ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Solving the single crane scheduling problem at rail transshipment yards

Xiyu Li [a], Alena Otto [a,*], Erwin Pesch [a,b]

[a] University of Siegen, Chair of Management Information Science, Kohlbettstraße 15, D-57068 Siegen, Germany
[b] Center for Advanced Studies in Management - CASiM, HHL Leipzig, Jahnallee 59, D-04109 Leipzig, Germany

ARTICLE INFO

Article history:
Received 17 October 2017
Received in revised form 20 May 2018
Accepted 19 July 2018
Available online xxxx

Keywords:
Crane scheduling
Transshipment yard
Branch-and-cut
Asymmetric traveling salesman problem with time windows
Minimum completion time problem

ABSTRACT

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.

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 Fig. 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 transshipment yards in Göttingen, Beiseförth, or Heilbronn. The single-crane scheduling problem also arises as a subproblem of crane scheduling problems with several cranes (e.g., [29,48,49,54]).

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 $\left[r_j^o, d_j\right]$ for each job $j \in J$ where the release time r_j^o is the earliest possible starting time of job j and the deadline d_j specifies the time when j

* Corresponding author.
E-mail address: alena.otto@uni-siegen.de (A. Otto).

https://doi.org/10.1016/j.dam.2018.07.021
0166-218X/

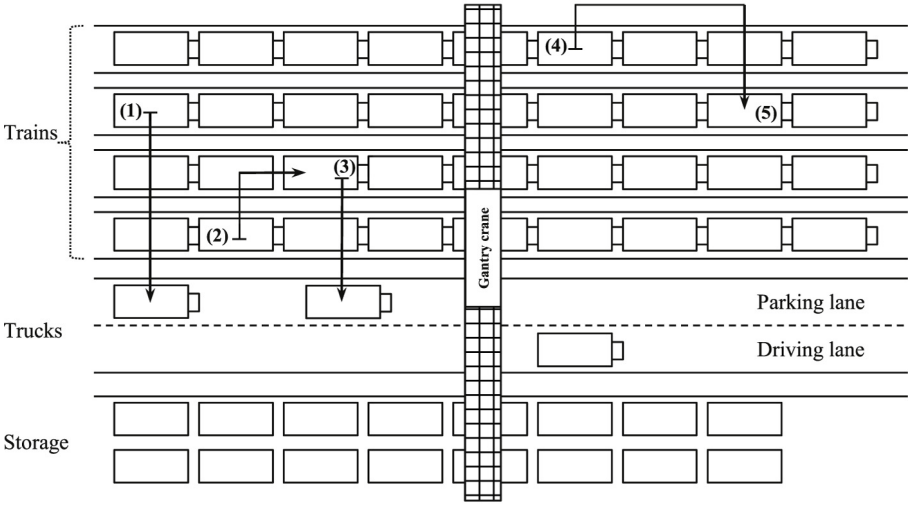

Fig. 1. Illustration of a typical rail terminal.

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 $p_a = p_z := 0$, $r_a^o = r_z^o := 0$, $d_a := 0$, $d_z := \infty$.

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 Fig. 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 [17,20,51]). 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 [20,21] as well as in [9].

Therefore the *Single-Crane Scheduling Problem (SgCSP)* is to find a schedule with a *minimal makespan* C_z, i.e. an optimal sequence $P^* = (j_1, j_2, \ldots, j_n)$ 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: $C_{j_k} \geq C_{j_{k-1}} + s_{j_{k-1}j_k}\ \forall k \in \{2, \ldots, n\}$.

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 [1] 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. [54] 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. [49] 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. [11,12] 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. [24] 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. [38] allow the pick-up positions and therefore the assignment of containers to cranes be a part of the optimization. Stephan and Boysen [57] 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. [19].

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

Several articles focus on single crane scheduling in the context of port container terminals. Thus, Ng and Mak [47] 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. [45] 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 [35] 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. [5] 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. [52] propose a mixed-integer model for crane scheduling to minimize the energy consumption. Fedtke and Boysen [28] 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 [31]. Dik and Kozan [26] 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. [18]. The classification scheme, complexity results and close to optimal approximation algorithms for twin cranes operating on the same level can be found in [37] and the fast branch and cut algorithm for dual cranes operating on different levels are proposed by Nossack et al. [48]. Barketau and Pesch [10] 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 [55] as well as Bierwirth and Meisel [14,15].

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. [8,23,25,39,59]).

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. [30]. Literature reviews on the machine scheduling literature with sequence-dependent setup times can be found in [3,4] and [2]. 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. [32,46,50,53,56]). 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: [13]; sum of weighted completion times: [22]; tardiness-related objectives: [16,34,40–43,58]). 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 [16,34,40–43,58]. 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. Sections 3 and 4 describe the DBC and the DA (and its combination), respectively. We report on extensive computational experiments in Section 5 and finally conclude with an outlook in Section 6.

2. Problem formulations for the single-crane scheduling problem

In Section 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), 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 to our preliminary evaluations. In Section 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 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.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

Table 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 times s_{ij}						
i	j					
	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	∞	∞	∞	∞	∞	–

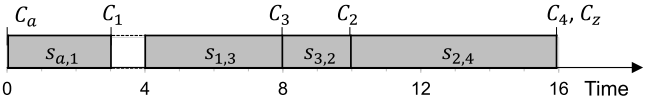

Fig. 2. Illustration of an optimal solution for the example in Table 1.

2.1. Some observations on the single-crane scheduling problem

Consider the example of Table 1 with six jobs, where job 3 has to precede job 2: $E = \{(3,2), (a,1), (a,2), (a,3), (a,4), (1,z), (2,z), (3,z), (4,z)\}$. Fig. 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.

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 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 $A = \{(i,j) \,|i \text{ may precede } j, i, j \in J\}$ describes all *possible* job pairs, in particular, $E \subseteq A$. Let $P = (j_1, j_2, \ldots, j_m)$ 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:

$$M(j_1) := r_{j_1}$$

$$M(j_1, \ldots, j_k) := \max\left\{r_{j_k}, M(j_1, \ldots, j_{k-1}) + s_{j_{k-1}j_k}\right\} \text{ for } k = 2, \ldots, m.$$

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 Fig. 2 is active.

Observation 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.

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 $P = (j_1, j_2, \ldots, j_m)$ 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 $\Theta(J, A, \text{GUB})$ 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.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

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 $C\left(\widetilde{P}^*\right)$.

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 $x\left(A'\right) = \sum_{(i,j)\in A'} x_{ij}$. 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. [7] and formulate a mixed-integer model for the SgCSP-Mincost as follows:

$$\min \quad \sum_{i\in J\setminus\{z\}} \sum_{j\in J\setminus\{a\}} s_{ij} \cdot x_{ij} \tag{1}$$

$$s.t. \quad x(\delta^+(i)) = 1 \qquad \forall\, i \in J \setminus \{z\} \tag{2}$$

$$x(\delta^-(i)) = 1 \qquad \forall\, i \in J \setminus \{a\} \tag{3}$$

$$x(A(K)) \leq |K| - 1 \qquad \forall\, K \subset J, 2 < |K| < n \tag{4}$$

$$x(P) \leq m - 2 \qquad \forall \text{ path } P = (j_1, j_2, \ldots, j_m), P \in \Theta \tag{5}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i,j \in J \tag{6}$$

Constraints (2) and (3) require that each job has exactly one successor and exactly one predecessor job. Subtours are excluded by constraints (4). The paths elimination constraints (5) eliminate all prohibited paths. The number of constraints of type (4) and (5) can be exponential in n.

Notice that any feasible solution of problem SgCSP-Mincost (J, A, GUB) is also a feasible solution for the original problem SgCSP (J, A), therefore $M(P^*(J,A)) \leq M\left(\widetilde{P}^*(J, A, \text{GUB})\right)$. 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 2. $C\left(\widetilde{P}^*(J, A, \text{GUB})\right) \leq M(P^*(J,A)) \leq M\left(\widetilde{P}^*(J, A, \text{GUB})\right)$.

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 $P := \widetilde{P}^*(J, A, \text{GUB})$, we update the global upper bound as $\text{GUB} := \min\left\{\text{GUB}, M\left(\widetilde{P}^*(J, A, \text{GUB})\right)\right\}$, 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 3.1, we provide an overview on the overall branch-and-cut procedure. Subsequently, in Section 3.2, we describe new separation routines designed for the SgCSP-Mincost problem.

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. [49] for a detailed discussion.

We compute a lower bound as an optimal solution to the assignment problem (1)–(3), and continuous variables in (6). An alternative lower bound is obtained from an optimal solution of the single machine scheduling problem $1\left|r'_j\right|C_{max}$ 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 $p'_j := \min_{i\in J|(i,j)\in A}\left\{s_{ij}\right\}$ and $r'_j := r_j - \min_{i\in J|(i,j)\in A}\left\{s_{ij}\right\}$ or as $p'_j := p_j + \min_{k\in J|(j,k)\in A}\left\{\vartheta_{jk}\right\}$ 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 [49] for details on preprocessing routines, GLB and GUB).

Starting with initial values for GLB and GUB, empty sets K and Θ in (4) and (5) as well as continuous variables in (6), we adapt cuts and separation routines in our branch-and-cut procedure as described by Ascheuer et al. [7]. We modify the

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

separation routines for the *path elimination constraints* (*PECs*, see Section 3.2). The cuts include besides subtour elimination constraints (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. [7]. We apply the cuts in the same order as in [7], 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 [49]). 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 $k' = \text{argmax}\{x_{k,i}(\varphi) \,|\, i \in J(\varphi), 0 < x_{k,i}(\varphi) < 1\}$ 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 3.2. 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 (A.1) (see Appendix) 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.

3.2. Extension of path elimination constraints

We extend the separation routine for path elimination constraints (PECs), originally used by Ascheuer et al. [7], by introducing precedence constraints tests (see Section 3.2.1) and waiting time tests (see Section 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 $P = (j_1, j_2, \ldots, j_m), m > 1$, 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 (1)–(4), (6). Note that because of constraints (2) and (3), there are only polynomially many such paths. In contrast to Ascheuer et al. [7], 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.

Observe that the logics of our tests in Sections 3.2.1–3.2.2 is somewhat related to the concept of the local-search domination of Bianco et al. [13] and Jamal et al. [33]. 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.

3.2.1. Precedence constraints tests

Precedence constraints test 1. Consider an infeasible path $P = (j_1, j_2, \ldots, j_{m-1}, j_m)$ 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 (A.4) (see Appendix) and all paths $([\{P\} \setminus \{j_m\}], j_m)$ where j_m is the last job, by PEC-constraints (A.3) (see Appendix).

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 $P = (j_1, j_2, \ldots, j_{m-1}, j_m)$. Then we exclude any path containing a consecutive permutation of jobs $\{P\}$, i.e., $[\{P\}]$ by constraint (A.2) (see Appendix).

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

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 $(J(\varphi), A(\varphi), \text{GUB})$ problem (see Section 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. 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 $P = (a = j_1, j_2, j_3 \ldots, j_m)$ 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 (A.2),
- paths of the form $[(\{P\} \cup \{j_k\}) \setminus \{j_m\}]$ by constraints (A.2) 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 $P' = (a, j_2, j_3 \ldots, j_{m-1}, j_k)$ 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 $P = (j_1, j_2 \ldots, z)$ 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 (A.2),
- 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 (A.2).

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 $w_{j_kj_2}(d_{j_k}) > 0$ implies that job j_2 is not available at time $(d_{j_k} + s_{j_kj_2})$ and therefore $C_{j_2} = r_{j_2}$. From $w_{j_kj_2}(d_{j_k}) \geq w_{j_1j_2}(d_{j_1})$ we conclude that $w(P') \geq w(P)$ where $P' = (j_k, j_2, j_3 \ldots, j_{m-1}, z)$ 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 $P = (j_1, j_2 \ldots, j_m)$ 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 (A.1),
- paths P' received from P by replacing job j_1 with j_k such that $w_{j_kj_2}(d_{j_k}) \geq w_{j_1j_2}(d_{j_1}) > 0$ by constraints (A.1),
- 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 (A.1), 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. $C_{j_{m-1}} := M(j_1, \ldots, j_{m-1})$.

Let us consider the second statement. Since $w_{j_kj_2}(d_{j_k}) > 0$ and $w_{j_1j_2}(d_{j_1}) > 0$, we receive that $r_{j_2} > d_{j_k} + s_{j_kj_2}$ and $r_{j_2} > d_{j_1} + s_{j_1j_2}$. 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''.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

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 $w_{j_kj_l} := w_{j_kj_l}(d_{j_k})$ 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

– $P' = (\ldots, j_k, j_c, j_l, \ldots)$ obtained from path P by inserting job j_c between j_k and j_l by constraint (A.1).

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. [36,44]).

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). If necessary we initialize $GUB := \infty$.

The initialization of the DA starts with the best feasible solution $P = (a, j_2, j_3, \ldots, j_{n-1}, z)$ 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).

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.

5. Computational experiments

Our computational experiments were run on a laptop with Intel i5-3450 3.1 GHz CPU. The algorithms are implemented in C++, utilizing the library of IBM ILOG CPLEX 12.7.

Next we describe our data sets in Section 5.1. We report on the results of our simulation of single-crane transshipment yards in Section 5.2. Section 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.

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 [6], 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. [27] 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. [49] for further details. We compute setup times and processing times based on typical horizontal and vertical velocities of a gantry crane,

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

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 Fig. 1). We generate time windows for rail–road jobs, which account for about 65% of all jobs (cf. [51]), 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 [6] originally describing instances of an asymmetric traveling salesman problem with time windows containing up to 233 jobs. In the original data set, Ascheuer [6] 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 [6] 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. [27], 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. [27] 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. [27] 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 s per instance. For the two-stage algorithms DBC&DA and DP&DA, we set a time limit to 600 s 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 $J^2 \neq \varnothing$ 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.

5.2. Simulation results

Table 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 s, 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 4). This is the case for 67 out of 90 instances. The computational time of the DBC&DA does not exceed 63 s per instance and the computational time of the DP&DA does not exceed 91 s 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 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. Fig. 3 illustrates that the makespan increases from about 33 min in case of no time windows to about 44 min (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 transshipment 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.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

Table 2
Performance of the DP&DA, the DBC&DA and the DP for DS1 with time limit of 600 s.

Setting, # of jobs (# of TW)[a]	Solution method	Avg. computational time (s)	# of instances decomposed with the DA
15 (all)	**DP&DA**	**0.0**	
	DBC&DA	**0.0**	10 out of 10
	DP	**0.0**	
20 (all)	**DP&DA**	**0.0**	
	DBC&DA	10.7	10 out of 10
	DP	1.1	
20 (10)	**DP&DA**	**0.0**	
	DBC&DA	**0.0**	10 out of 10
	DP	2.4	
20 (8)	**DP&DA**	**0.0**	
	DBC&DA	**0.0**	10 out of 10
	DP	4.4	
20 (6)	**DP&DA**	**0.5**	
	DBC&DA	0.9	9 out of 10
	DP	16.1	
20 (4)	DP&DA	7.1	
	DBC&DA	**0.4**	5 out of 10
	DP	12.9	
20 (2)	DP&DA	16.9	
	DBC&DA	**1.1**	3 out of 10
	DP	34.6	
20 (0)	DP&DA	2.1	
	DBC&DA	**0.1**	0 out of 10
	DP	44.0	
25 (all)	**DP&DA**	**0.0**	
	DBC&DA	**0.0**	10 out of 10
	DP	71.8	

Best algorithms in each setting are marked in bold.
[a] # of rail–road jobs with nontrivial time windows (TW).

Table 3
Differences in the makespan for the optimization with different objective functions.

Setting, # of jobs (# of TW)[a]	Avg. makespan for solutions with the minimal cost[b], (A)	Avg. minimal makespan, (B)	Avg. relative deviation $(A_i)/(B_i)$[c] (%)
15 (all)	2550.3	1944.3	132
20 (all)	3445.3	2619.9	132
25 (all)	4387.8	3191.0	138

[a] # of rail–road jobs with nontrivial time windows (TW).
[b] Average makespan of optimal solutions found by minimizing the total cost.
[c] $(A_i)/(B_i)$ is computed for each instance i, then the average is taken.

Table 4
Comparative performance of DBC&DA, DP&DA and DP on DS2 and DS3.

Data set	# of instances solved to optimality (total # of instances)			Avg. run time, sec.[a]			# 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

[a] For solved instances.

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. [27] 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 4 illustrates that the DBC&DA and the DP&DA outperform DP on DS2, which is the data set of Ascheuer [6]. 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.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

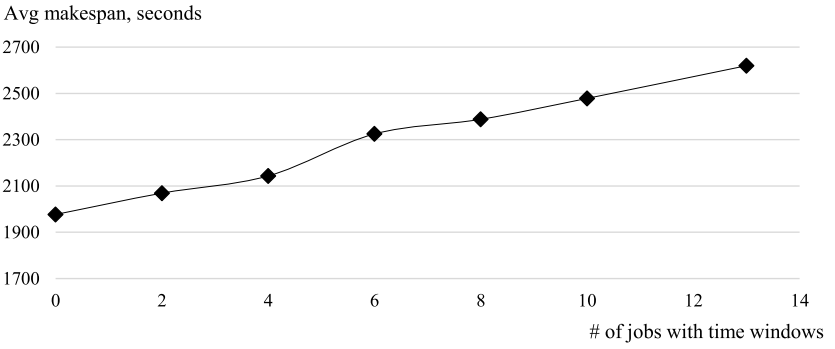

Fig. 3. Relation between the average optimal makespan and the number of rail–road jobs with time windows.

Table 5
Performance of the dominance rule of the DBC on DS3.

Avg # of nodes per instance	Avg # of nodes pruned with the dominance rule	Avg share of nodes pruned with the dominance rule, %
2708	525	15.8

Table 6
Performance of cuts in the DBC for DS2.

# of jobs	Path-elimination constraints (PECs)[a] based on				Precedence-related non-PEC cuts[b]		Other cuts	
	Precedence constraints tests, # cuts	Waiting time tests, # cuts	Other PECs, # cuts	CPU, %	# cuts	CPU, %	# 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.
[a] Precedence constraints and waiting time tests are described in Sections 3.2.1 and 3.2.2, respectively. Other tests check, whether paths violate time window constraints.
[b] π-, σ-inequalities as well as so-called precedence cycle breaking and special inequalities, see [7] for the details on the cuts and separation routines.

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 s each. On the other hand, the run time of the DP depends more strongly on instance sizes, which also points on the success of our decomposition algorithm. For example, the DP needs at least 9 s 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 2 and 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 s for the instances of DS1 with $n = 25$ jobs.

Performance of the specific algorithmic elements of the DBC. Table 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 5 and 6), we examine effectiveness of specific elements of our DBC procedure on the benchmark data set DS3. Table 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 3.1). Overall, the dominance rule prunes 15.8% of nodes per instance on average. Table 6 evaluates performance of the new tests in the separation routine for the path elimination constraints (see, cf. Section 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 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.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

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 traveling 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. [27], 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.

Acknowledgments

The authors have been supported by the German Science Foundation (DFG) through the grant "Optimierung der Containerabfertigung in Umschlagbahnhöfen, Germany" (PE 514/16-1,2), as well as the grant "Scheduling mechanisms for rail mounted gantries with regard to crane interdependencies, Germany" (PE 514/22-1).

Appendix

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

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

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

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

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

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

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

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

$$x\left(j_1 : Q\right) + x\left(A\left(Q\right)\right) := \sum_{k=2}^{m} x_{j_1 j_k} + \sum_{k=2}^{m} \sum_{l=2}^{m} x_{j_k j_l} \leq m - 2 \tag{A.4}$$

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

$$x\left(j_1 : Q\right) + x\left(A\left(Q\right)\right) + x(Q : j_m) := \sum_{k=2}^{m-1} x_{j_1 j_k} + \sum_{k=2}^{m-1} \sum_{l=2}^{m-1} x_{j_k j_l} + \sum_{k=2}^{m-1} x_{j_k j_m} \leq m - 2 \tag{A.5}$$

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

References

[1] K. Alicke, Modeling and optimization of the intermodal terminal mega hub, OR Spectrum 24 (1) (2002) 1–17.
[2] A. Allahverdi, The third comprehensive survey on scheduling problems with setup times/costs, European J. Oper. Res. 246 (2015) 345–378.
[3] A. Allahverdi, J.N.D. Gupta, T. Aldowaisan, A review of scheduling research involving setup considerations, Omega Int. J. Manag. Sci. 27 (1999) 219–239.
[4] A. Allahverdi, C.T. Ng, T.C.E. Cheng, M.Y. Kovalyov, A survey of scheduling problems with setup times or costs, European J. Oper. Res. 187 (2008) 985–1032.
[5] G. Alsoufi, X. Yang, A. Salhi, A combinatorial Benders' cuts approach to the seaside operations problem in container ports, in: Proceedings of the XI Metaheuristics International Conference, MIC 2015, 2015.
[6] N. Ascheuer, Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems. Technical Report TR 96–3, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.
[7] N. Ascheuer, M. Fischetti, M. Grötschel, Solving the asymmetric travelling salesman problem with time windows by branch-and-cut, Math. Program. 90 (3) (2001) 475–506.
[8] E.K. Baker, An exact algorithm for the time-constrained traveling salesman problem, Oper. Res. 31 (5) (1983) 938–945.
[9] M. Barketau, H. Kopfer, E. Pesch, A Lagrangian lower bound for the container transshipment problem at a railway hub for a fast branch-and-bound algorithm, J. Oper. Res. Soc. 64 (11) (2013) 1614–1621.
[10] M. Barketau, E. Pesch, An approximation algorithm for a special case of the asymmetric traveling salesman problem, Int. J. Prod. Res. 54 (14) (2016) 4205–4212.
[11] M. Barketau, E. Pesch, Y. Shafransky, Minimizing maximum weight of subsets of a maximum matching in a bipartite graph, Discrete Appl. Math. 196 (2015) 4–19.
[12] M. Barketau, E. Pesch, Y. Shafransky, Scheduling dedicated jobs with variative processing times, J. Comb. Optim. 31 (2) (2016) 774–785.
[13] L. Bianco, S. Ricciardelli, G. Rinaldi, A. Sassano, Scheduling tasks with sequence-dependent processing times, Navig. Res. Log. 35 (1988) 177–184.
[14] C. Bierwirth, F. Meisel, A survey of berth allocation and quay crane scheduling problems in container terminals, European J. Oper. Res. 202 (3) (2010) 615–627.
[15] C. Bierwirth, F. Meisel, A follow-up survey of berth allocation and quay crane scheduling problems in container terminals, European J. Oper. Res. 244 (3) (2015) 675–689.
[16] L.P. Bigras, M. Gamache, G. Savard, The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times, Discrete Optim. 5 (4) (2008) 685–699.
[17] N. Bostel, P. Dejax, Models and algorithms for container allocation problems on trains in a rapid transshipment shunting yard, Transp. Sci. 32 (4) (1998) 370–379.
[18] N. Boysen, D. Briskorn, F. Meisel, A generalized classification scheme for crane scheduling with interference, European J. Oper. Res. 258 (1) (2017) 343–357.
[19] N. Boysen, M. Fliedner, F. Jaehn, E. Pesch, A survey on container processing in railway yards, Transp. Sci. 47 (3) (2013) 312–329.
[20] N. Boysen, F. Jaehn, E. Pesch, Scheduling freight trains in rail-rail transshipment yards, Transp. Sci. 45 (2) (2011) 199–211.
[21] N. Boysen, F. Jaehn, E. Pesch, New bounds and algorithms for the Transshipment Yard Scheduling Problem, J. Sched. 15 (4) (2012) 499–511.
[22] F.D. Chou, H.M. Wang, T.Y. Chang, Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups, Int. Adv. Manuf. Technol. 43 (7–8) (2009) 810–821.
[23] N. Christofides, A. Mingozzi, P. Toth, State-space relaxation procedures for the computation of bounds to routing problems, Networks 11 (2) (1981) 145–164.
[24] M. Cichenski, F. Jaehn, G. Pawlak, E. Pesch, G. Singh, J. Blazewicz, An integrated model for the transshipment yard scheduling problem, J. Sched. 20 (1) (2017) 57–65.
[25] A.A. Cire, W.-J. van Hoeve, Multivalued decision diagrams for sequencing problems, Oper. Res. 61 (6) (2013) 1411–1428.
[26] G. Dik, E. Kozan, A flexible crane scheduling methodology for container terminals, Flexible Serv. Manuf. J. 29 (2017) 64–96.
[27] Y. Dumas, J. Desrosiers, E. Gelinas, M. Solomon, An optimal algorithm for the traveling salesman problem with time windows, Oper. Res. 43 (2) (1995) 367–371.
[28] S. Fedtke, N. Boysen, Gantry crane and shuttle car scheduling in modern rail–rail transshipment yards, OR Spectrum 39 (2) (2017) 473–503.
[29] G. Froyland, T. Koch, N. Megow, E. Duane, H. Wren, Optimizing the landside operation of a container terminal, OR Spectrum 30 (1) (2008) 53–75.
[30] R. Graham, E. Lawler, J. Lenstra, A. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Ann. Discrete Appl. Math. 5 (1979) 287–326.
[31] M. Held, R.M. Karp, A dynamic programming approach to sequencing problems, SIAM J. 10 (1962) 196–210.
[32] E.N. Ivanov, A.Y. Pogromsky, J.E. Rooda, Scheduling with sequence dependent setup times in application to magnetic resonance imaging scans processing. in: Proceedings of the IEEE International Conference on Control Applications/International Symposium on Intelligent Control, 2009, pp. 867–872.
[33] J. Jamal, G. Shobaki, V. Papapanagiotou, L.M. Gambardella, R. Montemanni, Solving the sequential ordering problem using branch and bound, in: Proceedings of IEEE SSCI 2017 — The 2017 IEEE Symposium Series on Computational Intelligence, 2017, pp. 3110–3118.
[34] J.G. Kim, D.H. Lee, Algorithms for common due-date assignment and sequencing on a single machine with sequence-dependent setup times, J. Oper. Res. Soc. 60 (9) (2009) 1264–1272.
[35] K.H. Kim, Y.M. Park, A crane scheduling method for port container terminals, European J. Oper. Res. 156 (3) (2004) 752–768.
[36] H. Klindworth, C. Otto, A. Scholl, On learning precedence graph concept for the automotive industry, European J. Oper. Res. 217 (1) (2012) 259–269.
[37] M.Y. Kovalyov, E. Pesch, A. Ryzhikov, A note on scheduling container storage operations of two non-passing stacking cranes, Networks 71 (3) (2018) 271–280.
[38] D. Kress, S. Meiswinkel, E. Pesch, The partitioning min–max weighted matching problem, European J. Oper. Res. 247 (3) (2015) 745–754.
[39] A. Langevin, M. Desrochers, J. Desrosiers, S. Gélinas, F. Soumis, A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows, Networks 23 (7) (1993) 631–640.
[40] X. Luo, F. Chu, A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness, Appl. Math. Computat. 183 (1) (2006) 575–588.
[41] X. Luo, C. Chu, A branch-and-bound algorithm of the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness, European J. Oper. Res. 180 (1) (2007) 68–81.
[42] X. Luo, C. Chu, C. Wang, Some dominance properties for single-machine tardiness problems with sequence-dependent setup, Int. J. Prod. Res. 44 (17) (2006) 3367–3378.
[43] X. Luo, D. Li, H. Song, Research on improvement of B&B algorithm performance of single machine total tardiness schedule, in: Proceedings of the international symposium on distributed computing and applications to business, engineering and science, no. (1-2), 2006, pp. 1368–1371.
[44] R. Montemanni, M. Mojana, G.A. Di Caro, L.M. Gambardella, A decomposition-based exact approach for the sequential ordering problem, J. Appl. Oper. Res. 5 (1) (2013) 2–13.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.

[45] R. Montemanni, D.H. Smith, A.E. Rizzoli, L.M. Gambardella, Sequential ordering problems for crane scheduling in port terminals, Int. J. Simul. Process. Model. 5 (12) (2009) 348–361.
[46] V. Nesello, A. Subramanian, M. Battarra, G. Laporte, Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times, European J. Oper. Res. 266 (2) (2018) 498–507.
[47] W.C. Ng, K.L. Mak, Yard crane scheduling in port container terminals, Appl. Math. Model. 29 (3) (2005) 263–276.
[48] J. Nossack, D. Briskorn, E. Pesch, Container dispatching and conflict-free yard crane routing in an automated container terminal, Transp. Sci. (2018). https://doi.org/10.1287/trsc.2017.0811.
[49] A. Otto, X. Li, E. Pesch, Two-way bounded dynamic programming approach for operations planning in transshipment yards, Transp. Sci. 51 (1) (2017) 325–342.
[50] J. Pei, X. Liu, P.M. Pardalos, W. Fan, S. Yang, Scheduling deteriorating jobs on a single serial-batching machine with multiple job types and sequence-dependent setup times, Ann. Oper. Res. 249 (1–2) (2017) 175–195.
[51] H. Rotter, New operating concepts for intermodal transport: The mega hub in Hanover Lehrte in Germany, Transp. Plan. Technol. 27 (5) (2004) 347–365.
[52] M. Sha, T. Zhang, Y. Lan, X. Zhou, T. Qin, D. Yu, K. Chen, Scheduling optimization of yard cranes with minimal energy consumption at container terminals, Comput. Ind. Eng. 113 (2017) 704–713.
[53] Y.L. Silva, A. Subramanian, Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times, Comput. Oper. Res. 90 (2018) 142–160.
[54] W. Souffriau, P. Vansteenwegen, G. van den Berghe, D. van Oudheusden, Variable neighborhood descent for planning crane operations in a train terminal, in: M.J. Geiger, W. Habernicht, M. Sevaux, K. Sörenson (Eds.), Metaheuristics in the Service Industry, in: Lecture Notes Econom. Math. Systems, vol. 624, Springer-Verlag, Berlin Heidelberg, 2009, pp. 83–98.
[55] R. Stahlbock, S. Voß, Operations research at container terminals: a literature update, OR Spectrum 30 (1) (2008) 1–52.
[56] G. Stecco, J.F. Cordeau, E. Moretti, A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times, Comput. Oper. Res. 35 (8) (2008) 2635–2655.
[57] K. Stephan, N. Boysen, Crane scheduling in railway yards: an analysis of computational complexity, J. Sched. 20 (5) (2017) 507–526.
[58] S. Tanaka, M. Araki, An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times, Comput. Oper. Res. 40 (2013) 344–352.
[59] C. Tilk, S. Irnich, Dynamic programming for the minimum tour duration problem, Transp. Sci. 51 (2) (2017) 549–565.

Please cite this article in press as: X. Li, et al., Solving the single crane scheduling problem at rail transshipment yards, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.021.