 30%) in case all the rail-road jobs have time windows.

Observe that characteristics of time windows may be very different at different transshipment yards, and such an analysis should be performed with the data of the transship-

Table 3.2: Performance of the DP&DA, the DBC&DA and the DP for $DS1$ with time limit of 600 seconds

Setting, No. of jobs (No. of TW)*	Solution method	Avg. computational time (sec)	No. of instances decomposed with the DA
15 (all)	**DP&DA**	0.0	10 out of 10
	DBC&DA	0.0	
	DP	0.0	
20 (all)	**DP&DA**	0.0	10 out of 10
	DBC&DA	10.7	
	DP	1.1	
20 (10)	**DP&DA**	0.0	10 out of 10
	DBC&DA	0.0	
	DP	2.4	
20 (8)	**DP&DA**	0.0	10 out of 10
	DBC&DA	0.0	
	DP	4.4	
20 (6)	**DP&DA**	0.5	9 out of 10
	DBC&DA	0.9	
	DP	16.1	
20 (4)	DP&DA	7.1	5 out of 10
	DBC&DA	0.4	
	DP	12.9	
20 (2)	DP&DA	16.9	3 out of 10
	DBC&DA	1.1	
	DP	34.6	
20 (0)	DP&DA	2.1	0 out of 10
	DBC&DA	0.1	
	DP	44.0	
25 (all)	**DP&DA**	0.0	10 out of 10
	DBC&DA	0.0	
	DP	71.8	

Best algorithms in each setting are marked in bold

* No. of rail-road jobs with nontrivial time windows (TW)

Table 3.3: Differences in the makespan for the optimization with different objective functions

Setting, no. of jobs (no. of TW)*	Avg. makespan for solutions with the minimal cost** (A)	Avg. minimal makespan (B)	Avg. relative deviation $(Ai)/(Bi)$*** (%)
15 (all)	2550.3	1944.3	132
20 (all)	3445.3	2619.9	132
25 (all)	4387.8	3191.0	138

* No. of rail-road jobs with nontrivial time windows (TW)

** Average makespan of optimal solutions found by minimizing the total cost

*** $(Ai)/(Bi)$ is computed for each instance i, then the average is taken

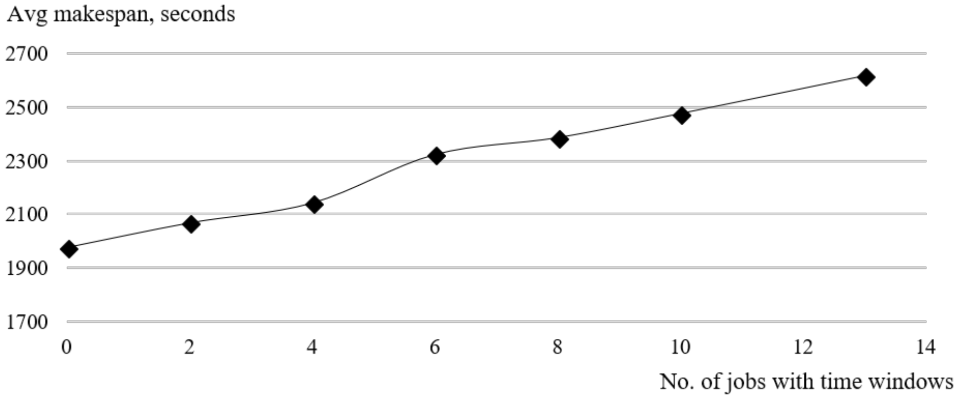


Figure 3.3: Relation between the average optimal makespan and the number of rail-road jobs with time windows

ment yard of interest. Nevertheless, this piece of analysis illustrates that our algorithms have no problem solving instances with only a few time windows and are perfectly suited for such an analysis.

Insight 3: Service promises to customer trucks lead to a significant increase in the makespan.

3.5.3 Detailed performance analysis of the algorithms

In this section, we discuss the performance of the proposed algorithms DBC&DA and DP&DA in more detail. We compare them to the dynamic programming algorithm (DP) of Dumas et al. (1995) for well-established benchmark data sets $DS2$ and $DS3$. Afterwards, we examine the performance of DA and DBC in more detail.

Comparative performance of the DBC&DA, the DP&DA and the DP on benchmark data sets $DS2$ *and* $DS3$. Table 3.4 illustrates that the DBC&DA and the DP&DA outperform DP on $DS2$, which is the data set of Ascheuer (1996). Interestingly, the DBC&DA was able to solve instances that were not solved by the DP and the DP was able to solve instances that were not solved by the DBC&DA. Overall, there is no dominance relation between the results of the DBC&DA and the DP&DA. For example, the DBC&DA outperforms the DP&DA on eight instances.

Whereas the DP&DA outperforms the DP on $DS3$, the DBC&DA shows an inferior performance. The DBC&DA seems to have more difficulties for $DS3$ instances with larger time windows. It is rather surprising, because the DBC&DA performs particularly well on $DS1$ and $DS2$ with large time windows. Interestingly, the DBC&DA is extremely fast on the solved instances, including those with 200 jobs, and solves 81% of instances in less than 0.5 second each. On the other hand, the run time of the DP depends more strongly

Table 3.4: Comparative performance of DBC&DA, DP&DA and DP on $DS2$ and $DS3$

Data set	No. of instances solved to optimality (total no. of instances)			Avg. run time (sec.)*			No. of instances decomposed with the DA
	DBC&DA	DP&DA	DP	DBC&DA	DP&DA	DP	
$DS2$	47 (50)	50 (50)	44 (50)	11.5	1.8	41.8	46
$DS3$	116 (135)	135 (135)	135 (135)	4.4	4.4	10.7	106

* for solved instances

on instance sizes, which also points on the success of our decomposition algorithm. For example, the DP needs at least 9 seconds to solve instances with 200 jobs.

Overall, although the DBC&DA was not able to solve 22 out of 185 instances of $DS2$ and $DS3$ within the run time limit, in 16 cases it found an optimal solution and in the remaining 6 cases the average relative deviation from optimality equaled just 0.6%.

Performance of the DA. Results of the DP&DA and the DP in Tables 3.2 and 3.4 confirm that the decomposition algorithm significantly reduces run times on all the tested data sets. Notably, it reduces the average run time from 71.8 to less than 0.05 seconds for the instances of $DS1$ with $n = 25$ jobs.

Performance of the specific algorithmic elements of the DBC. Table 3.2 illustrates that the DBC has a relative advantage for instances with a small number of time windows. In the following piece of analysis (see Tables 3.5 and 3.6), we examine effectiveness of specific elements of our DBC procedure on the benchmark data set $DS3$. Table 3.5 gives some insight on the average number of search tree nodes when the dominance rule is used to eliminate dominated nodes early in the search process (cf. Section 3.3.1). Overall, the dominance rule prunes 15.8% of nodes per instance on average. Table 3.6 evaluates performance of the new tests in the separation routine for the path elimination constraints (see, cf. Section 3.3.2). It reports the average number of cuts per instance and the average percentage of the total run time needed on the separation routines. Notice that besides the separation routines there are the LP-relaxations, branching and preprocessing that require some time. From Table 3.6 we see that the proposed waiting time tests and precedence constraints tests supply about 28% of cuts per instance on average. The proposed waiting time and precedence constraints tests are also very fast to compute.

Table 3.5: Performance of the dominance rule of the DBC on $DS3$

Avg no. of nodes per instance	Avg no. of nodes pruned with the dominance rule	Avg share of nodes pruned with the dominance rule (%)
2708	525	15.8

Table 3.6: Performance of cuts in the DBC for $DS2$

No. of jobs	Path-elimination constraints (PECs)* based on				Precedence-related non-PEC cuts**		Other cuts	
	Precedence constraints tests, no. of cuts	Waiting time tests, no. of cuts	Other PECs, no. of cuts	CPU (%)	No. of cuts	CPU (%)	No. of cuts	CPU (%)
$n = 20$	88	443	1050	1.7	33	17.3	144	17.7
$n = 40$	686	1316	4819	1.0	211	23.3	655	19.4
$n = 60$	1284	827	7918	0.8	246	31.5	983	15.7
$n = 80$	1646.4	431	6628	0.7	127	32.1	890	18.7
$n = 100$	2100	755	3806	0.6	77	35.7	749	17.2
$n = 150$	1831	575	1080	0.4	36	52.1	539	12.6
$n = 200$	1102	107	240	0.2	12	57.4	518	8.6

The column "CPU" reports the average percentage of the total run time needed on the separation routines.

* Precedence constraints and waiting time tests are described in Sections 3.3.2.1 and 3.3.2.2, respectively. Other tests check, whether paths violate time window constraints.

** π-, σ- inequalities as well as so-called precedence cycle breaking and special inequalities, see Ascheuer et al. (2001) for the details on the cuts and separation routines.

3.6 Conclusion

In this paper, we propose a decomposition algorithm (DA) and a solution procedure DBC&DA for the single-crane scheduling problem with the objective to minimize the makespan. To our best knowledge, the DBC&DA is the first branch-and-cut-based algorithm for the single-crane scheduling problem, moreover it is the first branch-and-cut algorithm for the asymmetric travelling salesman problem with a makespan objective function (called as the minimum completion time problem in the literature). The DA decomposes problem instances into smaller subproblems, which are faster to solve. In its essence, the DA is a metastructure that can be integrated with many other exact and heuristic algorithms for the single-crane scheduling problem.

In our experiments, DA significantly reduces the run time by a larger factor for almost all the examined instances. Applied either with our dynamic branch-and-cut procedure or with a dynamic programming procedure of Dumas et al. (1995), it is able to solve all the instances in our simulations of transshipment yards within short run times that practitioners expect.

Overall, the proposed branch-and-cut-based algorithm DBC&DA has a comparable performance to the DP&DA for the tested instances. According to our computational experiments, the DBC&DA has a relative advantage for instance with a small number of time windows. The developed separation routines for path elimination constraints, waiting time tests and precedence relations tests, supply a majority of cuts and are fast to compute.

Our simulations of transshipment yards indicate that the makespan objective cannot be approximated by a cost minimization objective and that introducing service promises for all the customer trucks leads to a significant increase in the makespan.

For future research, we suggest to consider hybrid algorithms, which combine advantages of dynamic programming or branch-and-bound algorithms with those of the dynamic branch-and-cut. In the next steps, the developed solution approaches should be extended to a more general class of the minimum tour duration problems.

3.A Appendix A

In the following, we summarize different types of the path elimination constraints.

We exclude an infeasible path $P = (j_1, ..., j_m)$ using the following constraint:

$$x(P) := \sum_{k=1}^{m-1} \sum_{l=k+1}^{m} x_{j_k j_l} \leq m - 2 \tag{3.A1}$$

If a set of jobs $j_1, ..., j_m$ admits no feasible path, then we introduce the following cut:

$$x(A(\{j_1, ..., j_m\})) := \sum_{k=1}^{m} \sum_{l=1}^{m} x_{j_k j_l} \leq m - 2 \tag{3.A2}$$

If a path of the type (Q, j_m) is infeasible, where Q is some permutation of the job set $\{j_1, ..., j_{m-1}\}$, then we introduce the following constraint:

$$x(A(Q)) + x(Q : j_m) := \sum_{k=1}^{m-1} \sum_{l=1}^{m-1} x_{j_k j_l} + \sum_{k=1}^{m-1} x_{j_k j_m} \leq m - 2 \tag{3.A3}$$

Similarly, if any path of the type (j_1, Q) is infeasible, where Q is some permutation of the job set $\{j_2, ..., j_m\}$, then a constraint is specified as follows:

$$x(j_1 : Q) + x(A(Q)) := \sum_{k=2}^{m} x_{j_1 j_k} + \sum_{k=2}^{m} \sum_{l=2}^{m} x_{j_k j_l} \leq m - 2 \tag{3.A4}$$

Finally, if any path of the type (j_1, Q, j_m) is infeasible, where Q is some permutation of the job set $\{j_2, ..., j_{m-1}\}$, then we include the following cut:

$$x(j_1 : Q) + x(A(Q)) + x(Q : j_m) := \sum_{k=2}^{m-1} x_{j_1 j_k} + \sum_{k=2}^{m-1} \sum_{l=2}^{m-1} x_{j_k j_l} + \sum_{k=2}^{m-1} x_{j_k j_m} \leq m - 2 \qquad (3.A5)$$

Chapter 4

Product sequencing in multiple-piece-flow assembly lines

Modern markets demand mass customization, that is, the manufacture of customized products at low cost. Mass customization represents a major challenge for the organization of assembly lines, which were originally designed for the manufacture of homogeneous products. The multiple-piece-flow assembly line is an organizational innovation that can address this challenge. Here, several customized workpieces, each associated with a separate customer order and, hence, a separate due date, are handled simultaneously in one cycle. Consequently, the idle times decrease as do the manufacturing costs. Multiple-piece-flow assembly lines are used, for instance, in manufacturing industrial equipment.

To the best of our knowledge, this paper is the first to investigate product sequencing in multiple-piece-flow assembly lines. We formalize the underlying planning problem, establish a mixed-integer model, examine its relation to several classic optimization problems, and describe useful problem properties. We leverage these properties to design an effective iterative variable neighborhood heuristic (IVNH). A detailed simulation based on real-world data and the rolling-horizon planning framework confirms that the IVNH is well suited for practical use. Furthermore, extensive computational experiments on well-structured randomly generated data sets show that the IVNH identifies optimal or near-optimal solutions within short run times. It outperformed an off-the-shelf optimization software, and in certain practice settings, the IVNH was even able to substantially reduce average order delays.

4.1 Introduction

In the coming decades, manufacturing organization is likely to completely change owing to technological innovations and increasingly sophisticated demand. Companies struggle to offer *mass customization* to the clients, that is, the manufacture of customized products at low cost (Davis, 1990; Pine, 1993). Since the assembly line was originally designed for manufacturing homogeneous products, mass customization is particularly challenging for this organizational form. Companies are seeking to re-organize the manufacturing process to secure advantages of traditional assembly lines, while also being able to produce customized products at low cost.

In traditional paced assembly lines, conveying technology moves products (workpieces) through sequentially arranged stations. At each station, workers or robots process the workpiece for a specified amount of time (i.e., *cycle time*), after which the workpiece is transported to the next station. Paced assembly lines offer several advantages (for more discussion see (Buxey et al., 1973; Nof et al., 1997)). For instance, manufacturing times can be drastically reduced because of specialization and learning effects (Jaber, 2016; Otto and Otto, 2014a). Furthermore, paced production increases the transparency of production processes for management and control. If products are not homogeneous, however, large idle times may emerge. Indeed, the processing time of a bottleneck workpiece at a bottleneck station dictates the cycle time. In addition, the processing times of customized products may be very different (Bukchin, 1998; Rekiek and Delchambre, 2006). Therefore, in the manufacture of customized products, the upsurge in cost due to large idle times may offset the cost reduction expected from pacing.

Multiple-piece-flow assembly lines may secure the benefits of paced production and, by having several workpieces bundled together, keep idle times low. In *multiple-piece-flow assembly lines*, a set of workpieces moves together, which enables processing several workpieces at one station in one cycle. Multiple-piece-flow assembly lines are used in the production of customized industrial equipment.

Multiple-piece-flow assembly lines are among several organizational forms that adapt paced assembly lines to the requirements of customized manufacturing. Other organizational forms include, for instance, *(i)* deploying multi-skilled workers, called *floaters*, that can assist at bottleneck stations (e.g., (Faccio et al., 2016; Mayrhofer et al., 2013)), *(ii)* increasing the cycle time and assigning several workers to (especially bottleneck) stations (e.g., (Dolgui et al., 2006; Lapierre et al., 2006; Simaria et al., 2009)), as well as *(iii)* establishing open-end stations, where workers can cross station borders to keep processing a bottleneck workpiece (cf. mixed-model assembly lines with open-end stations in (Bock et al., 2006; Sarker and Pan, 2001)). However, it may be difficult to ensure sufficient

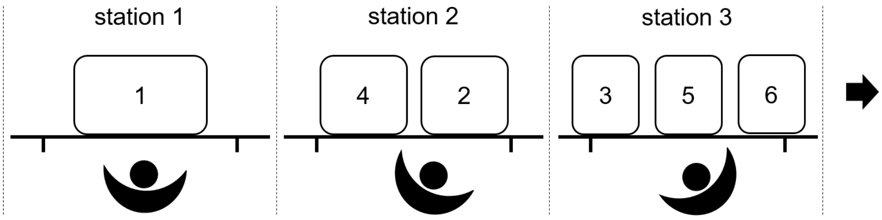


Figure 4.1: Illustrative example of a multiple-piece-flow assembly line. Workpieces are transported on mobile platforms between several stations in a series. One or more workpieces may enter the assembly line at the beginning of each cycle. The line is pictured after the launch of the third mobile platform.

utilization of the expensive multi-skilled floaters that would keep production costs low. Multi-manned stations may also be impossible because several workers cannot work on certain workpieces without severely hindering each other. In addition, open-end stations may pose hard-to-solve organizational issues of worker coordination and equipment accessibility. Therefore, the multiple-piece-flow assembly line is the first-choice organizational form, or at least the organizational form of interest, for many different companies, based on information that we gathered at cooperation meetings with firms and conferences.

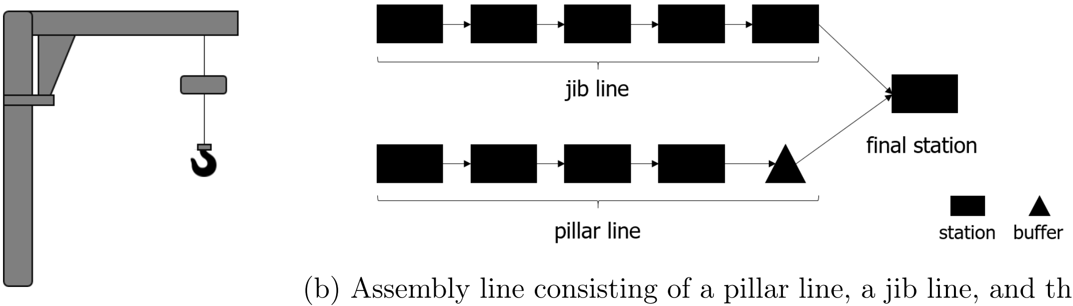
To the best of our knowledge, multiple-piece-flow assembly lines have not yet been discussed in the literature. As a result, the planners and industrial engineers lack evidence-based methodological support in organizing the production process. As a first step in closing this gap, we formulate and study the product sequencing problem in multiple-piece-flow assembly lines in the current paper.

We briefly outline specific examples of multiple-piece-flow assembly lines in Section 4.1.1 and formulate the respective optimization problem in Section 4.1.2. Section 4.1.3 provides an overview of the literature and Section 4.1.4 explains the article's contribution.

4.1.1 Examples of multiple-piece-flow assembly lines

In this section, we provide two real-world examples of multiple-piece-flow assembly lines describing two medium-sized companies in separate industries and from different federal states in Germany. Both companies produce customized industrial equipment. Based on interviews with representatives from these and other companies who highlight the advantages of the multiple-piece flow, we suggest that many more companies could benefit from using this alternative for manufacturing customized products.

The first example describes the final assembly of soldering machines shown in Figure 4.1. Soldering machines are used in the electronics industry to automatically fix components on circuit boards. Depending on customer requirements, the manufacturing times (and size) of soldering machines may vary significantly. In the final assembly at the company we visited, soldering machines in-process are transported on mobile platforms between different stations. The final assembly is organized as a paced line, where each platform carries one or more workpieces depending on their size and the extent of the manufacturing process required. At each station all the workpieces on one platform have to be processed within the set cycle time. Afterward, the platforms move one station forward.



(a) Industrial crane

(b) Assembly line consisting of a pillar line, a jib line, and the final processing station

Figure 4.2: Schematic illustration of the assembly system for industrial cranes

The second example describes manufacture of industrial jib cranes, which are used in industrial settings, such as warehouses or factory workshops (see Figure 4.2). Each workpiece has to be processed at several stations, and each station usually has a machine and a worker to service it. The stations are organized in two lines: a jib line consisting of five stations as well as a pillar line consisting of four stations and a buffer to compensate for different lengths of the two lines. Jibs and pillars are joined together and finally processed in the final station. The production is organized in batches. Each batch (i.e., bundle of workpieces) enters a station at the beginning of the day and all workpieces of that bundle have to be processed by the end of the day. At the beginning of the next day, this bundle of workpieces enters the next station. We discuss this example in more detail in our simulation study in Section 4.4.3.

4.1.2 Problem definition

To describe the product sequencing decisions in multiple-piece-flow assembly lines, we formulate the *m-vector bin packing and sequencing problem* (m-BiPacS) as follows. Workpieces $j \in J$ are launched in sets on the paced assembly line to be processed at sequentially arranged stations. Once launched, the composition of a set of workpieces remains the same, i.e., all the workpieces in a set enter and leave each station simultaneously. A set of workpieces is available for a certain amount of time at each station, called *cycle time c*. After c time units are over, the workpieces are moved to the next station and a new set of workpieces is launched at station 1. We denote the set of possible launch times as $\{1, \dots, \bar{B}\}$, where $\bar{B}$ is the number of potential launch times. Because each workpiece $j \in J$ corresponds to a specific customer order, it has a specific *due date* d_j, which is the latest possible launch time that allows the workpiece to be finished in time, and a specific *release time* r_j, which is the earliest possible launch time when the specification of the product order is fixed and the required material supply is secured. We denote the set of *dimensions* as $\{1, \dots, m\}$ and the *size* of workpiece $j \in J$ along dimension $i \in \{1, \dots, m\}$ as v_{ji}. Along each dimension, the total size of the workpieces launched together in a set is restricted. For example, each station represents a separate dimension because at each station the total processing time (=the total size) of workpieces that are launched together cannot exceed the cycle time. In some applications, the size of the mobile platform represents an additional dimension, because the total size of the workpieces that are launched together cannot be larger than the mobile platform. We also introduce cost parameters w_{jb} if workpiece j is launched at time $b \in \{1, \dots, \bar{B}\}$. For example, if the workpiece is launched after its due date d_j some lateness penalties may be incurred. In this paper, we examine *m-BiPacS* with *non-decreasing costs* in the ordering number of the launch times. That is, for any two launch times $b' > b$, the assignment cost to the second launch time is not lower than the assignment cost to the first launch time: $w_{jb'} \geq w_{jb} \; \forall j \in J$.

The objective of the m-BiPacS is to find mapping $x : J \to \{1, \dots, \bar{B}\}$ of workpieces to their launch times such that

- Each workpiece is launched exactly once and not earlier than its release time: $x(j) \geq r_j \; \forall j \in J$.
- Total size of the workpieces along each dimension does not exceed the cycle time: $\sum_{j \in J : x(j) = b} v_{ji} \leq c \; \forall b \in \{1, \dots, \bar{B}\}, \forall i \in \{1, \dots, m\}$.
- The objective function $F(\mathbf{x}) = \sum_{j \in J} w_{j,x(j)}$ is minimized.

Note that we use bold type, e.g., $\mathbf{x}$, to denote vectors or matrices.

Observe that the formulated objective function is quite general and can be used, for example, to minimize the weighted tardiness (by setting $w_{jb} := w_j \cdot \max\{b - d_j; 0\}$ where w_j is the *weight* of workpiece j), weighted lateness (by setting $w_{jb} := w_j \cdot (b - d_j)$), and average completion time (by setting $d_j := 0$, $w_{jb} := b$). Also observe that the m-BiPacS does not prohibit the launch of workpieces after their due date, but it penalizes late completions in the objective function.

Consider an illustrative example with the parameters in Table 4.1 and a multiple-piece-flow assembly line as shown in Figure **??**. Note that the workpieces that are launched together cannot exceed the size of the mobile platform. We can formulate the respective product sequencing problem as the m-BiPacS. In this formulation, the set $\{1, \dots, m\}$

Table 4.1: Illustrative example:
A m-BiPacS-M instance with six jobs, four dimensions $m := 4$, $c := 6$, $r_j := 0 \; \forall j \in J$,
$\bar{B} := 4$ possible launch times, and total tardiness objective function with $w_{jb} := \max\{b - d_j; 0\}$

	Workpieces (jobs)					
	1	2	3	4	5	6
Processing time at station 1, v_{j1}	5	3	2	3	2	2
Processing time at station 2, v_{j2}	6	4	1	2	3	2
Processing time at station 3, v_{j3}	4	2	3	4	2	1
Size of the workpiece, v_{j4}	3	2	1	2	1	1
Due date d_j	3	2	2	1	1	1

	Cost parameters w_{jb}					
	1	2	3	4	5	6
$b = 1$	0	0	0	0	0	0
$b = 2$	0	0	0	1	1	1
$b = 3$	0	1	1	2	2	2
$b = 4$	1	2	2	3	3	3

describes three assembly stations and an additional dimension, the size of the platform. So that the number of dimensions equals $m := 4$. Parameters v_{j1}, v_{j2}, and v_{j3} denote processing times of workpiece j and parameters v_{j4} denote the length of soldering machine j. Observe that we have normalized v_{j4} in order to set the size of the platform to be equal to 6 as the cycle time. Figure **??** illustrates a feasible solution for the instance in Table 4.1 pictured after the launch of the third mobile platform, where workpieces 3, 5, and 6 are launched at time $b = 1$, workpieces 2 and 4 at launch time $b = 2$, and workpiece 1 at launch time $b = 3$. The set of workpieces that are launched together is the *product bundle*, or *bin*. The total processing time of each product bundle in Figure **??** does not exceed the cycle time at any station; moreover, the total size of each product bundle is not larger than the size of the mobile platform. Because workpiece 4 is launched at time $b = 2$, i.e., 1 period later than its due date, the total tardiness equals 1.

As with the bin packing problem, we can interpret a product bundle launched at time b as an m-dimensional bin with capacity c. Therefore, we refer to product bundles as *bins* in the following sections. We will also use more general term *job* instead of workpiece.

4.1.3 Literature Overview

Assembly lines, in which each workpiece may represent a unique *model* with unique processing times, are referred to as *mixed-model* assembly lines in the literature (see literature reviews on assembly lines and assembly line balancing in (Battaïa and Dolgui, 2013; Baybars, 1986; Becker and Scholl, 2006; Boysen et al., 2007, 2008, 2009a; Erel and Sarin, 1998; Ghosh and Gagnon, 1989; Otto and Battaïa, 2017; Scholl and Becker, 2006)). To the best of our knowledge, none of the article in this thread of the literature considers multiple-piece-flow assembly lines (e.g., (Bautista et al., 2016; Cortez and Costa, 2015; Dörmer et al., 2015; Faccio et al., 2016; Mosadegh et al., 2017; Otto and Scholl, 2011; Otto and Otto, 2014b)). Boysen et al. (2009b) introduce a classification system for sequencing models in mixed-model assembly lines and provide an extensive literature review.

However, this classification solely pertains to one-piece-flow assembly lines.

The m-BiPacS is also closely related to the classical optimization problems of the vector bin packing, multiple knapsack, batch scheduling, and lot-sizing and scheduling. However, the existing problem formulations and their solution methods cannot be straightforwardly applied to solve the m-BiPacS as we explain below.

The *vector bin packing problem (VBP)* revolves around minimizing the number of bins – each stretching in m independent dimensions and having capacity c in each dimension – needed to store all the items $j \in J$ of size v_{ji} along dimension $i \in \{1, \ldots, m\}$. The VBP, which was introduced by Garey et al. (1976), is known to be NP-hard in the strong sense. Therefore, most publications study the two-dimensional VBP (Alves et al., 2014; Caprara and Toth, 2001; Chang et al., 2005; Heßler et al., 2018), with the exception of few recent papers mostly motivated by IT applications in which the authors study the VBP with more than two dimensions (Caprara et al., 2003; Gabay and Zaourar, 2016; Ng et al., 2008). We refer the interested reader to the survey of Christensen et al. (2016). The objective function of the m-BiPacS is different from that of the VBP. For instance, as we explain in Section 4.2.2, solutions of m-BiPacS instances with large numbers of bins may have better objective values that those with the minimal number of bins.

Several studies investigate a generalization of the VBP motivated by logistics applications in which bins represent boxes or containers and the objective is to minimize the sum of bin costs. The cost of the bin may depend on its size (Wei et al., 2015), the number of items it contains (Epstein and Levin, 2012), its utilization (Leung and Li, 2008; Li and Chen, 2006), or additional factors such as weight, volume, manpower, and transport distance of the bin (Hu et al., 2018). However, these articles usually consider just one- (Epstein and Levin, 2012; Leung and Li, 2008; Li and Chen, 2006) or two-dimensional bins (Hu et al., 2018). Moreover, in contrast to the m-BiPacS, the sequence of bins is irrelevant to the objective function in this generalization of the VBP.

In the *m-dimensional multiple knapsack problem (MMKP)*, a subset of items has to be packed into several multidimensional knapsacks so that the total value of the selected items is maximized (Camati et al., 2014; Kellerer et al., 2004; Song et al., 2008). We refer the interested reader to the surveys of Fréville (2004); Gavish and Pirkul (1985); Kellerer et al. (2004); Laabadi et al. (2018); Puchinger et al. (2010), and Varnamkhasti (2012). In contrast to the m-BiPacS, we do not have to pack all the items into the knapsacks (bins) in the MMKP. Moreover, the item profits (=negative cost parameters) $-w_{jb} := -w_j$ are independent from the knapsack's position b, so that the sequence of the knapsacks (=bins) does not influence the objective value.

Similar to the m-BiPacS, the *batching scheduling problem (BP)* is to partition jobs into ordered sets (batches) with respect to some scheduling criterion (e.g., minimize sum of completion times or weighted tardiness) (Brucker et al., 1998). Two variants of the batching problem have been studied in the literature: the parallel-batching scheduling problem (p-BP) and the serial-batching scheduling problem (s-BP) (Brucker, 2007). The p-BP is motivated by the scheduling of so-called batching machines that can handle several jobs simultaneously (Cabo et al., 2018, 2015; Wang, 2011). However, in the p-BP, the size of a batch, i.e. its processing time, is variable – it is normally computed as the maximum processing time of the jobs in the batch. In the p-BP, the number of jobs in a batch is usually given. Recall that in m-BiPacS, the size of batches, or bins, is given and equals c, whereas the number of jobs in a bin is variable. In the s-BP, the processing time of the batch is equal to the total processing time of all jobs in this batch and some setup time may be required between subsequent batches. The size of the batches (e.g., their processing time) is usually not limited in the s-BP, whereas the restriction on the size of the batches (=total size of the bins along dimensions $\{1, ...m\}$) is central to the formulation of the m-BiPacS. To the best of our knowledge, only Aloulou et al. (2014); Cheng and Kovalyov (2001), and Yuan et al. (2007) study the s-BP with batches of a limited size, however, even in these cases, the size of a batch is bounded just by the number of jobs. Moreover, none of these papers examines multidimensional batches. We refer the interested reader to the surveys of Brucker (2007); Potts and Kovalyov (2000); Potts and Van Wassenhove (1992), and Webster and Baker (1995), which explore scheduling problems with batching, and to the survey of Mönch et al. (2011), which examines various scheduling problems, including the batching problem, in the context of semiconductor manufacturing.

The m-BiPacS is distinct from the *lot-sizing and scheduling problem (LSP)* (Fleischmann, 1990; Fleischmann and Meyr, 1997; Haase, 1996). The LSP involves partitioning jobs into batches, or lots, and scheduling production of the batches to meet demand in each production period, while considering trade-offs between the inventory holding costs and the setup costs between the batches. We refer the reader to a recent survey by Copil et al. (2017) for a more detailed discussion of the LSP.

4.1.4 Contribution

To the best of our knowledge, this article is the first to study operations planning in multiple-piece-flow assembly lines. We examine product sequencing decisions. For this purpose, we formulate the m-BiPacS with a specific type of objective functions that include, for instance, weighted tardiness and we propose a mixed-integer model for the m-BiPacS. We show that m-BiPacS is NP-hard in the strong sense. We discuss a polynomially solvable special case of the problem and prove some useful properties, such as the maximum load rule. We also discuss lower bounds and possible ways to strengthen the problem formulation. Our fast and effective iterative variable neighborhood heuristic procedure IVNH finds good-quality solutions for instances of practice-relevant size. In particular, IVNH relies on neighborhood definitions of relatively low computational complexity that are able to construct many distinct neighbor solutions with maximally packed bins (cf. Section 4.2.2). We perform extensive computational experiments on a large number of randomly generated instances with different settings and perform a simulation based on a real-world data set. For instance, our computational experiments in Sections 4.4.2 and 4.4.3 illustrate that idle times can be kept low in multiple-piece-flow assembly lines even with highly customized products.

We proceed as follows. In Section 4.2, we analyze the problem. Section 4.3 describes our heuristic solution procedure. Computational experiments are presented in Section 4.4. We conclude with a summary of major findings and future research directions in Section 4.5.

4.2 The m-vector bin packing and sequencing problem

We present an integer programming formulation for the m-BiPacS in Section 4.2.1. Afterward, in Section 4.2.2, we describe several useful problem properties.

4.2.1 Integer programming formulation

Recall that, in analogy to the bin packing problem, we refer to the possible launch time b of the products in the paced multiple-piece-flow assembly line as *bin b*. The following is a mixed-integer formulation of the m-BiPacS with assignment decision variables:

$$\text{minimize} \quad F(\mathbf{x}) = \sum_{j \in J} \sum_{b=r_j}^{\bar{B}} w_{jb} \cdot x_{jb} \tag{4.1}$$

$$\text{s.t.} \quad \sum_{b=r_j}^{\bar{B}} x_{jb} = 1 \qquad \forall j \in J \tag{4.2}$$

$$\sum_{j \in J: r_j \leq b} v_{ji} \cdot x_{jb} \leq c \qquad \forall i \in \{1, \ldots, m\}, \forall b \in \{1, \ldots, \bar{B}\} \tag{4.3}$$

$$x_{jb} \in \{0, 1\} \qquad \forall j \in J, \forall b \in \{1, \ldots, \bar{B}\} \tag{4.4}$$

where binary decision variable x_{jb} denotes whether job j is assigned to bin b ($x_{jb} = 1$) or not ($x_{jb} = 0$). The objective (4.1) is to minimize total cost. Parameter $\bar{B}$ describes the upper bound on the number of the required bins, for example, $\bar{B} := |J| + \max_j\{r_j\}$.

Each job has to be assigned to exactly one bin and not earlier than its release time r_j (constraints (4.2)). Capacity constraints (4.3) should be respected for each bin along each dimension, and constraints (4.4) state that the decision variables are binary.

4.2.2 Properties of the m-BiPacS

In this section, we formulate several properties of the m-BiPacS that we use in our solution procedures.

Table 4.2: Illustrative example:
A m-BiPacS-M instance with five jobs, one dimension $m := 1$, $c := 10$, $r_j := 0\ \forall j \in J$,
and total tardiness objective function $w_{jb} := \max\{b - d_j; 0\}$

	Jobs				
	1	2	3	4	5
Size v_{j1}	5	5	3	3	4
Due date d_j	1	2	1	2	3

For instance, observe that in contrast to the closely related bin packing problem, optimal solutions of the m-BiPacS instances do not necessarily contain the lowest possible number of bins, although cost parameters w_{jb} in the m-BiPacS are non-decreasing in the ordering number of the bins. Indeed, we may have to increase the number of bins by moving large less-urgent jobs to the last bins and move small urgent jobs to the earlier bins to receive an optimal solution – as illustrated in the example in Table 4.2. In this example, we can pack all the jobs into two bins as $\{1, 2\}, \{3, 4, 5\}$ with cost of 1. But when we move less-urgent job 2 to a later bin and process urgent job 3 earlier as $\{1, 3\}, \{2, 4\}, \{5\}$, the cost falls to 0.

Lemma 4.2.1. *The m-BiPacS is NP-hard in the strong sense.*

Proof. Recall that the VBP is NP-hard in the strong sense (Garey et al., 1976). The respective decision problem (D-VBP) is to determine whether a feasible solution with not more than $\bar{B}^{VBP}$ bins exists (yes/no); $\bar{B}^{VBP}$ is a natural number. The decision problem corresponding to the m-BiPacS *(D-m-BiPacS)* is to find out whether a feasible solution with an objective value of not more than $\bar{F}$ exists (yes/no); $\bar{F}$ is some real number. Observe that the D-VBP can be solved by solving the instance of the D-m-BiPacS with parameters $r_j := 0, d_j := \bar{B}^{VBP}, \bar{F} := 0, w_{jb} := \begin{cases} 0 & \text{if } b \leq d_j, \\ 1 & \text{if } b > d_j. \end{cases}$. Therefore, the complexity of the D-m-BiPacS (and the m-BiPacS) is not less than that of the D-VBP (the VBP), respectively. We conclude that m-BiPacS is NP-hard in the strong sense. □

Let $J^b(s)$ be the set of jobs assigned to bin b and $J^{<b}(s)$ be the set of jobs assigned to the bins preceding bin b in some solution s of an m-BiPacS instance. In the following, we simply write J^b and $J^{<b}$ if the notation can be unambiguously constructed from the context. We call bin b *maximally packed* if no additional job $j \in J \setminus \left(J^{<b} \cup J^b\right)$ can be assigned to this bin without violating either its release time r_j or the bin's capacity.

Property 1. (Maximum load rule) There is an optimal solution to the m-BiPacS with only maximally packed bins.

Proof. The claim follows immediately from the assumption of non-decreasing cost parameters w_{jb} in the ordering number of the bins (cf. the definition of w_{jb} in Section 4.1.2). □

The maximum load rule is well known in bin packing. It allows us to construct algorithms that examine *only* solutions with maximally packed bins.

We use the next property, Property 2, to construct a well-performing neighborhood (neighborhood $\mathcal{N}_3$) in our heuristic algorithm in Section 4.3 and to strengthen the model formulation (4.1)-(4.4), as described below.

Property 2. The m-BiPacS with equal job sizes (i.e., $v_{ji} = v_{j'i}$ $\forall j, j' \in J$ and $\forall i \in \{1, \ldots, m\}$) is polynomially solvable.

Proof. Let n_b be the maximal number of jobs that can be packed in bin b. To assign jobs to bins, we formulate a transportation problem (see Hitchcock, 1941; Winston and Goldberg, 2004) as described below. The transportation problem can be solved, for example, with the algorithm of Kleinschmidt and Schannath (1995) in polynomial time.

The transportation problem consists of finding the number of items to be transported from each supply center to each demand center in order to satisfy the demand, so that the transportation cost is minimized and the needs of demand centers are satisfied. We model $\bar{B}$ bins as demand centers with demand n_b (because each bin contains a maximum of n_b jobs) and jobs as supply centers with a supply of 1 each (because each job has to be assigned to exactly one bin). We introduce $\bar{B} \cdot n_b - |J|$ dummy jobs to match the supply and the demand. We set transportation costs of one product unit from supply center $j \in J$ to demand center b to w_{jb} if $b \geq r_j$ and to some sufficiently large number M, e.g., $M := \sum_{j \in J} \max_{b \in \{1,\ldots,\bar{B}\}} \{w_{jb}\} + 1$, otherwise. Observe that each job j has to be assigned to exactly one bin and, therefore, the objective value of any feasible solution, in which all jobs are assigned to bins not earlier than their release times, is lower than M. Transportation costs per product unit from dummy supply centers equal 0.

The constructed transportation problem indeed solves the m-BiPacS with equal job sizes, because it respects all the constraints of the m-BiPacS (and only them): bin capacity constraints (4.3) as well as assignment and release time constraints (4.2) (see Section 4.2.1). The constructed problem also correctly represents the objective function of the m-BiPacS.

Note that if jobs additionally have the same assignment costs ($w_{jb} = w_b\ \forall j \in J$), we can get an optimal solution by examining them in the non-decreasing order of their due dates and assigning them to the earliest bin possible. □

Observe that the bin packing problem is closely related to the m-BiPacS, for instance, the decision problem of the bin packing is a special case of the decision problem D-m-BiPacS. Therefore, several properties of the bin packing problem can be directly adapted to the m-BiPacS. These properties include, for example, the Jackson dominance rule (cf. Jackson, 1956) and the lower bounds of Martello and Toth (1990). However, in our experience, the quality of these lower bounds drastically deteriorates with the increasing number of bin dimensions m. Therefore, we formulate an alternative lower bound LB based on computing N_b, which denotes an upper bound on the number of jobs that can be packed into bin b in a solution with maximally packed bins due to bin size and release time restrictions, and relaxing the problem to the transportation problem. We describe lower bound LB below. Further, observe that model (4.1)-(4.4) can be strengthened by adding the following constraints as a simple and computationally inexpensive way to strengthen model formulation (4.1)-(4.4):

$$\sum_{j \in J: r_j \leq b} x_{jb} \leq N_b \qquad \forall b \in \{1, \ldots, \bar{B}\} \tag{4.5}$$

In the computational experiments in Section 4.4.2.3, we show that constraints (4.5) may speed up the standard solver by several times.

We compute N_b in the increasing order of the bin numbers starting with $N_{b'}, b' = \min_{j \in J}\{r_j\}$. We estimate N_b by solving the *one*-dimensional knapsack problem for the current set of jobs J' with *surrogate constraints* (cf. (Caprara and Toth, 2001; Glover, 1977)), that is, some weighted sums of constraints (4.3). The objective $KS(\mathbf{x})$ of the knapsack problem is to maximize the number of the selected jobs. Given some weights

$l_i \geq 0, \sum_{i=1}^{m} l_i = 1$, the knapsack problem for bin b can be notated as follows:

$$\text{maximize} \qquad KS(\mathbf{x}) = \sum_{j \in J'} x_{jb} \tag{4.6}$$

$$\sum_{i=1}^{m} \sum_{j \in J'} l_i \cdot v_{ji} \cdot x_{jb} \leq c \cdot \sum_{i=1}^{m} l_i = c \tag{4.7}$$

$$x_{jb} \in \{0,1\} \qquad \forall j \in J' \tag{4.8}$$

We set $N_b := KS(\mathbf{x})$. Observe that we can solve the resulting knapsack problem simply by assigning the jobs in the non-decreasing order of their surrogate size in $O(|J| \log |J|)$.

Initially, for $N_{b'}$, we assign all jobs with sufficiently early release times to set $J' := \{j \in J : r_j \leq b'\}$. Because of the maximal load-rule, bin b' will contain *at least one job* if set J' is not empty. Therefore, for computing $N_{b'+1}$, we ignore job $j' \in J'$ with the largest surrogate size and update set $J' := (J' \setminus \{j'\}) \cup \{j \in J : r_j = b' + 1\}$ as described in Algorithm 1. Indeed, since we can solve the knapsack problem (4.6)-(4.8) by assigning the jobs in the non-decreasing order of their surrogate size, $N_{b'+1}$ is the largest if we select job $j' \in J'$ with the largest surrogate size. We proceed in a similar manner for all $b \in \{b', \ldots, \bar{B}\}$.

```
Initialise $b' := \min_{j \in J}\{r_j\}$ ;
Initialise $J' := \{j \in J : r_j \leq b'\}$ ;
for $(b := b'; b \leq \bar{B}; b{+}{+})$ do
    if $|J'| > 0$ then
        Find $N_b$ by solving knapsack problem (4.6)-(4.8) for $J'$ and $b$;
        Set $j'$ to be a job in $J'$ with the largest (surrogate) size;
        $J' := (J' \setminus \{j'\}) \cup \{j \in J : r_j = b + 1\}$;
    else
        $N_b := 0$;
        $J' := J' \cup \{j \in J : r_j = b + 1\}$;
    end
end
```

Algorithm 1: Computation of N_b

Based on the calculated parameters N_b, we compute lower bound LB by formulating and solving a transportation problem with $n_b := N_b$, as described in the proof of Property 2.

Observe that LB is indeed a lower bound for the m-BiPacS because each feasible solution of the m-BiPacS with maximally packed bins (see Property 1) is also feasible for the formulated transportation problem and has the same objective value. As explained above, N_b is an upper bound on the number of jobs that can be assigned to bin b in a feasible solution with maximally packed bins. Therefore, in fact, the formulated transportation problem is equivalent to problem (4.1)-(4.4) with constraints (4.3) relaxed to $\sum_{j\in J:r_j\leq b} x_{jb} \leq N_b \; \forall b \in \{1,\dots,\bar{B}\}$.

4.3 Iterative variable neighborhood heuristic

To address large problem instances of the m-BiPacS, we propose iterative variable neighborhood heuristic *(IVNH)*. The IVNH examines only solutions with *maximally packed bins.* Recall that the objective value of any solution does not get worse and potentially significantly improves, if we shift jobs to pack bins maximally (cf. Section 4.2, Property 1). This is a key factor influencing the performance of the IVNH.

The IVNH iteratively performs local search with three different neighborhoods: $\mathcal{N}_1, \mathcal{N}_2$, and $\mathcal{N}_3$ (see Section 4.3.2). Having reached a local optimum with respect to neighborhood $\mathcal{N}_i$, $i = 1, 2$, we move to the next neighborhood $\mathcal{N}_{i+1}$. Whenever we reach a local optimum in all the three neighborhoods, we perform a diversification procedure (see Section 4.3.3). We construct our initial solution with a greedy procedure as described in Section 4.3.1. Section 4.3.4 provides a summary of the IVNH and its pseudocode.

4.3.1 Initial heuristic

We adapt the priority rule based method *(PRBM)* of Otto and Otto (2014a) to find an initial solution. The PRBM is a greedy procedure based on multiple cues, which constructs a large number of different good-quality feasible solutions. At each PRBM iteration (l), the PRBM constructs a feasible solution by setting priorities $p_j^{(l)}$ for each job $j \in J$ and assigning jobs to bins in a non-increasing order of their priorities. We apply a *first-fit* procedure to assign jobs, i.e., we assign each job to the earliest bin, to which it can be assigned because of bin size and release time restrictions.

We compute three *elementary* priority values for each job j scaled to the interval of $[0, 1]$: p_j^w, p_j^{dd}, and p_j^s. Priority values describe factors (cues) that are likely to lead to a good assignment of jobs. Values $p_j^w := \frac{w_j}{\max_{j'\in J}\{w_{j'}\}}$ force jobs with large weights to be assigned to early bins. Values p_j^{dd} assign high priority values to jobs with early due dates. We set $p_j^{dd} := 1 - \frac{d_j - \min_{j'\in J}\{d_{j'}\}}{\max_{j'\in J}\{d_{j'}\} - \min_{j'\in J}\{d_{j'}\}}$ if due dates differ among jobs (i.e., $\max_{j'\in J}\{d_{j'}\} > \min_{j'\in J}\{d_{j'}\}$) and set $p_j^{dd} := 0$ otherwise. Value p_j^s informs on the job's size: $p_j^s := \frac{\sum_{i=1}^m v_{ji}\cdot e_i}{\max_{j'\in J}\{\sum_{i=1}^m v_{j'i}\cdot e_i\}}$, where coefficient $e_i := \sum_{j''\in J} v_{j''i}$ assigns a higher importance to the bottleneck dimensions.

We compute priorities as a weighted sum of the three elementary priority values: $p_j^{(l)} = \pi_1^{(l)} \cdot p_j^w + \pi_2^{(l)} \cdot p_j^{dd} + \pi_3^{(l)} \cdot p_j^s$. Coefficients $\pi_1^{(l)}, \pi_2^{(l)}, \pi_3^{(l)}$ weigh the importance of elementary priority values. We exploit the low computational complexity of priority rules and perform a brute search over many possible values of the elementary priority values' weights. We select values $\pi_1^{(l)}$ and $\pi_2^{(l)}$ from the set $\{0, 0.1, 0.2, \dots, 1\}$ to assign high importance to jobs with large weights and early deadlines. Observe that ranking jobs according to their size has two opposing effects. From the one hand, if we assign larger jobs first, we are likely to require less bins, which is good. On the other hand, since assignment costs are monotonous non-decreasing in the ordering number of the bins, we are interested in assigning as many jobs as possible to the earlier bins to reduce the assignment costs for the largest possible number of jobs. For the latter reason, it may be beneficial to assign smaller jobs first. Therefore, we construct priorities both with non-negative and negative values of coefficient $\pi_3^{(l)}$ and select values $\pi_3^{(l)}$ from the set $\{-1, -0.9, -0.8, \dots, 1\}$.

We perform $11 \cdot 11 \cdot 21 = 2541$ iterations of the PRBM in total, because $\pi_1^{(l)}$ and $\pi_2^{(l)}$ take 11 values and $\pi_3^{(l)}$ takes 21 values. Since some of these iterations use equivalent priorities, such as priorities with $(\pi_1^{(l)}, \pi_2^{(l)}, \pi_3^{(l)})$ of $(0.1, 0.1, 0.1)$ and $(0.2, 0.2, 0.2)$, the number of the constructed distinct feasible solutions is somewhat lower. We choose the best found solution to be the initial solution.

Table 4.3: Instance of the m-BiPacS-M with eight jobs, three dimensions, $c := 10$, equal release times $r_j := 0\ \forall j \in J$, and the objective to minimize total tardiness

	Jobs							
	1	2	3	4	5	6	7	8
v_{j1}	5	4	3	4	2	7	2	3
v_{j2}	4	5	4	2	1	6	3	2
v_{j3}	2	4	5	5	3	7	1	1
d_j	1	1	1	2	2	3	3	4

Table 4.4: Priorities of the jobs in the instance of Table 4.3 according to the EDD rule ($\pi_1^{(l)} := 0, \pi_2^{(l)} := 1$, and $\pi_3^{(l)} := 0$)

	Jobs							
	1	2	3	4	5	6	7	8
Priority values p_j	1	1	1	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0

Consider the instance in Table 4.3 with eight jobs, three dimensions, $c := 10$, equal release times $r_j := 0\ \forall j \in J$, and the objective to minimize total tardiness. Table 4.4 illustrates priorities of the jobs in some iteration l with $\pi_1^{(l)} := 0, \pi_2^{(l)} := 1$, and $\pi_3^{(l)} := 0$. (We call this rule as the *earliest due date rule (EDD)* in our computational experiments.) We break the ties by taking a job with a lower ordering number. In this iteration l of the PRBM, we receive a feasible solution as follows. We sort jobs in a non-decreasing

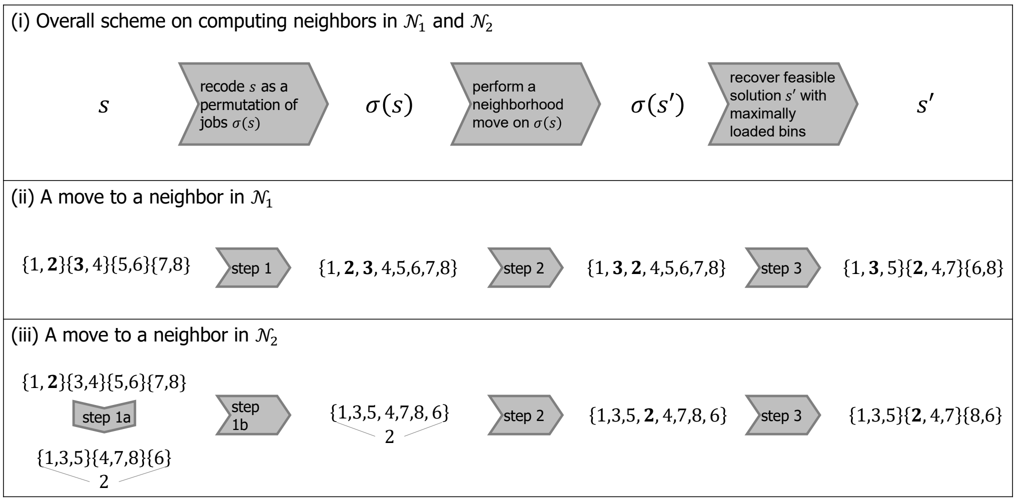

(i) Overall scheme on computing neighbors in $\mathcal{N}_1$ and $\mathcal{N}_2$
s
recode s as a permutation of jobs $\sigma(s)$
$\sigma(s)$
perform a neighborhood move on $\sigma(s)$
$\sigma(s')$
recover feasible solution s' with maximally loaded bins
s'
(ii) A move to a neighbor in $\mathcal{N}_1$
{1, **2**}{**3**, 4}{5,6}{7,8}
step 1
{1, **2**, **3**, 4,5,6,7,8}
step 2
{1, **3**, **2**, 4,5,6,7,8}
step 3
{1, **3**, 5}{**2**, 4,7}{6,8}
(iii) A move to a neighbor in $\mathcal{N}_2$
{1, **2**}{3,4}{5,6}{7,8}
step 1a
{1,3,5}{4,7,8}{6}
2
step 1b
{1,3,5, 4,7,8, 6}
2
step 2
{1,3,5, **2**, 4,7,8, 6}
step 3
{1,3,5}{**2**, 4,7}{8,6}

Figure 4.3: Illustration: Computing neighbors in $\mathcal{N}_1$ and $\mathcal{N}_2$

order of their priorities as $(1, 2, 3, 4, 5, 6, 7, 8)$ and assign them with the first-fit procedure to receive solution $\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}$ with objective value of 3.

The time complexity of the first-fit assignment procedure is $O(|J| \cdot m \cdot \bar{B})$ because we have to examine m dimensions of up to $\bar{B}$ bins for each job. The time complexity to compute vectors of elementary priority values is linear in $|J|$ for p^w and p^{dd}. It is linear in $|J| \cdot m$ for p^s. Therefore, the time complexity of one iteration of the PRBM equals $O(|J| \cdot m \cdot \bar{B})$, or $O(|J|^2 \cdot m)$, because the number of non-empty bins cannot exceed the number of jobs.

4.3.2 Local search procedures

We formulate three neighborhoods: $\mathcal{N}_1$ is based on two-job exchanges, $\mathcal{N}_2$ examines job reinsertions, and $\mathcal{N}_3$ looks for alternative sequences of the bins. We formulate neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ such that they *(i)* examine only solutions with maximally packed bins, *(ii)* have moderate time complexity, and *(iii)* each feasible solution has, as a rule, many distinct neighbors. Although the resequencing neighborhood $\mathcal{N}_3$ may contain solutions with not maximally packed bins, the maximum load rule is enforced in the further search since we always perform $\mathcal{N}_1$- and $\mathcal{N}_2$-local search for such solutions.

We construct a neighbor solution s' of the incumbent solution s *indirectly* by recoding solution s as shown in Figure 4.3(i). We represent incumbent solution s as a *permutation of jobs* $\sigma(s)$ skipping the assignment of jobs to the bins (jobs within each bin can be arbitrarily ordered). Afterward, we perform a neighborhood move and receive another permutation of jobs $\sigma(s')$. Then we construct s' by assigning jobs of $\sigma(s')$ to bins with the first-fit procedure as explained in Section 4.3.1. Such indirect local search procedures have two major advantages compared to the respective conventional local search procedures. First of all, if we would perform neighborhood moves on the original (not recoded) solution s, the number of these moves would be significantly limited, since we will have to preserve the feasibility of the resulting neighbor solution. In our indirect local search procedures, the neighbor moves are performed on permutation of jobs $\sigma(s)$ so that we are not restricted by the m-BiPacS constraints at this step and any resulting permutation of jobs is "feasible". We construct a feasible solution afterward with very flexible first-fit procedure. Secondly, our indirect local search procedures always result in solutions with maximally packed bins. We provide examples for these statements and explain Figures 4.3(ii) and 4.3(iii) below.

Two-job exchange neighborhood $\mathcal{N}_1$. Let denote the incumbent solution as s. Let $j \in J$ be one of jobs and $J^b \subseteq J$ be the set of jobs assigned to the same bin as j in s. We construct a permutation of jobs $\sigma(s)$ by skipping the assignment of jobs to the bins (step 1). Given $\sigma(s)$, we receive a neighbor s' of s by swapping j with some job $j' \in J \setminus J^b$ in $\sigma(s)$ (step 2) and assigning jobs to bins in the resulting order by the first-fit procedure (step 3).

Consider the instance in Table 4.3 and $s := \{1,2\}\{3,4\}\{5,6\}\{7,8\}$ as initial solution. Figure 4.3(ii) illustrates how to construct a neighbor of solution s by swapping jobs 2 and 3. We construct a permutation of jobs $\sigma(s) = \{1,2,3,4,5,6,7,8\}$ in step 1. After swapping jobs 2 and 3, we get permutation $\sigma(s') = \{1,3,2,4,5,6,7,8\}$ (step 2). Then, we apply the first-fit procedure, i.e., we assign each job to the earliest bin, to which it can be assigned, considering them in the order provided by $\sigma(s')$ (step 3). For instance, jobs 1 and 3 can be assigned to bin 1, afterward, jobs 2 and 4 can only be assigned to bin 2. The next job, job 5, fits bin 1. And so on. The resulting solution s' is, in fact, an optimal solution with objective value of 1.

The time complexity to examine all the neighbors of solution s is $O(|J|^3 \cdot m \cdot \bar{B})$ because there are up to $O(|J|^2)$ job exchanges and the complexity of first-fit procedure is $O(|J| \cdot m \cdot \bar{B})$.

Let us illustrate the effectiveness of the formulated indirect neighborhood moves by comparing neighborhood $\mathcal{N}_1$ to the conventional two-job exchange neighborhood on the example in Table 4.3 for $s := \{1,2\}\{3,4\}\{5,6\}\{7,8\}$. In $\mathcal{N}_1$, solution s has 19 distinct neighbor solutions and all of them consist of maximally packed bins. In the conventional neighborhood search, in which we examine feasible swaps of two jobs, there are only 16 (15% less) distinct neighbor solutions and only four of them consist of maximally packed bins; none of these solutions is optimal. This illustrative example is not an exception (see Appendices A and B). It can be shown that neighborhood $\mathcal{N}_1$ of some feasible solution s contains *all* the neighbors of the conventional two-job exchange neighborhood of s that are improved by the subsequent shift of jobs to receive maximally packed bins (see Appendix

A). Recall that by shifting jobs to earlier bins we cannot worsen, but can potentially improve the objective function (see Property 1 in Section 4.2.2).

Job reinsertion neighborhood $\mathcal{N}_2$. Let consider reinsertion of job $j \in J$. Observe that the recoding of a solution is a one-to-many mapping because several different permutations of jobs may correspond to a single solution of m-BiPacS. Therefore, we introduce an additional computational step of low time complexity to exclude (almost all the) identical solutions from the neighborhoods: We transform incumbent solution s into partial solution s^p by assigning jobs $J \setminus \{j\}$ from $\sigma(s)$ according to the first-fit procedure (step 1a). Then, we perform a neighborhood move by inserting job j between two consequent neighboring jobs j_{k-1} and j_k in $\sigma(s^p)$, which belong to two *different* bins in s^p (step 1b and step 2). Afterward, we assign jobs to bins according to the first-fit procedure to receive neighbor solution s' (step 3).

For the instance in Table 4.3 and initial solution $s := \{1,2\}\{3,4\}\{5,6\}\{7,8\}$, we can construct a neighbor solution by reinserting job 2 as shown in Figure 4.3(iii). We receive an optimal solution s' with objective value of 1.

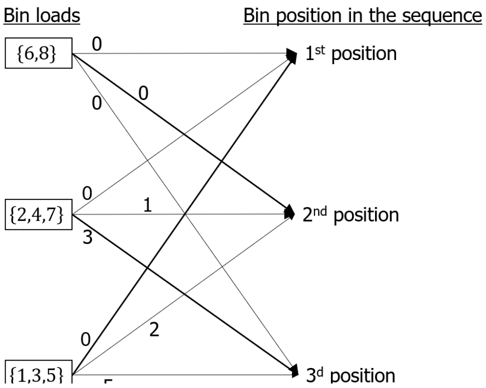
The time complexity to examine all the neighbors of some solution s is $O(|J|^3 \cdot m \cdot \bar{B})$ because there are $|J|$ candidates for the job reinsertion that can be reinserted in up to $O(|J|)$ positions and the complexity of the first-fit procedure is $O(|J| \cdot m \cdot \bar{B})$.

Similar to neighborhood $\mathcal{N}_1$, neighborhood $\mathcal{N}_2$ constructs many distinct neighbor solutions with maximally packed bins. Recall that in the conventional job reinsertion neighborhood, we construct *feasible* neighbor solutions by shifting some job j to some other bin. Therefore, in this conventional neighborhood of some solution s with *maximally packed bins*, we can move any job j only to a later bin. Thus, there exist no improving neighbors for solutions with maximally packed bins.

Overall, in our local search procedure, we search neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ of incumbent solution s until we find some neighbor s' with a better objective value (a so-called *improving neighbor*). We immediately set the improving neighbor to be a new incumbent solution $s := s'$. We repeat our search, until the incumbent solution does not have any improving neighbors, i.e., until a local optimum is found.

Resequencing neighborhood $\mathcal{N}_3$. In the third neighborhood, $\mathcal{N}_3$, we treat the partition of jobs $J = J^1 \cup J^2 \cup \ldots$ into bin loads as given and look for a permutation of the bin loads that results in a feasible solution with the best possible objective value. We formulate the respective assignment problem and solve it in $O(\bar{B}^3 + |J| \cdot \bar{B})$ because the improved version of the Hungarian algorithm takes $O(\bar{B}^3)$ and we can construct the assignment graph in $|J| \cdot \bar{B}$.

Consider neighborhood $\mathcal{N}_3$ of $s := \{1,3,5\}\{2,4,7\}\{6,8\}$. Figure 4.4 visualizes the resulting assignment problem. Its optimal solution is 1. Therefore, there is no better



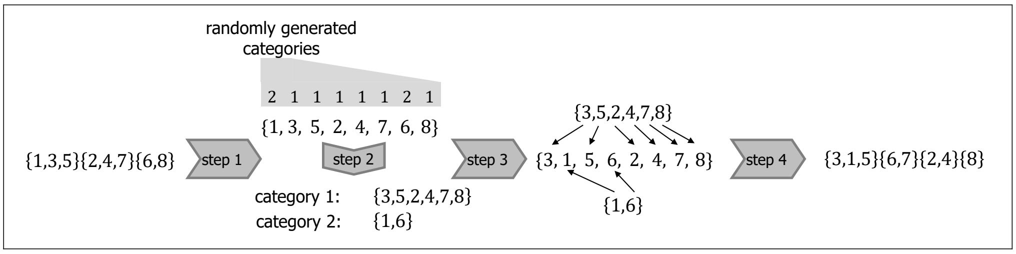


Figure 4.5: Illustration of a diversification step

assignment (sequence of bins) and s is a local optimum with respect to $\mathcal{N}_3$.

4.3.3 Diversification step

A good diversification procedure would overcome a local optimum by jumping to a *far-enough* solution in the solution space. And it should also learn some "good" characteristics of the incumbent solution in order to jump to a "good" region in the solution space. In our computational experiments, the diversification procedure we describe below has outperformed the conventional random diversification, which is jumping to a randomly generated solution (cf. Section 4.4.2.2). Presumably, our diversification procedure has a rather high probability to keep some good combinations of jobs from the current local optimal solution s^* together in the same bin.

To perform diversification, we randomly assign jobs of the recoded current local optimal solution $\sigma(s^*)$ to f categories (subsequences of jobs), i.e., we randomly generate a category for each job from discrete uniform distribution $\mathcal{U}\{1, f\}$ (step 1 and step 2 in Figure 4.5). Afterward, we form a new sequence of jobs $\sigma(s)$ by taking jobs with number one from the f categories, then jobs with number two and so on (step 3). We receive a new incumbent solution s by assigning jobs to bins in the resulting order $\sigma(s)$ by the first-fit procedure (step 4).

Figure 4.5 illustrates the diversification procedure for local optimal solution $s^* := \{1,3,5\}\{2,4,7\}\{6,8\}$ and $f := 2$ categories for the instance in Table 4.3. In step 1, we receive a permutation of jobs $\sigma(s^*) = \{1,3,5,2,4,7,6,8\}$. A random assignment of jobs to the categories resulted in jobs 1 and 6 belonging to category 2 and the rest of the jobs belonging to category 1 (step 2). After having alternately assigned the jobs from the two categories, we receive a new sequence of jobs $\sigma(s) = \{3,1,5,6,2,4,7,8\}$ (step 3). And the first-fit procedure constructs solution $s = \{3,1,5\}\{6,7\}\{2,4\}\{8\}$ (step 4). Observe that the resulting solution s preserves a well-fitting bin load $\{1,3,5\}$.

4.3.4 Summary of the IVNH procedure

Algorithm 2 summarizes the IVNH procedure, which uses parameters $Tlim, Glim, Flim$, and f_{init}. We describe the parameter values that we used in our experiments in Section 4.4.1.

Let denote the following steps as an *iteration* of the IVNH: construction of a new incumbent solution (either as described in Section 4.3.1 or as described in Section 4.3.3) and local search until a local optimal solution with respect to the three neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ is found.

We start by initializing iterations counter it, the objective value of the best solution found so far F^*, the number of iterations from the last update of the best found objective value gt, the locally optimal solution found in the previous iteration s^*_d, and diversification parameter f (row 1). The IVNH starts with an initial solution computed as described in Section 4.3.1 (row 2). Afterward, we perform the local search with respect to the three neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ until a feasible solution s^*_3 is found, which is locally optimal with respect to all the neighborhoods (rows 3-8). In the diversification procedure, we initialize parameter f as $f := f_{init}$ (row 1). If the objective function of the new local optimum s^*_3 in current iteration it is not better than that of the previous one s^*_d, we return to the previous local optimal solution s^*_d and increase f by one (row 13) unless $f = Flim$. In the latter case, we accept the current local optimal solution as incumbent solution $s^*_d := s^*_3$ and reset $f := f_{init}$ (row 11). We update $s^*_d := s^*_3$ and reset the counter f otherwise (row 15). At each iteration, we update the objective value of the best solution found so far (rows 16-19). We stop the IVNH either when the time limit $Tlim$ have been reached or the maximal number of iterations without the improvement in the best found solution equals $Glim$ (rows 23-25).

```
   // initialization
1  Initialize $f := f_{init}, F^* := \infty$, $gt := 0$; $s_d^* := \emptyset; it := 0$;
2  Construct $s$ with initial heuristic (see Section 4.3.1);
   // local search
3  Starting from $s$, find locally optimal solution $s_1^*$ with respect to $\mathcal{N}_1$;
4  Starting from $s_1^*$, find locally optimal solution $s_2^*$ with respect to $\mathcal{N}_2$;
5  Starting from $s_2^*$, find locally optimal solution $s_3^*$ with respect to $\mathcal{N}_3$;
6  if $s \neq s_3^*$ then
7  |   $s := s_3^*$;
8  |   go to 3 ;
9  end
   // diversification
10 if $s_d^* \neq \emptyset$ and $F(s_d^*) \leq F(s_3^*)$ then
11 |   if $f = Flim$ then
12 |   |   $f := f_{init}$; $s_d^* := s_3^*$;
13 |   else
14 |   |   $f := f + 1$; $s_d^* := s_d^*$;
15 |   end
16 else
17 |   $f := f_{init}$; $s_d^* := s_3^*$;
18 end
19 if $F^* > F(s_d^*)$ then
20 |   $F^* := F(s_d^*)$; $gt := 0$;
21 else
22 |   $gt := gt + 1$;
23 end
24 $it := it + 1$;
25 Apply diversification procedure for $s_d^*$ with parameter $f$ to receive $s_{div}^*$ (see Section 4.3.3);
26 $s := s_{div}^*$;
   // verification of the stopping criterion
27 if elapsed time $< Tlim$ and $gt < Glim$ then
28 |   goto 3;
29 end
```

Algorithm 2: Pseudocode of the IVNH

4.4 Computational Experiments

We perform our computational experiments on a diversified randomly generated data set and as a simulation based on the real-world data of an industrial equipment producer. Section 4.4.1 describes the randomly generated data set. We then compare the performance of the IVNH and of an off-the-shelf solver in Section 4.4.2.1 and illustrate the results of the IVNH in a simulation of a real-world rolling-horizon planning framework in Section 4.4.3.

4.4.1 Data generation

Since the literature lacks a benchmark data set for the m-BiPacS, we have randomly generated a data set with a wide range of various instance characteristics for our experiments. We examine a total of 440 instances in our experiments, encompassing 22 parameter settings with 20 randomly generated instances for each parameter setting.

The *large data set (LDS)* contains large instances with $n \geq 100$ jobs. In the *basic* setting, we set $n := 150$, number of dimensions $m := 5$, dimensions' capacities $c := 100$, and release times $r_j := 0$ for all the jobs. For each job $j \in J$ and each dimension $i \in \{1, \ldots, m\}$, we randomly and independently generate sizes v_{ji} from the interval $[\underline{v}, \overline{v}] := [10, 30]$. Deadlines d_j are randomly and independently drawn from the interval $[\lceil \frac{L}{2} \rceil, \lceil L \rceil]$ (we designate this interval as *large*), where L is a reference value that depends on the number of jobs and their sizes. The value of L increases if the number of jobs or their sizes increase: $L := \frac{n}{c/E(v_{ji})}$, where $E\left(v_{ji}\right)$ is the expectation of the job's size. In the basic setting, $[\lceil \frac{L}{2} \rceil, \lceil L \rceil] = [15, 30]$. We have selected this interval for the due dates to include both in-time and delayed jobs in each instance. For example, on average about 5.5% of jobs are delayed in the best known solution of the basic setting. We minimize the total tardiness, so that $w_{jb} := \max\{d_j - b, 0\}$.

We construct another 11 settings in the LDS by varying one set of parameters at a time. For example, we examine two additional settings with $n := 100$ and $n := 200$, two settings with $m := 2$ and $m := 8$, and four settings with intervals $[\underline{v}, \overline{v}]$ equal to $[0, 20]$, $[20, 40]$, $[30, 50]$, and $[0, 100]$. We introduce a setting with release times by randomly and independently drawing them from $[0, \lceil \frac{L}{2} \rceil]$. Two further settings examine smaller, more restrictive due dates by drawing them from intervals $[0, \lceil \frac{L}{2} \rceil]$ *(small)* and $[\lceil \frac{L}{4} \rceil, \lceil \frac{3L}{4} \rceil]$ *(medium).* The LDS contains 12 settings with 240 instances in total.

The *small data set (SDS)* contains smaller instances with $n := 30$ jobs. Because we fix the number of jobs in the SDS, it consists of 10 settings arranged along the same lines as the LDS.

In the following sections, we describe a setting as $(n, m, [\underline{v}, \overline{v}]$, release time, due date$)$.

We performed our experiments on a personal computer with an Intel i7-8700K processor, 6 cores, and 16 GB RAM. We compared the performance of the IVNH to that of the off-the-shelf software IBM ILOG CPLEX 12.8 (which we simply call *CPLEX* below). We used default settings of CPLEX. We set the objective function to optimize the total tardiness, i.e., $w_{jb} := \max\{b - d_j; 0\}$, unless specified otherwise.

We set the IVNH parameters to $Glim := 20$, $f_{init} := 2$, $Flim := 4$, and $Tlim := 10$ minutes in our experiments.

In the computational experiments, we report the *average absolute performance* and the *average relative performance.* Because optimal objective values of some of the instances are not known, we compare the algorithm's solutions for each instance ψ to the *best* objective value found in all the computational experiments for this instance ψ.

The average absolute performance refers to the average difference in the absolute objective value of the algorithm's solution and the objective value of the best found solution. For instance, the absolute performance equals 1 if just one tardy job was assigned one bin later compared to the best found solution.

The average relative performance denotes the average relation of the difference between the objective values found by the algorithm and the best found solution to the best found objective value. Note, however, that the average relative performance is generally *not very informative* for such objective functions as tardiness. Overall, relative performance may be large even for very small absolute gaps between the algorithm's objective function value and the optimal objective function value. For instance, if the optimal tardiness is 0, then relative performance would be undefined (equal infinity).

Overall, we were able to solve all the instances in the SDS and 37 instances of the LDS to optimality.

4.4.2 Computational experiments on the randomly generated data sets

4.4.2.1 Comparative performance of the IVNH and CPLEX

Table 4.5: Performance of the IVNH and CPLEX on the LDS

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	No. of found best known solutions		Avg. absolute performance		Avg. relative performance		Avg. objective value of best known solutions	Avg. abs. performance of the best known solution to the lower bound of CPLEX
					IVNH	CPLEX	IVNH	CPLEX	IVNH	CPLEX		
100	5	[10 30]	no	large	19	2	0.1	3.4	0.8%	68.1%	5.9	5.4
⌈	2	[10 30]	no	large	13	18	0.8	0.9	87.9%	56.7%	1.7	0.7
|	⌈	[0 100]	no	large	20	14	0.0	48.4	0.0%	4.8%	1350.7	1090.4
|	|	[0 20]	no	large	20	20	0.0	0.0	-*	-*	0.0	0.0
|	|	⌈	no	large	20	0	0.0	15.0	0.0%	140.9%	12.6	11.6
150	5	[10 30]	yes	large	20	0	0.0	14.0	0.0%	122.2%	12.9	11.8
|	|	|	no	medium	19	1	0.3	25.2	0.1%	6.9%	370.3	104.0
|	|	⌊	no	small	19	2	0.1	15.1	0.0%	1.4%	1077.3	141.4
|	|	[20 40]	no	large	19	3	0.1	3.1	0.2%	13.3%	25.5	25.5
|	⌊	[30 50]	no	large	20	20	0.0	0.0	0.0%	0.0%	345.9	80.0
⌊	8	[10 30]	no	large	20	0	0.0	34.8	0.0%	112.0%	33.1	32.1
200	5	[10 30]	no	large	20	0	0.0	41.6	0.0%	163.8%	27.3	25.6

* Not defined if the best found objective value is 0

Table 4.5 reports results of the IVNH and CPLEX for the LDS. We run CPLEX on the extended model formulation (4.1)-(4.5). Because the IVNH takes about 1 minute on average and never more than 10 minutes per instance, we limit the run time of CPLEX to 10 minutes for the sake of a meaningful comparison.

CPLEX solved just 37 out of 240 instances to optimality: 14 instances in setting (150, 2, [10 30], no, large), 20 instances in setting (150, 5, [0 20], no, large), and 3 instances in setting (150, 5, [30 50], no, large). The average absolute optimality gap, that is the difference between the CPLEX upper and the CPLEX lower bounds, remained quite large.

In fact, CPLEX lower bounds were close to 0 in the majority of the settings (e.g., compare the last two columns in Table 4.5).

The IVNH required just 88 seconds per instance on average and performed at most 101 iterations for each instance. Table 4.5 illustrates that the IVNH clearly outperformed CPLEX in 9 out of 12 settings and found solutions of comparable quality as CPLEX in the remaining three settings (150, 2, [10 30], no, large), (150, 5, [0 20], no, large), and (150, 5, [30 50], no, large).

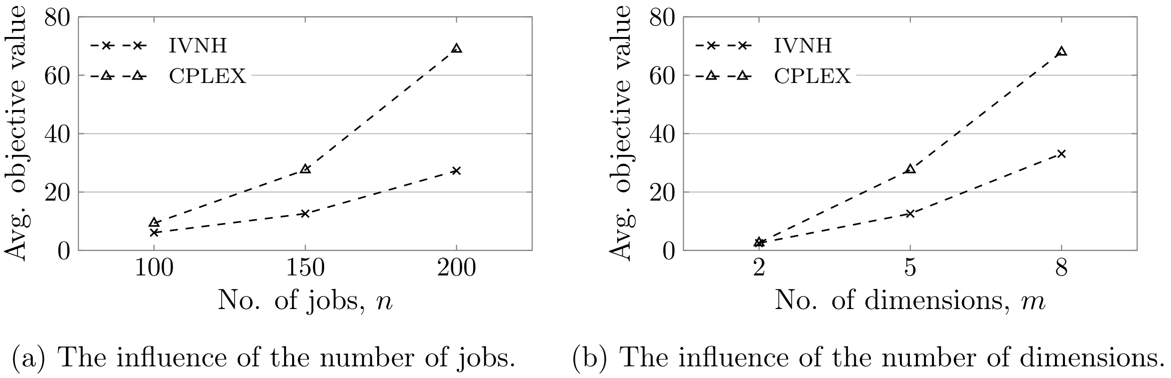
Note that we were not able to compare the results of the IVNH to optimal solutions of all the instances of the LDS. Indeed, CPLEX was not able to solve the LDS instances to optimality even within a several-hour run time per instance.

Table 4.6: Performance of the IVNH and CPLEX on the SDS

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	IVNS		
					Avg. CPU, sec.	No. of optimal solutions (out of 20)	Avg. absolute performance
30	2	[10 30]	no	large	0.00	20	0.0
	5	[0 100]	no	large	0.39	20	0.0
		[0 20]	no	large	0.00	20	0.0
		[10 30]	no	large	0.01	20	0.0
			yes	large	0.01	20	0.0
			no	medium	0.16	15	0.3
			no	small	0.16	16	0.2
		[20 40]	no	large	0.00	20	0.0
		[30 50]	no	large	0.20	20	0.0
	8	[10 30]	no	large	0.03	20	0.0

Therefore, in order to compare the results of the IVNH with optimal solutions, we run the algorithms on the SDS data set (see Table 4.6). We did not limit the run time of CPLEX. CPLEX required up to 70 minutes per instance. Overall, the IVNH found optimal solutions in 95.5% of cases (for 191 instances out of 200) in a fraction of a second. In a few (nine out of 200) cases, in which the IVNH was not able to find an optimal solution, the remaining gap to optimality (= absolute performance) was very low. It equaled 1 in eight instances and 2 in the ninth instance. Recall that the gap to optimality equals 1, for instance, if just one job has been assigned one bin later. Interestingly, all these nine instances belong to the settings with more restrictive – small and medium – due dates. The IVNH found optimal solutions for $\frac{15+16}{20+20} \approx 75\%$ of the instances in these two settings. A possible reason could be the following. Because of more restrictive due dates, optimal solutions are likely to contain many tardy jobs. As a consequence, we are more likely to be trapped in a local optimum, because many neighbors in the neighborhoods $\mathcal{N}_1, \mathcal{N}_2$, and $\mathcal{N}_3$ are not improving neighbors and will be rejected. So that the instances get harder for the IVNH and the relative advantage of enumeration-based procedures, such as CPLEX, becomes more pronounced.

We conclude that the IVNH performs very well. Overall, the relative advantage of



(a) The influence of the number of jobs. (b) The influence of the number of dimensions.

Figure 4.6: Sensitivity analysis of the basic setting of the LDS instances – (150, 5, [10 30], no, large): Relative advantage of the IVNH over CPLEX

the IVNH seems larger for a larger number of dimensions m and for a larger number of jobs n, see Figure 4.6.

4.4.2.2 Performance drivers of the IVNH

In this section, we analyze main performance drivers of the IVNH. We start by examining the influence of the number of iterations, or so-called convergence behavior of the IVNH.

Recall that the following algorithmic steps constitute an iteration of the IVNH (cf. Section 4.3.4):

- construction of a new incumbent solution with procedures described in Section 4.3.1 or Section 4.3.3,
- local search with respect to neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ (see Section 4.3.2).

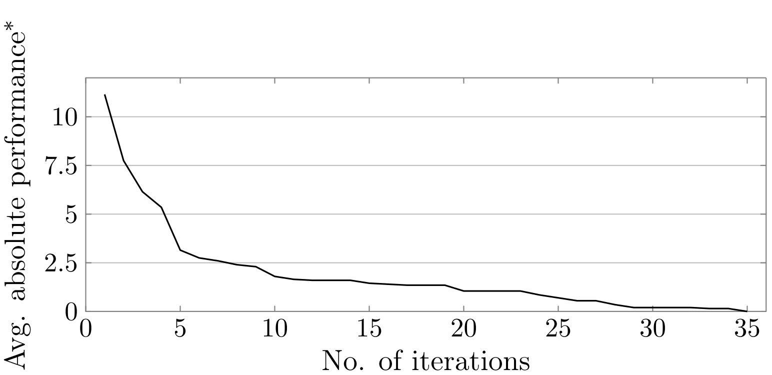


Figure 4.7: Convergence analysis of the basic setting of the LDS instances – (150, 5, [10 30], no, large)

* The average absolute performance of the best incumbent solution found so far

Table 4.7: Average number of iterations until the best solution is found which is not improved in the *Glim* := 20 consequent iterations

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	Avg. number of iterations
100	5	[10 30]	no	large	12.9
⌈	2	[10 30]	no	large	6.3
│	⌈	[0 100]	no	large	11.0
│	│	[0 20]	no	large	1.0
│	│	⌈	no	large	14.4
150	5	[10 30]	yes	large	22.1
│	│	│	no	medium	19.4
│	│	⌊	no	small	16.9
│	│	[20 40]	no	large	11.7
│	⌊	[30 50]	no	large	1.0
⌊	8	[10 30]	no	large	22.3
200	5	[10 30]	no	large	21.4

Figure 4.7 illustrates the convergence behavior for instances of the basic setting (150, 5, [10 30], no, large). For each iteration, the figure shows the average absolute performance of the best solution found until this iteration. As we see, the quality of the best found solution rapidly improves in the first five iterations and continuously ameliorates at a diminishing rate afterward. It is exactly the kind of behavior we would like to have in a good heuristic procedure, because (i) solutions of very good quality are found very fast and (ii) there are excellent chances to compute even better solutions if we relax the stopping criterion and perform more iterations, unless an optimal solution has been already found. Note that we found a similar behavior of the IVNH in other settings as well. For instance, in Table 4.7, we report the average number of iterations until the best solution was found that could not be improved in the *Glim* := 20 consequent iterations. Table 4.7 illustrates that, in concordance with intuition, the flattening-out of the average absolute performance of the IVNH occurs at later iterations for instances with a larger number of jobs or a larger number of dimensions.

We also analyze the impact of different algorithmic elements on the performance of the IVNH in Table 4.8. In the initial algorithm of the IVNH, we disable local search procedures (see Section 4.3.2) one at a time. We also replace our customized diversification procedure (see Section 4.3.3) with a straightforward random generation of a new feasible solution by assigning jobs to the bins by the first-fit procedure (see Section 4.3.1) in a randomly generated order. We call the resulting algorithm as 'Random diversification' in Table 4.8.

According to the received results, local search with respect to neighborhood $\mathcal{N}_1$ has the largest impact on the IVNH performance in almost all the settings. Neighborhood $\mathcal{N}_3$ appears to be especially important in data settings with job sizes [10 30] and [0 100], i.e., data settings in which small jobs are present (the results on setting [0 20] are inconclusive

Table 4.8: Impact of the algorithmic elements of the IVNH on its performance

n	m	$[\underline{v},\overline{v}]$	Release time	Due date	Difference in the average absolute performance to that of the IVNH				CPU ratio to that of the IVNH
					Disabled $\mathcal{N}_1$	Disabled $\mathcal{N}_2$	Disabled $\mathcal{N}_3$	Random diversification	Random diversification
100	5	[10 30]	no	large	5.8	1.1	0.5	0.2	277%
⌈	2	[10 30]	no	large	4.0	1.0	0.7	0.1	313%
│	⌈	[0 100]	no	large	18.5	0.8	4.7	0.1	109%
│	│	[0 20]	no	large	0.0	0.0	0.0	0.1	-*
│	│	⌈	no	large	17.3	1.4	2.5	0.1	233%
150	5	[10 30]	yes	large	14.8	1.0	2.4	0.2	222%
│	│	│	no	medium	64.0	4.8	15.4	2.3	318%
│	│	⌊	no	small	48.2	2.4	15.8	4.5	262%
│	│	[20 40]	no	large	6.0	0.9	1.7	0.6	379%
│	⌊	[30 50]	no	large	0.0	0.0	0.0	0.0	769%
⌊	8	[10 30]	no	large	21.3	2.7	4.0	0.0	253%
200	5	[10 30]	no	large	31.7	1.9	3.6	0.0	172%

* Not defined if the CPU time of the IVNH is 0

because all the examined algorithms found solutions of very good quality). A possible explanation for this effect can be the following. If many small jobs are present, then feasible solutions are likely to have many jobs in each bin and each neighborhood move in $\mathcal{N}_3$ relocates many jobs at once. If the jobs are large in size, however, so that only very few (one, two, or three) jobs can be packed into one bin, then the 'bin moves' of neighborhood $\mathcal{N}_3$ are equivalent to a few (one, two, or three) subsequent swap moves of neighborhood $\mathcal{N}_1$ or a few subsequent reinsertion moves of neighborhood $\mathcal{N}_2$. In such settings, locally optimal solutions with respect to neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ are more likely to be locally optimal with respect to the neighborhood $\mathcal{N}_3$ as well and the relative impact of $\mathcal{N}_3$ in the IVNH performance gets smaller.

The impact of our diversification procedure is twofold. On the one hand, it improves the final solutions (see Table 4.8). On the other hand, it speeds up the IVNH-algorithm several times. In other words, our diversification procedure seems to construct promising new incumbent solutions, whereas the incumbent solutions generated by the random diversification are usually far from the local optima and need more steps of the local search (see also our discussion in Section 4.3.3).

4.4.2.3 Comparison of the two integer programming model formulations

In Table 4.9, we additionally illustrate performance of the strengthened model formulation (4.1)-(4.5) and the basic model formulation (4.1)-(4.4) on the SDS. In this computational experiment, we did not limit the run time of CPLEX. The strengthened model (4.1)-(4.5) reduced computational time of CPLEX significantly compared to the basic model. For example, in setting (30, 5, [30 50], no, large), the computational time was reduced by 69% on average and the number of nodes decreased to 0 for all 20 instances, i.e., all the

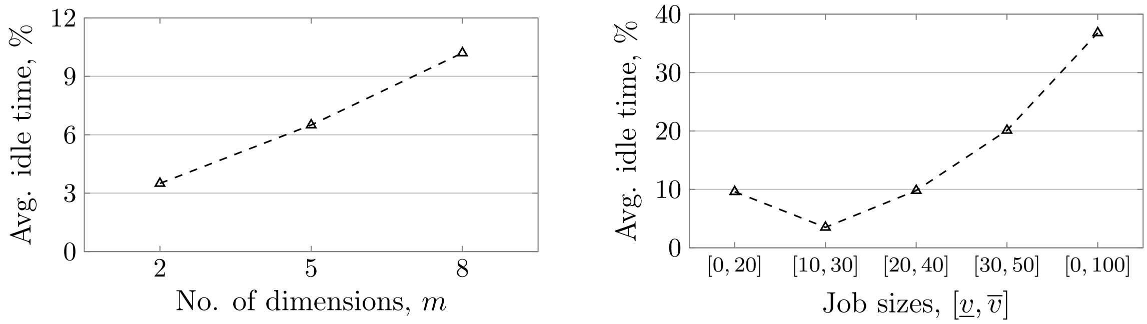
Table 4.9: CPLEX results for the two model formulations

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	Model (4.1)-(4.4)		Model (4.1)-(4.5)	
					Avg. CPU, sec.	Avg. no. of nodes	Avg. CPU, sec.	Avg. no. of nodes
30	2	[10 30]	no	large	0.06	0.0	0.06	0.0
	5	[0 100]	no	large	6.64	25,639.8	6.71	17,734.4
		[0 20]	no	large	0.06	0.0	0.06	0.0
		[10 30]	no	large	0.13	458.1	0.12	328.1
			yes	large	0.23	2,426.4	0.12	353.8
			no	medium	395.88	1,871,376.8	115.84	658,962.1
			no	small	30.26	203,352.1	21.93	133,944.6
		[20 40]	no	large	0.07	0.0	0.08	0.0
		[30 50]	no	large	1.00	1,154.6	0.25	0.0
	8	[10 30]	no	large	15.92	90,686.8	2.68	15,731.2

instances were solved in the root node.

4.4.2.4 Report on the idle times in the found solutions

Of course, the most important objective for the firms that produce customized products is meeting the promised delivery dates. Nevertheless, it is interesting to examine the dynamics of idle times in our computational experiments. Idle times are an important performance indicator of the cost-efficiency of the assembly line. Figures 4.8a and 4.8b illustrate idle times for the basic setting of the LDS instances (150, 5, [10 30], no, large) and how it changes with the number of dimensions (Figures 4.8a) and the job size (Figures 4.8b). Interestingly, the average idle time remains moderate and does not exceed 20% for a large range of settings, which illustrates the efficiency of the multiple-piece-flow manufacturing policy.



(a) The influence of the number of dimensions

(b) The influence of job sizes

Figure 4.8: Sensitivity analysis of the basic setting of the LDS instances– (150, 5, [10 30], no, large): The average idle time

Figures 4.8b illustrates that two opposite effects may influence idle times. On the one hand, the smaller the jobs are, the better we can combine them to fill bins to their capacity and the idle time decreases. The other effect is related to the peculiarity of the total tardiness function: any schedule, in which jobs are finished not later than their due dates, is equally good notwithstanding the amount of the idle time. In other words, if jobs are so small so that we can pack many jobs in one bin and finish them without delay, the idle time may increase as in the setting (150, 5, [0 20], no, large).

4.4.3 Simulation study: The IVNH as part of rolling-horizon planning

We have also examined our algorithm in a case study of the production of industrial cranes.

The aim of the current simulation study is to illustrate the suitability of the tardiness objective function in general and the designed heuristic IVNH in particular for the rolling-horizon planning framework used in the company. Therefore, we compare the IVNH to the status quo planning rule and perform further analysis of its performance in Section 4.4.3.2. In Section 4.4.3.3, we compare the tardiness objective function to another widespread objective function – minimization of the total idle time. We describe the simulation framework in Section 4.4.3.1.

Based on the results in Section 4.4.2, we do not use CPLEX in our simulation experiments for the following two reasons:

- The resulting problem instances that have to be solved within the simulation framework are too large to be solved by CPLEX in a reasonable amount of time. Observe that in the rolling-horizon planning framework, an m-BiPacS instance have to be solved at each replanning step.
- If we set some acceptable run time limit, CPLEX performs worse than the IVNH.

We also conducted some selective preliminary testing, which reconfirmed the reasons stated above.

4.4.3.1 Data generation

To simulate the actual production planning process, we set up a rolling-horizon planning framework: The simulation runs for 15 days and the replanning is performed in the beginning of each day.

The product consists of two large workpieces – a pillar and a jib –, each of them is processed in a separate assembly line consisting of four and five stations, respectively (see Figure 4.2). Workpieces should be cleansed, trimmed to the desired size, drilled, and welded. Each process requires a specialized machine and processing times highly depend on the customer orders, such as on the length and diameter of the pillars or jibs and on the ordered fixtures. Afterward, the two workpieces are joined and processed together at the final station. So that there are $4 + 5 + 1 = 10$ dimensions in total. A bundle of workpieces (a bin of jobs) enters each station in the beginning of the day and leaves the station in the beginning of the next day. There is a buffer before the final station to compensate for different lengths of the two assembly lines. The work on each station is organized in shifts, and bottleneck stations work several shifts per day, whereas the rest stations work just one shift or a half shift a day. We have taken these different shift schedules into account by an appropriate scaling of the dimensions' capacities c and job sizes v_{ji}.

We have setup the arrival process of the jobs to mimic the actual company data, for instance, in the expected number of the arriving jobs each day and in the distribution of different crane models. Following the practice data, we also start the simulation with a four-day backlog of jobs, i.e., the jobs arrived in the last five days (including the current day) are available for immediate processing. In the beginning of the next day, the set of available jobs reduces by the bin load whose manufacture has already started and increases by the newly arrived jobs; a new plan is made for the now available jobs. Because we simulate the production process for 15 days, we perform 15 replannings in total. At each replanning step of the rolling-horizon planning framework, all the available jobs are scheduled. Because the simulated system is in the steady state, the number of the scheduled jobs roughly corresponds to the work amount for the next five days. Thereby, in the default setting (in contrast to the setting with full information), the planner only knows the characteristics of the already arrived jobs. In each simulation setting, we generate 100 simulation instances (runs) and report the average results obtained in these runs. We have also carefully checked the simulated data for possible warm-up and phasing-out effects. As a consequence, we have truncated the last four bins in the simulated data in the idle time experiment (Figure 4.10a in Section 4.4.3.3). Indeed, because of the phasing-out effect, less jobs were available for planning and four last bins had moderately larger idle times.

We formulate several simulation settings to examine different promised delivery times u: $d_j := r_j + u$, where r_j is the arrival time of the job. The initial level of u, denoted as u_0, was calibrated to the current policy of the company – the time between the final specification of the job (i.e., crane characteristics) and the latest acceptable start of production given the promised delivery time. At the request of the company management, we also examine the values of $u := u_0 + 1$ and $u := u_0 + 2$ in our experiments, which correspond to later promised delivery times.

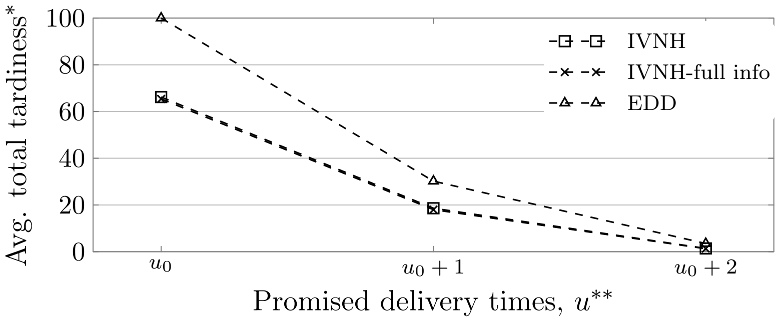


Figure 4.9: Results of the simulation study

* The objective value computed by the EDD for $u := u_0$ is set to 100.

** The values of u are anonymized, the minimal value is denoted as u_0.

4.4.3.2 Tardiness objective function in the rolling-horizon planning framework

In this section, we examine the suitability of the set up optimization problem with the tardiness objective function for the rolling-horizon planning framework used in the company. For instance, to investigate the improvement of the status quo, we compare the results of the IVNH to the earliest due date (EDD) heuristic (see Section 4.3.1) which is common in practice and roughly describes the currently used planning routine in the company. Recall that in the rolling-horizon planning framework, the information is incomplete: The planner only knows the characteristics of the jobs that have already arrived and knows nothing about the jobs that are about to enter the system in the future. Therefore, we also perform the *competitive analysis*, i.e., we compare the results of the IVNH in this incomplete-information framework to the results of the IVNH in the full-information case (designated as the *IVNH-full info*). The latter schedules jobs in the beginning of day 1 with a full information about all the future jobs and their release times (i.e., times of arrival).

Figure 4.9 illustrates the average objective value (tardiness) in the solutions found by the IVNH, EDD, and the IVNH-full info. To anonymize the data, we set the average delay per job for the EDD at initial value u_0 to 100.

As expected, the IVNH significantly outperforms the EDD, especially at shorter promised delivery times. If the promised delivery times are large and set at $u := u_0 + 2$, the jobs can be easily manufactured without tardiness and the advantage of the IVNH over the EDD vanishes out.

Interestingly, the value of full information, which is the difference in the objective value computed with the IVNH-full info compared to the results of the IVNH, is modest, so that the tardiness objective function performs very well in the rolling-horizon procedure for the studied company.

Overall, the total tardiness in Figure 4.9 is nonlinear in u: it increases sharply if the promised delivery times u become short. Indeed, in case of less restrictive promised delivery times, jobs may be combined more freely into bins, so that more jobs can be produced in the given time period. We have also observed that at low values of u large urgent jobs are pushed back into the later bins in order to fill earlier bins with many smaller jobs.

4.4.3.3 Comparison of the tardiness and the idle-time objective functions

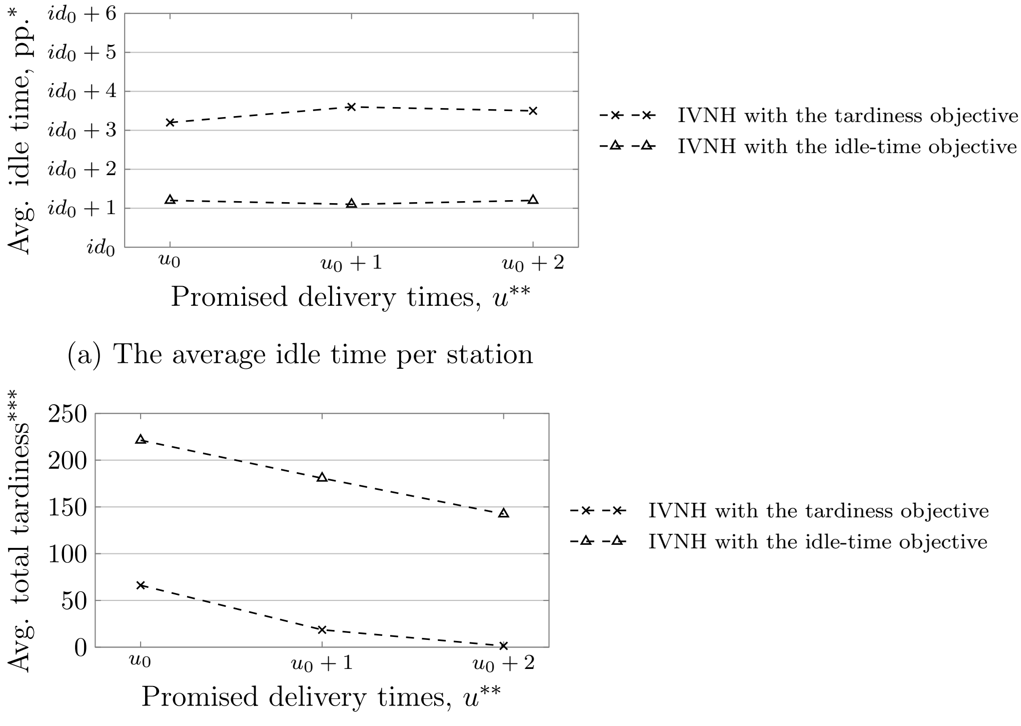
In this section, we compare the tardiness objective function to the idle time objective function in the rolling-horizon planning framework. The minimization of the average idle time per station (i.e., per dimension) leads to lower manufacturing costs per product unit. This is a widespread objective function in the assembly line balancing and bin packing literature. Observe that because the sizes of each bin dimension and the sizes of the jobs are given, the minimization of the average idle time per station is equivalent to the minimization of the total idle time and to the minimization of the number of the bins.

The adaptation of the IVNH algorithm from Section 4.3 to the idle-time objective function is straightforward and simply amounted to using the new objective function in the respective algorithmic routines, such as selection of an improving neighbor, update of the best found solution etc. Observe, however, that the local search with respect to neighborhood $\mathcal{N}_3$ turns obsolete, since we cannot improve the average idle time by resequencing the bin loads.

Figure 4.10a compares the average idle time per station in case the rolling-horizon planning uses the IVNH with the tardiness and with the idle-time objective functions, respectively. As expected, the average idle time in the latter case is lower. But the degree of improvement, which equals about 2.3 percentage points, is considered to be very moderate by the company management.

The average total tardiness of the manufactured jobs is presented in Figure 4.10b. As we see, the average total tardiness is extremely high in case the rolling-horizon planning uses the IVNH with the idle-time objective function, for instance, it is about 2.3 times larger compared to the status-quo rule EDD and $u := u_0$. Indeed, the solution algorithm often pushes some urgent jobs to the ever later bins. Interestingly, the average tardiness remains very high even if the promised delivery times are set large, at $u := u_0 + 2$.

We conclude that the idle time objective function is not suitable for the rolling-horizon planning framework.



(a) The average idle time per station

(b) The average total tardiness

Figure 4.10: Comparison of two different objective functions in the rolling-horizon planning framework: the minimization of the total tardiness and the minimization of the total idle time

* The data is anonymized, the scale starts with some initial idle time level id_0.

** The values of u are anonymized, the minimal value is denoted as u_0.

*** The objective value computed by the EDD for $u := u_0$ is set to 100.

4.5 Conclusion

This paper studies product sequencing in paced multiple-piece-flow assembly lines that are used, for instance, by medium-sized companies in manufacturing large pieces of customized equipment. We set up a mixed-integer model and analyze the relation of the problem, which we call m-vector bin packing and sequencing problem (m-BiPacS), to classical optimization problems in the literature. We design an iterative variable neighborhood search metaheuristic, the IVNH, to address large problem instances of the m-BiPacS. The IVNH utilizes several innovative local search procedures that examine many distinct good-quality solutions with maximally packed bins in a short time. In our computational experiments based on well-diversified randomly generated data sets, the IVNH clearly outperformed the off-the-shelf optimization tool IBM ILOG CPLEX: It found better solutions and had much shorter run times. For smaller problem instances with known optimal solutions, the IVNH was able to find an optimal solution in a fraction of a second in almost all cases. The gap to optimality in the remaining cases was negligible. An extended simulation based on real-world data confirms suitability of the IVNH and the tardiness objective function for the rolling-horizon planning framework in practice.

Future studies may develop exact solution methods that can solve instances of practice-relevant size in acceptably short run times. Research is needed on related planning problems, such as assembly line balancing and committing order due dates in multiple-piece-assembly lines. In particular, our simulation illustrates the importance of accurately estimated promised delivery dates. Promised delivery dates that are too restrictive prohibit a favorable combination of jobs (workpieces) into bins (lots that are launched simultaneously) and manufacturing cost may increase significantly. Overall, companies need the support of academia in developing further organizational concepts and planning tools that help to reduce manufacturing costs in the production of highly customized products. For instance, future field and case studies may investigate whether the performance of multiple-piece-flow assembly lines can be further improved by flexible deployment of machines (robots) and workers and more flexible flow of workpieces.

4.A Appendix A

In this appendix, we explain the relation of $\mathcal{N}_1$ to the conventional two-job exchange neighborhood.

Recall that we get neighbor s'_c of some solution s in the conventional two-job exchange neighborhood by swapping two jobs belonging to two different bins such that the resulting solution is feasible. We say that a *subsequent sequential shift of jobs* is performed to solution s_c', when we shift the jobs belonging to some bin to the earliest possible bins to which they can be assigned; we apply this procedure to all the bins in the increasing order of the bin number. We call the resulting solution s_c''. For example, consider the instance in Table 4.3 and $s := \{1, 2\}\{3, 4\}\{5, 6\}\{7, 8\}$. Feasible solution $s_c' = \{1, \mathbf{8}\}\{3, 4\}\{5, 6\}\{7, \mathbf{2}\}$ is a neighbor solution of s with respect to the conventional two-job exchange neighborhood. After the subsequent sequential shift of jobs, we receive $s_c'' = \{1, \mathbf{5}, 8\}\{3, 4\}\{6, \mathbf{7}\}\{2\}$.

Lemma 4.A.1. *Neighborhood $\mathcal{N}_1$ of some feasible solution s contains all the neighbor solutions of the conventional two-job exchange neighborhood of s that are improved by subsequent sequential shift of jobs to receive maximally packed bins. In other words, for each neighbor solution s_c' of s with respect to the conventional two-job exchange neighborhood, there exists a feasible solution s_c'' that is part of neighborhood $\mathcal{N}_1(s)$ and that can be constructed from s_c' with the subsequent sequential shift of jobs.*

Proof. The proof follows straightforwardly from the definition of $\mathcal{N}_1$. Indeed, to construct $s'_{\mathcal{N}_1} \in \mathcal{N}_1(s)$, we exchange job j with some job j' that belongs to another bin in s in steps 1 and 2 in Figure 4.3(ii). The subsequent application of the first-fit procedure can be interpreted as the subsequent sequential shift of jobs. □

4.B Appendix B

In this appendix, we perform a computational experiment on the LDS instances (see Section 4.4.1) to compare the number of neighbors in $\mathcal{N}_1$ and the conventional two-job exchange neighborhood.

Table 4.B1: Comparison of the $\mathcal{N}_1$ neighborhood and the conventional two-job exchange neighborhood

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	$\mathcal{N}_1$			Conventional two-job exchange neighborhood			
					Avg. total No. of neighbors	Avg. No. of distinct neighbors	Avg. absolute performance of the best neighbor	Avg. total No. of neighbors	Avg. No. of distinct neighbors*	Avg. No. of distinct max. loaded neighbors**	Avg. absolute performance of the best neighbor
100	5	[10 30]	no	large	4,950.0	4,703.1	145.4	2,186.1	2,186.1	2,180.2	161.8
⌈	2	[10 30]	no	large	11,175.0	10,739.0	253.0	3,997.0	3,997.0	3,988.0	282.6
|	⌈	[0 100]	no	large	11,175.0	10,743.3	2,209.4	4,759.2	4,759.2	4,759.2	2,280.8
|	|	[0 20]	no	large	11,175.0	9,940.7	89.3	1,008.9	1,008.9	1,000.8	109.6
|	|	⌈	no	large	11,175.0	10,788.5	340.5	4,630.1	4,630.1	4,620.8	371.7
150	5	[10 30]	yes	large	11,175.0	10,286.5	325.4	3,628.0	3,628.0	3,620.7	365.3
|	|	|	no	medium	11,175.0	10,785.3	868.2	4,432.5	4,432.5	4,424.8	924.2
|	|	⌊	no	small	11,175.0	10,782.9	1,551.2	4,285.7	4,285.7	4,279.9	1,610.4
|	|	[20 40]	no	large	11,175.0	10,846.6	406.9	2,439.0	2,439.0	2,438.4	421.5
|	⌊	[30 50]	no	large	11,175.0	11,100.0	933.5	11,100.0	11,100.0	11,100.0	933.5
⌊	8	[10 30]	no	large	11,175.0	10,816.1	394.3	5,077.8	5,077.8	5,072.2	421.9
200	5	[10 30]	no	large	19,900.0	19,372.4	601.9	8,164.5	8,164.5	8,153.5	657.4

* The neighbor solutions with *not* maximally packed bins are also included
** After subsequent sequential shift of jobs

We construct an initial solution with the PRBM heuristic described in Section 4.3.1, afterward we enumerate all the neighbors of this solution in the respective neighborhood. Observe that some neighbors in $\mathcal{N}_1$ may represent essentially identical solutions, therefore, we also compute the number of distinct neighbors. Table 4.B1 summarizes the results.

Overall, neighborhood $\mathcal{N}_1$ contained 242.6% more neighbors and 221.8% more distinct neighbors on average than the conventional two-job exchange neighborhood. Recall that neighbors in the conventional neighborhood may contain *not* maximally packed bins. After we applied a subsequent sequential shift of jobs to the neighbors in the conventional neighborhood (cf. Appendix 4.A), the number of distinct solutions reduced even further. As a consequence, average absolute performance of the best neighbor was 8.5% worse in the conventional neighborhood, even after the subsequent sequential shift of jobs had been applied.

This computational experiment confirmed that $\mathcal{N}_1$ tend to contain more neighbors and, therefore, its best improving neighbor may have a better objective function value than the conventional two-job exchange neighborhood.

Chapter 5

Summary and Outlook

This dissertation summarizes several papers of operational planning problems that are crane scheduling problems at rail transshipment yards and a sequencing problem in assembly lines.

For each problem, we propose a model formulation and develop exact and/or heuristic algorithms that can solve practice-relevant sized instances. Most of our test instances are randomly generated based on real data from practice, as much as possible to simulate the practical situation.

A crane scheduling problem at rail container transshipment yards is considered in Chapter 2. This problem addresses the working areas for each crane and the sequence of containers' moves. The objective is to minimize the makespan. We formulate the problem as a mixed-integer model and propose a two-way bounded dynamic programming (TBDP) procedure. We also propose an exact algorithm and a heuristic approach within the TBDP framework. The computational study results indicate that the heuristic approach obtains high-quality solutions in a short time, and the exact algorithm can solve some instances of real-life size in a reasonable time. We observe that it improves efficiency up to 14% when considering yard partitioning and container scheduling together. Future research could consider a more flexible range of container movements, e.g., containers can pass through more than one crane working area by introducing sorter vehicles. Furthermore, dynamic policy for the crane zone may further improve the makespan, i.e., the crane working zones can overlap.

In chapter 3, we address an asymmetric traveling salesman problem with a makespan objective function, namely the single crane scheduling problem, which is also a subproblem of the crane scheduling problem in chapter 2. We propose a dynamic branch-and-cut algorithm (DBC) with new separation routines. The DBC is, to the best of our knowledge, the first branch and cut based algorithm for such a problem. We also design a decomposition algorithm (DA) specifically for the makespan objective, which is a metastructure that can be integrated with other algorithms. In a computational study, we observe that the DA reduces the run time significantly for many test instances, and the combined algorithm DBC&DA is able to solve all the instances of our real-life simulations. Future research could consider improving this algorithm's efficiency further, e.g., consider combining more advantages of other exact algorithms with DBC.

Finally, Chapter 4 focuses on the product sequencing problem in paced multiple-piece-flow assembly lines, namely m-vector bin packing and sequencing problem. We intensely study the properties of the problem and examine its relation to several classic optimization problems. We present a mixed-integer model formulation solved by IBM ILOG Cplex and design an iterative variable neighborhood heuristic (IVNH). We compare the performance of IVNH and Cplex. For smaller test instances, the IVNH is able to find an optimal solution on the millisecond time scale for almost all cases; for real-life sized instances, it clearly outperformed Cplex with better solutions and shorter run times. Furthermore, the calculation results show the importance of reasonable promised delivery dates. Specifically, sometimes tight delivery times make some better combinations of workpieces impossible. Further work needs to be done to develop an efficient exact algorithm that can solve instances of real-life size in acceptable run times.

Bibliography

Alicke, K. (2005). Modeling and optimization of the intermodal terminal mega hub. In Günther, H. O. and Kim, K. H., editors, Container Terminals and Automated Transport Systems, pages 307–323. Springer, Berlin, Heidelberg.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. European Journal of Operational Research, 246(2):345–378.

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. Omega, 27(2):219–239.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 187(3):985–1032.

Aloulou, M. A., Bouzaiene, A., Dridi, N., and Vanderpooten, D. (2014). A bicriteria two-machine flow-shop serial-batching scheduling problem with bounded batch size. Journal of Scheduling, 17:17–29.

Alsoufi, G., Yang, X., and Salhi, A. (2015). A combinatorial benders' cuts approach to the seaside operations problem in container ports. The XI Metaheuristics International Conference.

Alves, C., de Carvalho, J. V., Clautiaux, F., and Rietz, J. (2014). Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. European Journal of Operational Research, 233(1):43–63.

Ascheuer, N. (1996). Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. TechReport TR 96-03, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf.

Ascheuer, N., Fischetti, M., and Grötschel, M. (2001). Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. Mathematical Programming, 90:475–506.

Baker, E. K. (1983). An exact algorithm for the time constrained traveling salesman problem. Operations Research, 31(5):938–945.

Ballis, A. and Golias, J. (2002). Comparative evaluation of existing and innovative rail-road freight transport terminals. Transportation Research, 36(7):593–611.

Barketau, M., Kopfer, H., and Pesch, E. (2013). A Lagrangian lower bound for the container transshipment problem at a railway hub for a fast branch-and-bound algorithm. Journal of the Operational Research Society, 64(11):1614–1621.

Barketau, M. and Pesch, E. (2016). An approximation algorithm for a special case of the asymmetric traveling salesman problem. International Journal of Production Research, 54(14):4205–4212.

Barketau, M., Pesch, E., and Shafransky, Y. (2015). Minimizing maximum weight of subsets of a maximum matching in a bipartite graph. Discrete Applied Mathematics, 196(11):4–19.

Barketau, M., Pesch, E., and Shafransky, Y. (2016). Scheduling dedicated jobs with variative processing times. Journal of Combinatorial Optimization, 31:774–785.

Battaïa, O. and Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. International Journal of Production Economics, 142(2):259 – 277.

Bautista, J., Alfaro-Pozo, R., and Batalla-García, C. (2016). GRASP for sequencing mixed models in an assembly line with work overload, useless time and production regularity. Progress in Artificial Intelligence, 5:27–33.

Baybars, İ. (1986). A survey of exact algorithms for the simple assembly line balancing problem. Management Science, 32(8):909–932.

Becker, C. and Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. European Journal of Operational Research, 168(3):694–715.

Bellman, R. (1954). The theory of dynamic programming. Bulletin of the American Mathematical Society, 60:503–515.

Bertsimas, D. and Demir, R. (2002). An approximate dynamic programming approach to multidimensional knapsack problems. Management Science, 48(4):550–565.

Bianco, L., Ricciardelli, S., Rinaldi, G., and Sassano, A. (1988). Scheduling tasks with sequence-dependent processing times. Naval Research Logistics, 35(2):177–184.

Bierwirth, C. and Meisel, F. (2010). A survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research, 202(3):615–627.

Bierwirth, C. and Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research, 244(3):675–689.

Bigras, L.-P., Gamache, M., and Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optimization, 5(4):685–699.

BMVI (2014). Verkehrsverflechtungsprognose 2030. ResReport 96.0981/2011, Bundesministerium für Verkehr und digitale Infrastruktur (Federal Ministry for Transport and Digital Infrastructure).

Bock, S., Rosenberg, O., and Brackel, T. V. (2006). Controlling mixed-model assembly lines in real-time by using distributed systems. European Journal of Operational Research, 168(3):880–904.

Bostel, N. and Dejax, P. (1998). Models and algorithms for container allocation problems on trains in a rapid transshipment shunting yard. Transportation Science, 32(4):370–379.

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. and Fliedner, M. (2010). Determining crane areas in intermodal transshipment yards: The yard partition problem. European Journal of Operational Research, 204(2):336–342.

Boysen, N., Fliedner, M., Jaehn, F., and Pesch, E. (2013). A survey on container processing in railway yards. Transportation Science, 47(3):312–329.

Boysen, N., Fliedner, M., and Kellner, M. (2010). Determining fixed crane areas in rail-rail transshipment yards. Transportation Research Part E: Logistics and Transportation Review, 46(6):1005–1016.

Boysen, N., Fliedner, M., and Scholl, A. (2007). A classification of assembly line balancing problems. European Journal of Operational Research, 183(2):674–693.

Boysen, N., Fliedner, M., and Scholl, A. (2008). Assembly line balancing: Which model to use when? International Journal of Production Economics, 111(2):509–528.

Boysen, N., Fliedner, M., and Scholl, A. (2009a). Production planning of mixed-model assembly lines: Overview and extensions. Production Planning & Control, 20(5):455–471.

Boysen, N., Fliedner, M., and Scholl, A. (2009b). Sequencing mixed-model assembly lines: Survey, classification and model critique. European Journal of Operational Research, 192(2):349–373.

Boysen, N., Jaehn, F., and Pesch, E. (2011). Scheduling freight trains in rail-rail transshipment yards. Transportation Science, 45(2):199–211.

Boysen, N., Jaehn, F., and Pesch, E. (2012). New bounds and algorithms for the transshipment yard scheduling problem. Journal of Scheduling, 15:499–511.

Brucker, P. (2007). Scheduling Algorithms. Springer: Berlin, Heidelberg.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C., Tautenhahn, T., and van de Velde, S. (1998). Scheduling a batching machine. Journal of Scheduling, 1(1):31–54.

Bruns, F. and Knust, S. (2012). Optimized load planning of trains in intermodal transportation. OR Spectrum, 34:511–533.

Bukchin, J. (1998). A comparative study of performance measures for throughput of a mixed model assembly line in a JIT environment. International Journal of Production Research, 36(10):2669–2685.

Burkard, R., Dell'Amico, M., and Martello, S. (2009). Assignment Problems. Society for Industrial and Applied Mathematics.

Buxey, G. M., Slack, N. D., and Wild, R. (1973). Production Flow Line System Design-A Review. AIIE Transactions, 5(1):37–48.

Cabo, M., González-Velarde, J. L., Possani, E., and Solís, Y. Á. R. (2018). Bi-objective scheduling on a restricted batching machine. Computers & Operations Research, 100:201–210.

Cabo, M., Possani, E., Potts, C. N., and Song, X. (2015). Split-merge: Using exponential neighborhood search for scheduling a batching machine. Computers & Operations Research, 63:125–135.

Camati, R. S., Calsavara, A., and Jr., L. L. (2014). Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. In Gyires, T., Westphall, C. B., and Kálmán, G., editors, ICN 2014: The Thirteenth International Conference on Networks, pages 253–260.

Caprara, A., Kellerer, H., and Pferschy, U. (2003). Approximation schemes for ordered vector packing problems. Naval Research Logistics, 50(1):58–69.

Caprara, A. and Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. Discrete Applied Mathematics, 111(3):231–262.

Chang, S. Y., Hwang, H.-C., and Park, S. (2005). A two-dimensional vector packing model for the efficient use of coil cassettes. Computers & Operations Research, 32(8):2051–2058.

Cheng, T. C. E. and Kovalyov, M. Y. (2001). Single machine batch scheduling with sequential job processing. IIE Transactions, 33:413–420.

Chou, F. D., Wang, H. M., and Chang, T. Y. (2009). Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups. The International Journal of Advanced Manufacturing Technology, 43:810–821.

Christensen, H. I., Khan, A., Pokutta, S., and Tetali, P. (2016). Multidimensional bin packing and other related problems: A survey. https://people.math.gatech.edu/ tetali/PUBLIS/CKPT.pdf.

Christofides, N., Mingozzi, A., and Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. Networks, 11(2):145–164.

Cichenski, M., Jaehn, F., Pawlak, G., Pesch, E., Singh, G., and Blazewicz, J. (2017). An integrated model for the transshipment yard scheduling problem. Journal of Scheduling, 20:57–65.

Cire, A. A. and van Hoeve, W. J. (2013). Multivalued decision diagrams for sequencing problems. Operations Research, 61(6):1411–1428.

Copil, K., Wörbelauer, M., Meyr, H., and Tempelmeier, H. (2017). Simultaneous lotsizing and scheduling problems: a classification and review of models. OR Spectrum, 39:1–64.

Corry, P. and Kozan, E. (2006). An assignment model for dynamic load planning of intermodal trains. Computers & Operations Research, 33(1):1–17.

Corry, P. and Kozan, E. (2008). Optimised loading patterns for intermodal trains. OR Spectrum, 30:721–750.

Cortez, P. M. C. and Costa, A. M. (2015). Sequencing mixed-model assembly lines operating with a heterogeneous workforce. International Journal of Production Research, 53(11):3419–3432.

Cunha, C. B. and Swait, J. (2000). New dominance criteria for the generalized permanent labelling algorithm for the shortest path problem with time windows on dense graphs. International Transactions in Operational Research, 7(2):139–157.

Davis, S. M. (1990). Future perfect. In Evans, P., Doz, Y., and Laurent, A., editors, Human Resource Management in International Firms. Palgrave Macmillan, London.

de Jong, G., Vierth, I., Tavasszy, L., and Ben-Akiva, M. (2013). Recent developments in national and international freight transport models within europe. Transportation, 40:347–371.

Desrosiers, J., Soumis, F., and Desrochers, M. (1984). Routing with time windows by column generation. Networks, 14(4):545–565.

Dik, G. and Kozan, E. (2017). A flexible crane scheduling methodology for container terminals. Flexible Services and Manufacturing Journal, 29:64–96.

DIOMIS (2007). International Combined Transport Production Systems including long and heavy trains (Workpackage A7). ResReport, Developing Infrastructure and Operation Models for Intermodal Shift.

DIOMIS (2009). Benchmarking Intermodal Rail Transport in the United States and Europe. ResReport, Developing Infrastructure and Operation Models for Intermodal Shift.

Dolgui, A., Guschinsky, N., and Levin, G. (2006). A special case of transfer lines balancing by graph approach. European Journal of Operational Research, 168(3):732–746.

Dörmer, J., Günther, H. O., and Gujjula, R. (2015). Master production scheduling and sequencing at mixed-model assembly lines in the automotive industry. Flexible Services and Manufacturing Journal, 27:1–29.

Dumas, Y., Desrosiers, J., Gelinas, E., and Solomon, M. M. (1995). An optimal algorithm for the traveling salesman problem with time windows. Operations Research, 43(2):367–371.

DUSS (2010). Duss - Daten & Fakten. TechReport, Deutsche Umschlaggesellschaft Schiene - Straße (DUSS) mbH.

DUSS (2015). Duss - Terminals in Überblick. TechReport, Deutsche Umschlaggesellschaft Schiene - Straße (DUSS) mbH.

Epstein, L. and Levin, A. (2012). Bin packing with general cost structures. Mathematical Programming, 132:355–391.

Erel, E. and Sarin, S. C. (1998). A survey of the assembly line balancing procedures. Production Planning & Control, 9(5):414–434.

EU Commission (2007). Mitteilung der Kommission an den Rat und das Europäische Parlament über die Überwachung der Entwicklung des Schienenverkehrsmarkts. TechReport, European Commission.

EU Commission (2011). WHITE PAPER: Roadmap to a Single European Transport Area - Towards a competitive and resource efficient transport system. TechReport, European Commission.

EU Commission (2015). Analysis of the EU combined transport. TechReport, European Commission.

Faccio, M., Gamberi, M., and Bortolini, M. (2016). Hierarchical approach for paced mixed-model assembly line balancing and sequencing with jolly operators. International Journal of Production Research, 54(3):761–777.

Fleischmann, B. (1990). The discrete lot-sizing and scheduling problem. European Journal of Operational Research, 44(3):337–348.

Fleischmann, B. and Meyr, H. (1997). The general lotsizing and scheduling problem. OR Spektrum, 19(1).

Froyland, G., Koch, T., Megow, N., Duane, E., and Wren, H. (2008). Optimizing the landside operation of a container terminal. OR Spectrum, 30:53–75.

Fréville, A. (2004). The multidimensional 0-1 knapsack problem: An overview. European Journal of Operational Research, 155(1):1–21.

Gabay, M. and Zaourar, S. (2016). Vector bin packing with heterogeneous bins: application to the machine reassignment problem. Annals of Operations Research, 242:161–194.

Gaidzik, M., Karcher, S., Riebe, E., Mertel, R., Petri, K., Präg, V., and Sondermann, K.-U. (2012). Erstellung eines Entwicklungskonzeptes KV 2025 in Deutschland als Entscheidungshilfe für die Bewillingungsbehörden: Abschlussbericht. ResReport 96.0981/2011, Bundesministerium für Verkehr und digitale Infrastruktur (Federal Ministry for Transport and Digital Infrastructure).

Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C. (1976). Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory: Series A, 21:257–298.

Gavish, B. and Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. Mathematical Programming, 31:78–105.

Ghosh, S. and Gagnon, R. J. (1989). A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research, 27(4):637–670.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. Decision Sciences, 8(1):156–166.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 5:287–326.

Haase, K. (1996). Capacitated lot-sizing with sequence dependent setup costs. OR Spektrum, 18:51–59.

Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics, 10(1):196–210.

Heßler, K., Gschwind, T., and Irnich, S. (2018). Stabilized branch-and-price algorithms for vector packing problems. European Journal of Operational Research, 271(2):401–419.

Hitchcock, F. L. (1941). The distribution of a product from several sources to numerous localities. Journal of Mathematical Physics, 20:224–230.

Hu, Q., Wei, L., and Lim, A. (2018). The two-dimensional vector packing problem with general costs. Omega, 74:59–69.

Ivanov, E. N., Pogromsky, A. Y., and Rooda, J. E. (2009). Scheduling with sequence dependent setup times in application to magnetic resonance imaging scans processing. In 2009 IEEE Control Applications, (CCA) Intelligent Control, (ISIC), pages 867–872.

Jaber, M. Y., editor (2016). Learning Curves: Theory, Models, and Applications. CRC Press.

Jackson, J. R. (1956). A computing procedure for a line balancing problem. Management Science, 2(3):261–271.

Jamal, J., Shobaki, G., Papapanagiotou, V., Gambardella, L., and Montemanni, R. (2017). Solving the sequential ordering problem using branch and bound. In 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1–9.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). Knapsack Problems. Springer-Verlag Berlin Heidelberg.

Kellner, M., Boysen, N., and Fliedner, M. (2012). How to park freight trains on rail-rail transshipment yards: the train location problem. OR Spectrum, 34:535–561.

Kille, C. and Schmidt, N. (2008). Wirtschaftliche Rahmenbedingungen des Güterverkehrs: Studie zum Vergleich der Verkehrsträger im Rahmen des Logistikprozesses in Deutschland. Fraunhofer IRB Verlag, Nürnberg, Germany.

Kim, J. G. and Lee, D. H. (2009). Algorithms for common due-date assignment and sequencing on a single machine with sequence-dependent setup times. Journal of the Operational Research Society, 60(9):1264–1272.

Kim, K. H. and Park, Y. M. (2004). A crane scheduling method for port container terminals. European Journal of Operational Research, 156(3):752–768.

Kleinschmidt, P. and Schannath, H. (1995). A strongly polynomial algorithm for the transportation problem. Mathematical Programming, 68:1–13.

Klindworth, H., Otto, C., and Scholl, A. (2012). On learning precedence graph concept for the automotive industry. European Journal of Operational Research, 217(2):259–269.

Kovalyov, M. Y., Pesch, E., and Ryzhikov, A. (2018). A note on scheduling container storage operations of two non-passing stacking cranes. Networks, 71(3):271–280.

Kozan, E. (1997). Increasing the operational efficiency of container terminals in australia. Journal of the Operational Research Society, 48(2):151–161.

Kress, D., Meiswinkel, S., and Pesch, E. (2015). The partitioning min-max weighted matching problem. European Journal of Operational Research, 247(3):745–754.

Laabadi, S., Naimi, M., Amri, H. E., and Achchab, B. (2018). The 0/1 multidimensional knapsack problem and its variants: A survey of practical models and heuristic approaches. American Journal of Operations Research, 8(5):295–439.

Langevin, A., Desrochers, M., Desrosiers, J., Gelinas, S., and Soumis, F. (1993). A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. Networks, 23(7):631–640.

Lapierre, S. D., Ruiz, A., and Soriano, P. (2006). Balancing assembly lines with tabu search. European Journal of Operational Research, 168(3):826–837.

Leung, J. Y.-T. and Li, C.-L. (2008). An asymptotic approximation scheme for the concave cost bin packing problem. European Journal of Operational Research, 191(2):582–586.

Li, C.-L. and Chen, Z.-L. (2006). Bin-packing problem with concave costs of bin utilization. Naval Research Logistics, 53(4):298–308.

Li, X., Otto, A., and Pesch, E. (2019). Solving the single crane scheduling problem at rail transshipment yards. Discrete Applied Mathematics, 264:134–147.

Luo, X. and Chu, C. (2007). A branch-and-bound algorithm of the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness. European Journal of Operational Research, 180(1):68–81.

Luo, X., Chu, C., and Wang, C. (2006). Some dominance properties for single-machine tardiness problems with sequence-dependent setup. International Journal of Production Research, 44(17):3367–3378.

Luo, X. and Chu, F. (2006). A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness. Applied Mathematics and Computation, 183(1):575–588.

Martello, S. and Toth, P. (1990). Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons.

Mayrhofer, W., März, L., and Sihn, W. (2013). Simulation-based optimization of personnel assignment planning in sequenced commercial vehicle assembly: A software tool for iterative improvement. Journal of Manufacturing Systems, 32(3):423–428.

Mönch, L., Fowler, J., Dauzère-Pérès, S., Mason, S. J., and Rose, O. (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. Journal of Scheduling, 14:583–599.

Montemanni, R., Mojana, M., Caro, G. A. D., and Gambardella, L. M. (2013). A decomposition-based exact approach for the sequential ordering problem. Journal of Applied Operational Research, 5(1):2–13.

Montemanni, R., Smith, D. H., Rizzoli, A. E., and Gambardella, L. M. (2010). Sequential ordering problems for crane scheduling in port terminals. International Journal of Simulation and Process Modelling, 5(12):348–361.

Mosadegh, H., Ghomi, S. M. T. F., and Süer, G. A. (2017). A control theoretical modelling for velocity tuning of the conveyor belt in a dynamic mixed-model assembly line. International Journal of Production Research, 55(24):7473–7495.

Nesello, V., Subramanian, A., Battarra, M., and Laporte, G. (2018). Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times. European Journal of Operational Research, 266(2):498–507.

Ng, V. T. Y., Chan, B., Shun, L. L. Y., and Tsang, R. (2008). Quality service assignments for role-based web services. In 2008 IEEE International Conference on Systems, Man and Cybernetics, pages 2219–2224.

Ng, W. C. and Mak, K. L. (2005). Yard crane scheduling in port container terminals. Applied Mathematical Modelling, 29(3):263–276.

Nof, S. Y., Wilhelm, W., and Warnecke, H. (1997). Industrial Assembly. Springer Science & Business Media.

Nossack, J., Briskorn, D., and Pesch, E. (2018). Container dispatching and conflict-free yard crane routing in an automated container terminal. Transportation Science, 52(5):1059–1076.

Nossack, J. and Pesch, E. (2014). A branch-and-bound algorithm for the acyclic partitioning problem. Computers & Operations Research, 41:174–184.

Otto, A. and Battaïa, O. (2017). Reducing physical ergonomic risks at assembly lines by line balancing and job rotation: A survey. Computers & Industrial Engineering, 111:467–480.

Otto, A. and Li, X. (2020). Product sequencing in multiple-piece-flow assembly lines. Omega, 91:102055.

Otto, A., Li, X., and Pesch, E. (2017). Two-way bounded dynamic programming approach for operations planning in transshipment yards. Transportation Science, 51(1):325–342.

Otto, A. and Otto, C. (2014a). How to design effective priority rules: Example of simple assembly line balancing. Computers & Industrial Engineering, 69:43–52.

Otto, A. and Scholl, A. (2011). Incorporating ergonomic risks into assembly line balancing. European Journal of Operational Research, 212(2):277–286.

Otto, C. and Otto, A. (2014b). Extending assembly line balancing problem by incorporating learning effects. International Journal of Production Research, 52(24):7193–7208.

Pei, J., Liu, X., Pardalos, P. M., Fan, W., and Yang, S. (2017). Scheduling deteriorating jobs on a single serial-batching machine with multiple job types and sequence-dependent setup times. Annals of Operations Research, 249:175–195.

Pine, B. J. (1993). Mass Customization: The New Frontier in Business Competition. Harvard Business School Press, Boston, Massachusetts.

Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. European Journal of Operational Research, 120(2):228–249.

Potts, C. N. and Van Wassenhove, L. N. (1992). Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. Journal of the Operational Research Society, 43:395–406.

Powell, W. B. (2010). Approximate Dynamic Programming. John Wiley & Sons.

Puchinger, J., Raidl, G. R., and Pferschy, U. (2010). The multidimensional knapsack problem: Structure and algorithms. INFORMS Journal on Computing, 22(2):250–265.

Rekiek, B. and Delchambre, A. (2006). Assembly Line Design: The Balancing of Mixed-Model Hybrid Assembly Lines with Genetic Algorithms. Springer-Verlag London.

Rotter, H. (2004). New operating concepts for intermodal transport: The mega hub in Hanover/Lehrte in germany. Transportation Planning and Technology, 27(5):347–365.

Sarker, B. R. and Pan, H. (2001). Designing a mixed-model, open-station assembly line using mixed-integer programming. Journal of Operational Research Society, 52(5):545–558.

Scholl, A. and Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research, 168(3):666–693.

Sha, M., Zhang, T., Lan, Y., Zhou, X., Qin, T., Yu, D., and Chen, K. (2017). Scheduling optimization of yard cranes with minimal energy consumption at container terminals. Computers & Industrial Engineering, 113:704–713.

Silva, Y. L. T. V., Subramanian, A., and Pessoa, A. A. (2018). Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. Computers & Operations Research, 90:142–160.

Simaria, A. S., Zanella de Sá, M., and Vilarinho, P. M. (2009). Meeting demand variation using flexible U-shaped assembly lines. International Journal of Production Research, 47(14):3937–3955.

Song, Y., Zhang, C., and Fang, Y. (2008). Multiple multidimensional knapsack problem and its applications in cognitive radio networks. In MILCOM 2008 - 2008 IEEE Military Communications Conference, pages 1–7. IEEE.

Souffriau, W., Vansteenwegen, P., Berghe, G. V., and Van Oudheusden, D. (2009). Variable neighborhood descent for planning crane operations in a train terminal. In Geiger, M. J., Habenicht, W., Sevaux, M., and Sörensen, K., editors, Metaheuristics in the Service Industry, pages 83–98. Springer.

Stahlbock, R. and Voß, S. (2008). Operations research at container terminals: a literature update. OR Spectrum, 30:1–52.

Stecco, G., Cordeau, J.-F., and Moretti, E. (2008). A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. Computers & Operations Research, 35(8):2635–2655.

Stephan, K. and Boysen, N. (2017). Crane scheduling in railway yards: an analysis of computational complexity. Journal of Scheduling, 20:507–526.

Strocko, E., Sprung, M., Nguyen, L., Rick, C., and Sedor, J. (2014). Freight facts and figures 2013. TechReport FHWA-HOP-14-004FHWA-HOP-14-004, U.S. Department of Transportation.

Tanaka, S. and Araki, M. (2013). An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. Computers & Operations Research, 40(1):344–352.

Tilk, C. and Irnich, S. (2017). Dynamic programming for the minimum tour duration problem. Transportation Science, 51(2):549–565.

U.S. Department of Transportation (2010). Freight transportation: Global highlights 2010. TechReport, U.S. Department of Transportation.

U.S. Department of Transportation (2013). Transportation for a new generation: Strategic plan for fiscal years 2012-2016. TechReport, U.S. Department of Transportation.

Varnamkhasti, M. J. (2012). Overview of the algorithms for solving the multidimensional knapsack problems. Advanced Studies in Biology, 4(1):37–47.

Verband der Automobilindustrie (2006). Auto Jahresbericht 2006. TechReport, Verband der Automobilindustrie.

Verband der Bahnindustrie in Deutschland (VDB) e.V. (2012). Wer hat die Nase vorn? der Umweltvergleich der Verkehrsträger. TechReport, Verband der Bahnindustrie in Deutschland (VDB) e.V.

Wang, H. M. (2011). Solving single batch-processing machine problems using an iterated heuristic. International Journal of Production Research, 49(14):4245–4261.

Webster, S. T. and Baker, K. R. (1995). Scheduling groups of jobs on a single machine. Operations Research, 43(4):692–703.

Wei, L., Zhu, W., and Lim, A. (2015). A goal-driven prototype column generation strategy for the multiple container loading cost minimization problem. European Journal of Operational Research, 241(1):39–49.

Winston, W. L. and Goldberg, J. B. (2004). Operation research: applications and algorithms. Thomson/Brooks/Cole.

Yuan, J. J., Lin, Y. X., Cheng, T. C. E., and Ng, C. T. (2007). Single machine serial-batching scheduling problem with a common batch size to minimize total weighted completion time. International Journal of Production Economics, 105(2):402–406.