Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 843–851

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

Stacking and transporting steel slabs using high-capacity vehicles

Biljana Roljic[a,*], Sebastian Leitner[b], Karl F. Doerner[a]

[a] *Josef Ressel Center for Adaptive Optimization in Dynamic Environments, Department of Business Decisions and Analytics, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria*

[b] *Josef Ressel Center for Adaptive Optimization in Dynamic Environments, University of Applied Sciences Upper Austria, Softwarepark 13, 4232 Hagenberg, Austria*

Abstract

A crucial task of steel plant logistics is the transportation and intermediate storage of semi-finished casting products, so-called steel slabs. We consider a fleet that consists of high-capacity vehicles. The vehicles are used to transport steel slabs within the steel plant, i.e., between facilities and distinct areas of the slab yard, and they can carry out stacking operations autonomously. The slab yard stores a majority of all steel slabs. It is organized in rows that function as a two-dimensional stacking area. Each row consists of multiple stacks built up by steel slabs that are located on top of each other. Retrieving steel slabs from stacks amounts to solving an extended block relocation problem (BRP). We study a combined approach for solving both the routing of steels slabs within the plant and the stacking of steel slabs in storage as part of their retrieval during a vehicle's pickup operation.

Keywords: steel industry; vehicle routing; stacking; combined optimization

1. Introduction

A steel plant consists of multiple facilities and storage areas. A location of the steel plant's network can represent the continuous casting facility, a cutting and machining facility, the hot rolling mill, an indoor storage hall, and one of many outdoor storage areas that, as a whole, represent the slab yard. Steel slabs are produced at the continuous casting facility. Based on work plans, the cast steel is further processed at different facilities inside the steel plant and/or for a limited period stored in indoor storage halls or the slab yard. Eventually, all steel slabs end up at the hot rolling mill where they are transformed into coils.

The scope of our optimization problem in terms of routing is the transportation of steel slabs between the network's locations during a four hour-long operational period. For each steel slab that is to be transported during the considered planning horizon, a transportation request is issued, associated with a pickup location, delivery location, and time

* Corresponding author. Tel.: +43-1-4277-37963.
E-mail address: biljana.roljic@univie.ac.at

1877-0509
10.1016/j.procs.2021.01.334

window during which the transportation task is to be executed. The fleet consists of high-capacity vehicles, so-called straddle carriers, that can hold multiple steels slabs at once and autonomously perform loading and unloading operations. The steel slab routing problem corresponds to a multiple-vehicle one-to-one pickup-and-delivery problem, which is usually referred to as the pickup-and-delivery vehicle routing problem (PDVRP) [2], with time windows and additional loading constraints specific to steel plants. In the PDVRP each transportation request $r \in R$, where R is the set of all requests, consists of carrying an item from one pickup location to one destination location. Generally, neither of these locations is a depot. Time windows are modelled either by introducing a separate pickup time window (e^{r^+}, l^{r^+}) and delivery time window (e^{r^-}, l^{r^-}) or by specifying the earliest possible pickup time from the pickup location and the latest acceptable delivery time to a delivery location (e^{r^+}, l^{r^-}). A large number of exact [1, 17] and metaheuristic [4, 18] algorithms were developed for handling the more general PDVRP with time windows. For more details on the general pickup-and-delivery problem in the context of goods transportation we refer to [2, 10, 11].

The slab yard is a stock zone, which acts as an inventory buffer between the continuous casting facility and hot rolling mill. It is partitioned into areas in which each area consists of multiple rows of steel slab stacks. We put special emphasis on the pickup operations of the straddle carriers. When picking up steel slabs from the slab yard, the straddle carrier must assemble the transportation batch of requested steel slabs on its own. A steel slab may only be retrieved if it is located on top of a stack. Otherwise, the items that block it, i.e., that are located on top of it, must be relocated to another stack. This results in complex stacking operations that need to be performed. The straddle carrier can carry out all necessary stacking operations due to an integrated crane in-between its wheels. However, movement restrictions specific to slab yards apply, limiting the straddle carrier in space when performing certain moves. With regards to the storage aspect of our combined optimization problem, we describe our steel slab stacking problem as an unrestricted block relocation problem (BRP) with distinct item priorities, multi-lift cranes, and movement restrictions specific to steel slab yards. In what follows, we clarify this problem description. In the basic variant of the BRP, also commonly referred to as the container relocation problem [7], items are organized in stacks inside of a bay and they have to be retrieved in a certain order. Each item, representing a container, box, steel slab or the like, is assigned a distinct priority value and usually item with priority value 1 has to be retrieved first, followed by item with priority value 2, etc. An item may only be retrieved if it is on top of a stack. Any other item that is located above the next item to be retrieved must first be relocated to another stack. The problem is solved once the bay is emptied and the objective is to minimize the total number of moves, or more precisely, relocations. The device used for moving items has a capacity limited to one item. In [8], the authors introduced the BRP and they studied a variant that is now known as the restricted BRP. This variant, as described in [9], only allows forced moves, i.e., moves that involve relocating items blocking the next item to be retrieved. The unrestricted BRP additionally considers cleaning moves, commonly referred to as voluntary moves, i.e., by allowing any other type of relocation. As this setting explores a larger solution space, hence yielding more opportunities for optimization, it has become subject to an increasing research interest particularly in recent years [6, 20, 21]. BRPs that stem from real world problem settings in steel plants are already established in the literature, particularly the variant with cranes that move one item at a time. In [16], the authors study a steel slab stacking problem that arises at the continuous casting facility inside a steel plant. This problem corresponds to an unrestricted BRP with gantry cranes with a capacity limited to one item. A further contribution is provided in [15], where the previous work of [16] is extended by studying a dynamic variant of the steel slab stacking problem. In [3], the authors study a more holistic approach of the steel slab stacking problem by investigating the aspect of uncertainty at a continuous casting facility. Steel slab stacking problems that take multi-lift cranes into account have already been studies in the past, though to a modest degree. In [14], the authors study an inventory matching problem inside a steel plant. They too study the stacking operations of straddle carriers, however, they explore the steel slab stacking aspect in a rather abstract manner. In [22], the authors study what they title the BRP with batch moves (BRP-BM). This problem occurs in steel slab yards operated by multi-lift bridge cranes and relates to the unrestricted BRP with distinct item priorities. They propose a lower bound on the number of moves for the BRP-BM and a metaheuristic based on a tree search.

In our combined steel slab routing and stacking problem, practitioners impose high processing rates such that not all steel slabs can be delivered during the considered planing horizon. Therefore, the objective function is organized in a lexicographic fashion: first, maximize the throughput; second, minimize the sum of the total time travelled inside the steel plant (whilst routing) and total stacking time inside the slab yard (whilst stacking). We solve the problem by means of a large neighbourhood search (LNS) metaheuristic, addressing the routing of the steel slabs, and a tailored

stacking heuristic, optimizing the stacking operations. A straddle carrier's route is composed of a sequence of pickup and delivery visits to different locations inside the steel plant during the planning horizon. Part of each pickup visit to the slab yard, is the retrieval of the requested steel slabs. This requires a steel slab stacking problem to be solved for each pickup operation. Since the combined steel slab routing and stacking is solved in an iterative manner, it is of utmost importance that the steel slab stacking problem is solved quickly to allocate time to the routing model to globally optimize all routes. Hence and as a first step, our goal is to develop a fast method to solve the steel slab stacking problem able to provide a feasible upper bound on the duration of the pickup operations, i.e., the stacking. In this paper, we propose a quick steel slab stacking heuristic, describe its integration into the routing metaheuristic, and discuss the interplay between the two methods. We contribute to integrated planning in the context of Industry 4.0, by collectively addressing two otherwise separately treated logistic tasks of a steel plant, namely the routing and stacking of steel slabs.

2. Steel slab routing

From this point forward, we refer to steel slabs as items and to straddle carriers as vehicles. The steel slab routing problem models a fleet of vehicles K. Each vehicle $k \in K$ visits various locations to perform pickup and delivery operations. Items are stacked on top of each other inside of a vehicle. The time required for loading or unloading an item to or from a vehicle is denoted by σ^k. Every vehicle has a capacity in terms of weight Q^k as well as the total height of items stacked on top of each other that can be simultaneously transported U^k. We consider a complete directed graph $G = (V, A)$ with node set V and arc set A. Each arc $(i, j) \in A$ is associated with a travel time t_{ij}. The nodes represent locations inside the steel plant, i.e., production facilities and storage locations. For each item that is to be delivered during the considered planning horizon, a transportation request is issued. Each request $r \in R$, where R is the set of all requests, specifies a pickup location p^r, a delivery location d^r, a release time e^r formulated as a lower bound on the pickup time, and a due time l^r formulated as an upper bound on the delivery time. Generally, the steel slab routing model is free to decide on the composure of transportation batches, as items can be liberally mixed and matched when transporting them collectively.

We propose a LNS metaheuristic for solving the steel slab routing problem. LNS, originally proposed in [19], is a metaheuristic approach well suited for a broad range of routing problems [13]. It is based on the idea of running the algorithm in successive segments and repetitively destroying and repairing parts of the solution. Newly generated solutions are accepted as new incumbent solution following an simulated annealing acceptance criterion. We adopt most removal and reinsertion operators as described in [18], such as *random*, *worst* and *Shaw removal*; *basic greedy* and *regret-k reinsertion*. The additional removal operators built into our LNS are as follows. An *entire route removal* operator [5], removes one or more complete routes from a solution. The route to be removed is selected among non-empty vehicles in a random fashion. We keep track of each item's movement by means of their legs, i.e., all arcs that an item travels along from its pickup location to its delivery location. An *item leg-oriented removal* operator selects requests for removal by applying a roulette wheel with the number of item legs as weights. After each destroy iteration, a repair iteration follows. Here, all removed requests are reinserted into the solution. Additionally, all requests that could not be included in the incumbent solution are considered as candidates for insertion.

3. Steel slab stacking

The steel slab stacking problem models the movements of a vehicle during the retrieval procedure of requested items as part of a vehicle's pickup operation inside of a row of the slab yard. The transportation model determines for each vehicle of fleet K the requests that they serve by providing a sequence of request pickups and deliveries. In our combined steel slab routing and stacking problem, we treat each row of the slab yard as a separate bay. Requests that share the same row as a pickup point are likely to be picked up collectively given that the vehicle's capacity restrictions are not violated. Accordingly, a steel slab stacking problem needs to be solved for all requests that are collectively retrieved from a row or, more precisely, a bay. In what follows, we formally describe our steel slab stacking problem. First, we elaborate on the physical properties of the bays, items, and vehicles. Second, we discuss the priority values of each item of a bay.

Steel slabs are heterogeneous items that differ from each other with regards to their weight q^r, width w^r, length l^r, and height h^r. These physical properties are relevant for vehicle loading and stacking constraints. In addition to the fleet's capacity constraints, Q^k and U^k, as mentioned in the previous section 2, dimensional loading constraints $\mathcal{D}_{constr}$ need to be considered. These refer to length and width restrictions among items that are collectively lifted as a batch by the vehicle. They specify that wider and longer items need to be at the lower positions of a batch unless dimensional thresholds are obeyed. The straddle carrier is a vehicle with an integrated crane in-between its wheels. It accesses the row of a slab yard by enclosing it between its wheels, hence hovering above the item stacks. Within the row, the vehicle can only navigate one-dimensionally by moving backward or forward. The stacking area, i.e., the bay, contains a set of items $B = \{1, ..., n\}$ and is defined by its length L (number of stacks in a row) and height H (number of tiers). The travel time from one stack to a neighboring stack equals time unit τ, then again, the travel time from one stack to the second neighboring stack equals $2 \cdot \tau$, etc. The possible positions of items within the stacking area are called slots, that are defined by a stack number i and a tier number j. Particular properties of such a stock zone are:

(i) a single item is accessible only from the top, i.e., it can be retrieved only if it is the highest item in the stack,

(ii) each item in the stacking area must be placed either on top of another item or on the ground (tier one),

(iii) the number of stacks is bounded by L, i.e., opening a new $(L + 1)^{\text{th}}$ stack is not feasible, however, it is feasible to use the $(L + 1)^{\text{th}}$ stack as an assembly stack for the transportation batch,

(iv) the number of tiers of each stack is bounded by H,

(v) the total height and weight of all items that can be simultaneously lifted are bounded by U^k and Q^k respectively,

(vi) an empty vehicle can navigate over a stack height of $H + 1$, however, when it is loaded with n slabs, it can then navigate over a maximal stack height of $|H| + 1 - n$. Each movement from stack m to stack n as in $(m, n) \in L$ is associated with a vehicle travel time t^k_{mn}. We refer to those limitations as movement constraints $\mathcal{M}_{constr}$.

Each requested item $r \in T$, where T is the set of items to be retrieved from bay B during the considered pickup operation, is placed onto the assembly stack $(L + 1)$ of the transportation batch. A priority value i^r is assigned to each item in bay B. The assembly stack contains items that are sorted according to their priority values, following an increasing sequence starting at the lowest tier $j = 1$ upward. The priority value i^r of each requested item $r \in T$ is determined in a lexicographic fashion: first, by means of its physical properties (length and width restrictions); second, following a sorting parameter. The latter aspect refers to business policies. All requested items in T have to be sorted in a specific manner on the assembly stack $L + 1$ of the transportation batch. These sorting criteria result in distinct priority values. Finally, the priority values of the remaining items of B excluding items of T, i.e., $B \setminus T$, are provided as input. We would like to stress that even though the items that remain in the bay $B \setminus T$ are not picked up and retrieved when solving an instance of our combined steel slab routing and stacking problem corresponding to a four hour-long planning horizon, they will be considered at a later time outside of this scope of time. Hence, we still take their priority values into account during retrieval operations.

In our extended BRP considered in this paper, more than one item at a time can be lifted by the vehicle. Applying well-known solution methods from literature, which address the standard BRP where only one item at a time can be lifted, requires a methodological adjustments. We solve our steel slab stacking problem by means of a look-ahead algorithm (LA-N), as proposed in [12] for solving the standard BRP. In order to fit the multi-lift crane capacity of our vehicles, we extend the LA-N algorithm with tailored batching rules. Furthermore, we incorporate travel time considerations as part of tie breaking decisions when selecting one of multiple stacks feasible for a relocation. The purpose of the steel slab stacking method discussed in this paper is to serve the steel slab routing problem by providing a feasible upper bound on the duration of the pickup operation. Hence, whenever inserting a request into a route during the reinsertion procedure of the LNS, a steel slab stacking problem is solved. This procedure requires, first and foremost, a fast steel slab stacking method. Hence, we choose to adopt and extend the LA-N algorithm as it is known for being computationally fast and as the extent to which cleaning moves are considered is adjustable through parameter N.

The LA-N algorithm refers to a family of algorithms, each corresponding to a different value of the parameter N. The value of N refers to the degree of look-ahead. Forced moves, in particular, only involve item relocations blocking

the next item to be retrieved. In other words, only moves of items that are in the same stack as the item to be retrieved next are allowed and no look-ahead mechanism applies. This is the case when $N = 1$. For higher values of N, cleaning moves come into play by looking at the stacks of the next N items to retrieve. Cleaning moves are performed by relocating an item in a way that does not require the item to be relocated a second time. For a more formal description of the LA-N heuristic we refer to [12].

We propose a heuristic scheme that considers the following types of stacking operations:

1. Initialize bay:
   1.1. Select a value for N (the look-ahead). Otherwise, default value $N = |T|$.
   1.2. Batch the bay: For each stack $s \in S$ of bay B, starting at the lowest tier $j = 1$ and moving upward, check if items are located on top of each other such that item with priority value i^r is located directly below item with priority value $i^r + 1$. If the previous statement holds true and if the batching does not violate any capacity restrictions U^k or Q^k, batch those items as they will be further collectively relocated or retrieved. If the batch cannot be further extended as it would violate capacity restrictions U^k or Q^k, then mark the batch as permanent and safe it to $\Omega^{permanent}$. Otherwise, safe it to $\Omega^{incomplete}$.
2. Batching moves: Let $b^*(s)$ denote the individual item located on top of stack s. Let $B^*(s)$ denote the incomplete batch (member of $\Omega^{incomplete}$) located on top of stack s. If the top of stack s contains a permanent batch (member of $\Omega^{permanent}$), then $b^*(s)$ and $B^*(s)$ remain empty. For each stack $s \in S$ examine if it contains an item $b^*(s)$ or batch $B^*(s)$. If the previous statement holds true, examine if item $b^*(s)$ (or batch $B^*(s)$) can be moved to the top of another stack $s' \in S \setminus s$ that contains a single item $b'(s')$ (or another incomplete batch $B'(s')$) on top such that the priority value i^r of item $b^*(s)$ (or the item at the bottom of batch $B^*(s)$) is located directly above item $b'(s')$ (or the item on top of batch $B'(s)$) with priority value $i^r - 1$. If possible, perform the batching move and combine item $b^*(s)$ (or batch $B^*(s)$) with item $b'(s)$ or (batch $B'(s)$) to a new batch. If the new batch cannot be further extended as it would violate capacity restrictions U^k or Q^k, movement constraints $\mathcal{M}_{constr}$, or dimensional constraints $\mathcal{D}_{constr}$, then mark the batch as permanent and safe it to $\Omega^{permanent}$. Otherwise, save it to $\Omega^{incomplete}$.
3. Retrieval moves:
   3.1. If possible, i.e., if no movement constraints $\mathcal{M}_{constr}$ or dimensional constraints $\mathcal{D}_{constr}$ are violated, retrieve item batches (member of $\Omega^{permanent}$ or $\Omega^{incomplete}$) or individual items that can be retrieved without a relocation.
   3.2. If all requested items of T have been retrieved, terminate.
   3.3. If batching moves are possible, return to step 2. Otherwise, proceed to step 4.
4. Cleaning moves:
   4.1. Let $b^*(s)$ denote the individual item located on top of stack s. Let $B^*(s)$ denote the batch (member of $\Omega^{incomplete}$ or $\Omega^{permanent}$) located on top of stack s. If possible, perform cleaning moves by looking at the stacks of the next N items to retrieve. For each of those stacks $s \in S^N$, evaluate the cleaning move for item $b^*(s)$ or batch $B^*(s)$. A cleaning move does not apply because either (i) it cannot be relocated so as never to be relocated again or (ii) it is already the lowest numbered item in its stack. Ensure that no capacity restrictions U^k or Q^k, movement constraints $\mathcal{M}_{constr}$ or dimensional constraints $\mathcal{D}_{constr}$ are violated. Ties are broken by selecting the closer stack (such that the travel time is minimized).
   4.2. If batching moves are possible, return to step 2.
   4.3. If retrievals are possible, return to step 3. Otherwise, proceed to step 5.
5. Forced moves:
   5.1. Let s^* denote the stack that contains the item or batch (member of $\Omega^{permanent}$ or $\Omega^{incomplete}$) next to retrieve. If possible, temporarily batch items for a forced move. From the top tier of stack s^* moving downward

until the item or batch next to retrieve is reached, check if items are located on top of each other such that item with priority value i^r is located directly above item with priority value greater than $i^r + 2$. If this holds true and if no capacity restrictions U^k or Q^k are violated, temporarily combine those items or batches (member of $\Omega^{incomplete}$) to batch $F^*(s^*)$ as the forced move will be collectively performed. Otherwise, let $b^*(s^*)$ be equal to the individual item on the top of stack s^* or let $B^*(s^*)$ be equal to the batch (member of $\Omega^{incomplete}$ or $\Omega^{permanent}$) on the top of stack s^*.

5.2. Let $Height(s)$ denote the height of stack s. Let $Low(s)$ denote the lowest numbered item in stack s. An individual item or a batch will be relocated during the forced move. Let $n = b^*(s^*)$ or n be equal to the item with the largest priority value in $F^*(s^*)$ or n be equal to the item with the largest priority value in $B^*(s^*)$ on top of stack s^*. Let $D = \{s|Low(s) > n$ and $Height(s) < H^{max}\}$. If D is empty, item or batch n will have to be relocated yet again before finally being retrieved. Place item or batch n on the stack s with the highest value of $Low(s)$ such that $Height(s) < H$ and no movement constraints $\mathcal{M}_{constr}$ or dimensional constraints $\mathcal{D}_{constr}$ are violated. Otherwise, if D is non-empty, there is a way to relocate item n such that it is not relocated again until its retrieval. In this case, relocate item n to the stack in D with the lowest $Low(s)$ while ensuring that movement constraints $\mathcal{M}_{constr}$ or dimensional constraints $\mathcal{D}_{constr}$ are not violated. Ties are broken by selecting the closer stack (such that the travel time is minimized).

5.3. Go to step 2.

Figure 1 illustrates the decisions made by the LA-N algorithm extended by our batching rules using three different example instances a), b), and c). The bays of each example are initialized with 9 items. The number inside each item corresponds to its priority value. In all examples, 5 items (following their priority value from smallest to largest) must be retrieved, $H = 5$, $L = 3$, and, for the sake of simplicity, the vehicle capacities U^k and Q^k are limited to 2 items. The *batching of a bay*, as described in step 1.2. of our heuristic scheme, is illustrated in the initial bays of each example specified by "start". The dotted borders indicate batches. For instance, the initial bay of example c) shows, that even though stack (1) contains 4 items sorted in an increasing order from bottom to top, they cannot be permanently batched as this would violate the vehicle's capacity restrictions limited to 2 items. Therefore items 6 and 7 and items 8 and 9 are separately batched and saved to $\Omega^{permanent}$. Keep in mind, that having items 8 and 9 located on top of 6 and 7 is not desirable as items 6 and 7 need to be retrieved first followed by items 8 and 9. Hence, items 8 and 9 must at some point be relocated first. However, assuming that the vehicle capacity would allow 4 items, then items 6, 7, 8 and 9 would be ideally sorted, combined to a common batch, saved to $\Omega^{permanent}$, and retrieved without any relocations. *Batching moves*, as described in step 2., are demonstrated in the second move of example a) and b) as well as in the first move of example c). In all cases, the resulting batch corresponds to a permanent batch and is saved to $\Omega^{permanent}$. The *retrieval moves*, as described in step 3.1., are self-explanatory. The examples demonstrate retrievals of single items, e.g., example a) move 3, and batches, e.g., example a) move 4. *Cleaning moves*, as described in step 4., are illustrated twice. In particular in example a) during the first move, items 4 and 5 are moved to stack (1) as the lowest item of this stack $Low(1) = 6$ is still larger then items 4 and 5. Stack (2) is not an option as it contains item 1 ($Low(2) = 1$) and 3, all of which have to be retrieved before items 4 and 5. *Forced moves*, as described in step 5., are illustrated in example b) during move 1 and in example c) during move 4. Both forced moves result in the desirable state where the relocated batches do not have to be relocated again until their retrieval. Move 1 of example b) describes how items are temporarily batch for a forced move as described in step 5.1.. Move 4 of example c) shows a forced relocation of a permanent batch $B^* \in \Omega^{incomplete}$. Moreover, it shows that items 2 and 3 can be relocated from stack (3) to either stack (1) or stack (2). The tie is broken by selecting the closest stack, such that the travel time is reduced.

4. Experimental results

As this work is motivated by a problem faced by our industrial partner, we consult their historic data to study our combined steel slab routing and stacking approach. The vehicle capacities on average allows for up to 4 items to be lifted and carried at once. The average values for the number of stacks and height of each stack of a bay are $L = 10$ and $H = 9$, respectively. During a four hour-long operational period up to 100 requests are placed. We test our combined steel slab routing and stacking model on 8 real-world instances. The instances vary according to how well the slab

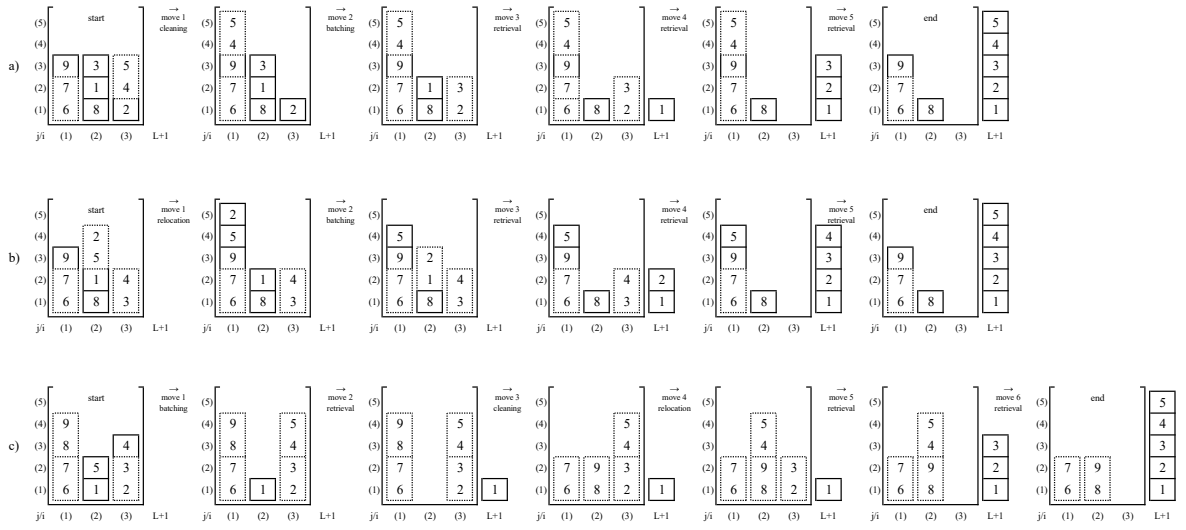

Fig. 1. Steel slab stacking examples

yard is organized. We investigate the impact of a well and poorly-organized slab yard and the interplay of the steel slab routing and stacking problem, and make following observations regarding these instances.

4.1. On the impact of a well-organized slab yard

A well-organized slab yard eases the steel slab stacking problem. In such a case, the rows of the slab yard containing items for pickup, i.e., retrieval, are conveniently sorted with regards to their priority value. In such a case, the total stacking duration consumes less time. This in turn allows the fleet to fulfil more requests. Consequently, this balances the total stacking time (resulting from the stacking problem) and the total travel time (resulting from the routing problem). Experimental results show that in such a scenario, the stacking time amounts to 40%-50%, while the travel time takes up more than halve the time. Naturally, a poorly organized slab yard, requires more relocations resulting in a longer total stacking duration overall. This motivates the combination of the routing and stacking problems and exploration several neighborhoods. While the stacking method is concerned with the retrieval of items, it is however the routing problem that decides which items are collectively retrieved during which pickup operation. The importance of the routing model's decision is amplified in instances where a row, i.e., bay, is visited multiple times by the vehicles as it contains lots of requested items. Results show that in such a scenario the stacking time requires up to 70% of the total time, while the travel time fades into the background. This finding reinforces the requirement to use a fast stacking method to provide the routing model with estimates on the stacking duration of each pickup operation, such that it can make suitable decisions on the assignment of requests to pickup operations.

4.2. On the performance of the steel slab routing and stacking model separately

When examining the routing model independently from the stacking model, experimental studies indicate that the total travel time of the fleet can be reduced by 6% on average compared to the solutions provided by our industrial partner. When examining only the stacking results and comparing them to solutions provided by our industrial partner, experimental results show large deviations. In some rather simple instances, our extended LA-N algorithm can naturally find equally good or even slightly improved solutions. For more complex instances the stacking method fails to improve the industrial partner's results. However, this outcome was anticipated as the purpose of our extended LA-N algorithm is to provide an upper bound to the stacking time with a low computational effort. This encourages the need to develop an improved steel slab stacking method, able to provide better results with regards to solution quality.

5. Outlook

In this paper, we have described a steel slab stacking problem specific to a steel slab yard and straddle carrier. We have proposed a fast steel slab stacking algorithm able to quickly obtain a feasible solution that provides an upper bound on the duration of a vehicle's pickup operation. In future work, we intend to improve the combined steel slab routing and stacking models with an extended stacking model able to solve the stacking problems with regards to a better solution quality. This model will steer the LNS more accurately and provide improved stacking results. Moreover, we plan to introduce item transshipments to the steel slab routing model.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

References

[1] Baldacci, R., Bartolini, E., Mingozzi, A., 2011. An exact algorithm for the pickup and delivery problem with time windows. Operations Research 59, 414–426. doi:10.1287/opre.1100.0881.

[2] Battarra, M., Cordeau, J.F., Iori, M., 2014. Pickup-and-delivery problems for goods transportation, in: Toth, P., Vigo, D. (Eds.), Vehicle Routing: Problems, Methods, and Applications, Second Edition. chapter 6, pp. 161–191. doi:10.1137/1.9781611973594.ch6.

[3] Beham, A., Raggl, S., Wagner, S., Affenzeller, M., 2019. Uncertainty in real-world steel stacking problems, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, New York, NY, USA. p. 1438–1440. doi:10.1145/3319619.3326803.

[4] Bent, R., Hentenryck, P.V., 2006. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research 33, 875 – 893. doi:https://doi.org/10.1016/j.cor.2004.08.001. part Special Issue: Optimization Days 2003.

[5] Danloup, N., Allaoui, H., Goncalves, G., 2018. A comparison of two meta-heuristics for the pickup and delivery problem with transshipment. Computers & Operations Research 100, 155–171. doi:10.1016/j.cor.2018.07.013.

[6] Feillet, D., Parragh, S., Tricoire, F., 2019. A local-search based heuristic for the unrestricted block relocation problem. Computers & Operations Research 108, 44–56. doi:10.1016/j.cor.2019.04.006.

[7] Forster, F., Bortfeldt, A., 2012. A tree search procedure for the container relocation problem. Computers & Operations Research 39, 299–309. doi:10.1016/j.cor.2011.04.004.

[8] Kim, K., Hong, G.P., 2006. A heuristic rule for relocating blocks. Computers & Operations Research 33, 940–954. doi:10.1016/j.cor.2004.08.005.

[9] Lehnfeld, J., Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. European Journal of Operational Research 239, 297–312. doi:10.1016/j.ejor.2014.03.011.

[10] Parragh, S., Doerner, K., Hartl, R., 2008a. A survey on pickup and delivery problems: Part i: Transportation between customers and depot. Journal für Betriebswirtschaft 58, 21–51. doi:10.1007/s11301-008-0033-7.

[11] Parragh, S., Doerner, K., Hartl, R., 2008b. A survey on pickup and delivery problems: Part ii: Transportation between pickup and delivery locations. Journal für Betriebswirtschaft 58, 81–117. doi:10.1007/s11301-008-0036-4.

[12] Petering, M., Hussein, M., 2013. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. European Journal of Operational Research 231, 120–130. doi:10.1016/j.ejor.2013.05.037.

[13] Pisinger, D., Ropke, S., 2010. Large neighborhood search, in: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer US, Boston, MA, pp. 399–419. doi:10.1007/978-1-4419-1665-5_13.

[14] Raa, B., van der Stricht, W., 2006. A local search algorithm for the inventory matching problem in a steel company. IFAC Proceedings Volumes 39, 359–364. doi:10.3182/20060517-3-FR-2903.00193.

[15] Raggl, S., Beham, A., Affenzeller, M., 2020. Investigating the dynamic block relocation problem, in: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (Eds.), Computer Aided Systems Theory – EUROCAST 2019, Springer International Publishing, Cham. pp. 438–445.

[16] Raggl, S., Beham, A., Tricoire, F., Affenzeller, M., 2018. Solving a real world steel stacking problem. International Journal of Service and Computing Oriented Manufacturing 3, 94–108. doi:10.1504/IJSCOM.2018.091621.

[17] Ropke, S., Cordeau, J.F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43, 267–286. doi:10.1287/trsc.1090.0272.

[18] Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40, 455–472. doi:10.1287/trsc.1050.0135.

[19] Shaw, P., 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK .

[20] Tanaka, S., Mizuno, F., 2018. An exact algorithm for the unrestricted block relocation problem. Computers & Operations Research 95, 12–31. doi:10.1016/j.cor.2018.02.019.

[21] Tricoire, F., Fechter, J., Beham, A., 2018. New insights on the block relocation problem. Computers & Operations Research 89, 127–139. doi:10.1016/j.cor.2017.08.010.

[22] Zhang, R., Liu, S., Kopfer, H., 2015. Tree search procedures for the blocks relocation problem with batch moves. Flexible Services and Manufacturing Journal 28, 397–424. doi:10.1007/s10696-015-9229-z.