A Hybrid Algorithm for Computing Tours in a Spare Parts Warehouse

Matthias Prandtstetter, Günther R. Raidl, and Thomas Misar

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{prandtstetter|raidl}@ads.tuwien.ac.at

Abstract. We consider a real-world problem arising in a warehouse for spare parts. Items ordered by customers shall be collected and for this purpose our task is to determine efficient pickup tours within the warehouse. The algorithm we propose embeds a dynamic programming algorithm for computing individual optimal walks through the warehouse in a general *variable neighborhood search* (VNS) scheme. To enhance the performance of our approach we introduce a new self-adaptive *variable neighborhood descent* used as local improvement procedure within VNS. Experimental results indicate that our method provides valuable pickup plans, whereas the computation times are kept low and several constraints typically stated by spare parts suppliers are fulfilled.

1 Introduction

Nowadays, spare parts suppliers are confronted with several problems. On the one hand, they should be able to supply spare parts both on demand and as fast as possible. On the other hand, they have to keep their storage as small as possible for various economic reasons. Storage space itself is expensive, but more importantly by adding additional capacity to the stock, the complexity of administration increases substantially. Therefore, the demand for (semi-)automatic warehouse management systems arises. Beside keeping computerized inventory lists additional planning tasks can be transferred to the computer system. For example, lists containing all articles to be reordered can be automatically generated.

Obviously the main task to be performed within a spare parts warehouse is the issuing of items ordered by customers and to be sent to them as quickly as possible. For this purpose, several warehousemen traverse the storage and collect ordered articles which will then be brought to a packing station where all items are boxed and shipped for each customer. Of course, the possible savings related with minimizing collecting times of items are high and therefore effort should be put into a proper tour planning. Various constraints related to capacities of trolleys used for transporting collected articles, structural conditions of the warehouse and delivery times guaranteed to the customers have to be considered.

The rest of this paper is organized as follows: The next section gives a detailed problem definition. In Sec. 3 we present an overview of related work. A

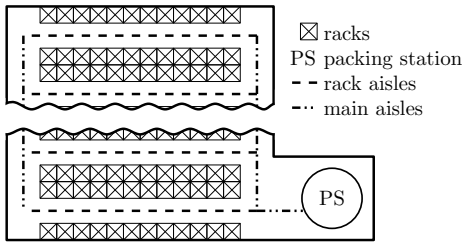

Fig. 1: An exemplary storage layout. Main aisles (vertical in this sketch) and rack aisles (horizontally aligned) are joining each other orthogonally.

new hybrid approach based on *variable neighborhood search* and *dynamic programming* is presented in Sec. 4. Experimental results and conclusions complete the paper.

2 Problem Definition

The problem as considered within this paper can be defined as follows: We are given a warehouse with a storage layout similar to that presented in Fig. 1, i.e. several racks are aligned such that two types of aisles arise: *rack aisles* and *main aisles*, whereas we assume that there are two main aisles with an arbitrary number of rack aisles lying between them. While rack aisles provide access to the racks main aisles only act as an interconnection between the rack aisles and the packing station. We denote by $\mathcal{R} = \{1, \ldots, n_{\mathrm{R}}\}$ the set of racks located in the rack aisles.

In addition, a set of articles $\mathcal{A}$, with $\mathcal{A} = \{1, \ldots, n_{\mathrm{A}}\}$, $n_{\mathrm{A}} \geq 1$, is given. Each article $a \in \mathcal{A}$ is stored at a non-empty set $\mathfrak{R}_a$ of one or more racks, i.e. $\emptyset \neq \mathfrak{R}_a \subseteq \mathcal{R}$. The quantities of articles $a \in \mathcal{A}$ are given by $\mathfrak{q}_a : \mathfrak{R}_a \to \mathbb{N}$.

We assume that a homogeneous fleet of n_{T} trolleys used for carrying collected items exists, each having capacity $\mathfrak{c}$. A group of n_{W} warehousemen, $1 \leq n_{\mathrm{W}} \leq n_{\mathrm{T}}$, is operating these trolleys and issuing ordered articles.

A set of customer orders is given, whereas each order consists of a list of articles with demands to be shipped to a specific address. Further, a latest delivery time to be met is associated with each of these customer orders. Although the assignment of orders to customers is important for a production system we are only interested in the quantities of each article to be collected in the warehouse for this work, since extra workers are assigned to pack all items according to orders. Therefore, we define the set $\mathcal{O}$ of orders as the set of tuples $(a, \mathfrak{d}_a) \in \mathcal{O}$, with $|\mathcal{O}| = n_{\mathrm{O}}$, stating the total integer demand $\mathfrak{d}_a \geq 1$ of each article $a \in \mathcal{A}$. In addition, we assume that the capacities of all trolleys together, i.e. $n_{\mathrm{T}} \cdot \mathfrak{c}$, is greater than the amount of articles to be collected.

Let us denote by set $\mathcal{S}$ a finite set of selections, whereas a selection $S \in \mathcal{S}$ is a set of triples (a, δ, r) such that $r \in \mathfrak{R}_a$, $1 \leq \delta \leq \mathfrak{q}_a(l)$, and there exists an order $(a, \mathfrak{d}_a) \in \mathcal{O}$ with $\mathfrak{d}_a \geq \delta$. Further, we denote by $\mathcal{T}$ a set of tours whereas

for each $S_i \in \mathcal{S}$ a tour $T_i \in \mathcal{T}$ exists. By tour T_i we understand a walk through the warehouse visiting all locations contained in S_i such that the corresponding items of S_i can be collected. The length of tour $T_i \in \mathcal{T}$ is denoted by $c(T_i)$.

A solution $x = (\mathcal{S}, \mathcal{T}, \Pi)$ to the given problem consists of a set $\mathcal{T}$ of tours corresponding to the selections in $\mathcal{S}$ as well as a mapping $\Pi : \mathcal{T} \to \{1, \ldots, n_{\mathrm{W}}\}$ of tours to workers such that

- $|\mathcal{S}| = |\mathcal{T}| \leq n_{\mathrm{T}}$,
- tour $T_i \in \mathcal{T}$ collects all articles $a \in S_i$, $S_i \in \mathcal{S}$,
- $\sum_{S \in \mathcal{S}} \sum_{(a,\delta,l) \in S} \delta = \mathfrak{d}_a$, for all $a \in \mathcal{A}$,
- worker $\Pi(T_i)$ processes tour $T_i \in \mathcal{T}$,
- all time constraints are met, i.e. for each customer order it has to be guaranteed that tour $T \in \mathcal{T}$ picking up the last not yet collected article must be finished before the specific delivery time.

We formulate the given problem as an optimization problem in which the total length of the tours, i.e. $\sum_{T \in \mathcal{T}} c(T)$, as well as the violations of the capacity constraints defined by the trolleys, i.e. $\sum_{S \in \mathcal{S}} \max\left\{\sum_{(a,\delta,l) \in S} \delta - \mathfrak{c}, 0\right\}$, should be minimized. For weighting the relative importance of violating capacity constraints compared to tour lengths, we introduce a weighting coefficient ω such that the objective function can be written as

$$\min \sum_{T \in \mathcal{T}} c(T) + \omega \cdot \sum_{S \in \mathcal{S}} \max\left\{\sum_{(a,\delta,l) \in S} \delta - \mathfrak{c}, 0\right\} \tag{1}$$

For this work, ω is set to the maximum possible length of one tour picking up one article plus one. Such a choice for ω implies that it is always better to use an additional tour for collecting an item than violating a capacity constraint.

3 Related Work

Obviously, the stated problem is related to warehouse management in general. An introduction to this topic as well as an overview over tour finding, storage management and other related tasks is given in [1].

It is obvious that the stated problem forms a special variant of the well known *vehicle routing problem* (VRP) [2]. In fact, the classical VRP is extended by additional domain specific constraints. In the classical VRP one wants to find a set of tours minimal with respect to their total length starting at a depot and visiting a predefined set of customers. Further, the problem studied here is related to the *split delivery VRP* [3], the *VRP with time windows* [4] and the *capacitated VRP* [5]. To our knowledge, there exists so far no previous work considering a combination of these variants of VRP in connection with the additionally defined constraints.

Beside this obvious relationship with VRPs, this problem is also related to the *generalized network design problems* [6] with respect to the possibility to collect one article from different locations within the warehouse. At a time only one node of such a cluster has to be visited.

4 A Hybrid Variable Neighborhood Search Approach

Based on the fact that the problem examined within this paper is strongly related to the VRP, we expect that exact approaches are limited to relatively small instances. In addition, short computation times are important, since the observation was made that new orders are committed continuously by customers which implies that the algorithm is restarted frequently. Since recently highly effective *variable neighborhood search* (VNS) [7] approaches have been reported for diverse variants of the VRP [8, 9] we also based our approach on a similar concept. Within our hybrid VNS, *variable neighborhood descent* (VND) [7] is used as embedded local search procedure, and subproblems corresponding to the computation of individual tours for collecting particular items are solved by means of dynamic programming [10], exploiting the specific structure of the warehouse.

4.1 The Basic Principle

In this work, we assume that all orders stated by customers can be fulfilled with respect to the capacity constraints defined by the trolleys, i.e. the total capacity of the trolleys is not exceeded. Anyhow, in real-world settings the problem may arise that these constraints cannot be satisfied. In that case a straightforward preprocessing step is used, which partitions the set of orders such that for each partition the capacity constraints are satisfied.

The tour planning algorithm mainly consists of two parts: (1) the allocation of articles to at most n_{T} selections and (2) the computation of concrete routes through the warehouse for collecting all items assigned to the previously determined selections. Anyhow, both of these parts have to be executed intertwined, since the evaluation of the mapping of articles to tours is based on the lengths of these tours. Therefore, these two steps are repeated until no further improvement can be achieved. Finally, an assignment of the walks to n_{W} warehousemen is done, such that the latest finishing time is as early as possible. As soon as additional orders are committed by customers the whole algorithm is restarted from the beginning. This can be done efficiently by using a straightforward incremental update function.

4.2 Assignment of Articles to Tours

One crucial point of our algorithm is the assignment of articles to selections such that in a second step walks through the warehouse can be computed. Nevertheless, the capacity constraints stated by the trolleys as well as the maximum number of available trolleys n_{T} have to be regarded during this allocation step.

Construction Heuristic For quickly initializing our algorithm we developed a construction method called *collision avoiding heuristic* (CAH). The main idea of CAH is to divide the storage into $m \geq 1$ physically non-overlapping zones

whereupon each one is operated by one trolley, i.e. m selections are generated. For this work, we set m to n_{W}.

Since the capacities of the trolleys are not regarded within this initialization procedure the solution qualities produced by this heuristic are not outstanding. The required computation times are, however, very low. Therefore, CAH can be used for providing *ad hoc* solutions such that the workers start collecting the first scheduled item while the rest of the tours is improved in the meantime.

Improvement Heuristic For improving solutions generated by CAH, we present a *variable neighborhood search* (VNS) approach using an adapted version of *variable neighborhood descent* (VND) as subordinate. The basic idea of VNS/VND is to systematically swap between different neighborhood structures until no further improvement can be achieved. In fact, the crucial task in designing such an approach is the proper definition of appropriate moves used for defining the neighborhood structures incorporated in VNS and VND, respectively. In our approach, the following seven different move types were implemented:

BreakTour(i) Selection $S_i \in \mathcal{S}$ is removed from $\mathcal{S}$ and all articles assigned to S_i are randomly distributed over all other selections $S_j \in \mathcal{S} \setminus S_i$.

MergeTour(i, j) Selections $S_i \in \mathcal{S}$ and $S_j \in \mathcal{S}$ are both removed from $\mathcal{S}$ and merged with each other into a new selection $S_{i'}$, which is then added to $\mathcal{S}$.

ShiftArticle(i, j, a) Any solution generated by this move differs from the underlying solution in one article a which is moved from selection S_i to selection S_j, with $a \in S_i$ and $S_i, S_j \in \mathcal{S}$.

ShiftArticleChangeRack(i, j, a, r) Analogously to the ShiftArticle move, this move shifts an article $a \in S_i$ to selection S_j, with $S_i, S_j \in \mathcal{S}$. In addition to this, a is now collected from rack $r \in \mathfrak{R}_a$ regardless of the position it was acquired before.

SplitTour(i) By applying this move, selection $S_i \in \mathcal{S}$ is split into two new selections $S_{i'}$ and $S_{i''}$ such that $|S_{i'}| = |S_{i''}|$ or $|S_{i'}| = |S_{i''}| + 1$. Selection S_i is removed from $\mathcal{S}$, whereas $S_{i'}$ and $S_{i''}$ are added.

SwapArticle(i, j, a_1, a_2) This move swaps two articles $a_1 \in S_i$ and $a_2 \in S_j$, with $S_i, S_j \in \mathcal{S}$.

SwapArticleChangeRack$(i, j, a_1, a_2, r_1, r_2)$ This move is similar to the SwapArticle move. After swapping articles $a_1 \in S_i$ and $a_2 \in S_j$ between selections $S_i \in \mathcal{S}$ and $S_j \in \mathcal{S}$, the rack of a_1 is changed to $r_1 \in \mathfrak{R}_{a_1}$ and that of a_2 is changed to $r_2 \in \mathfrak{R}_{a_2}$.

Based on these move types, the neighborhood structures for VNS and VND are defined, whereas the neighborhoods $N_1(x), \ldots, N_{k_{\max}}(x)$, with $1 \leq k_{\max} \leq |\mathcal{S}| - 1$, used within the shaking phase of VNS are purely based on BreakTour moves such that within $N_k(x)$, with $1 \leq k \leq k_{\max}$, k randomly chosen BreakTour moves are applied to x. The neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_6$ for VND are defined by a single application of one of the other six move types, such that $\mathcal{N}_1, \ldots, \mathcal{N}_6$ apply SplitTour, MergeTour, ShiftArticle, ShiftArticleChangeRack, SwapArticle, SwapArticleChangeRack, respectively.

In addition to the proper definition of neighborhood structures, a beneficial order used for systematically examining them is necessary and has a great influence on the performance of VND (cf. [11, 12]). Preliminary tests showed that the contributions of neighborhood structures $\mathcal{N}_1$ and $\mathcal{N}_2$ are relatively high during the beginning of VND but dramatically decrease after only a few iterations. This is due to the fact that splitting and merging of tours is only important as long as the capacity constraints are either violated or highly over-satisfied, i.e. there is significant capacity left in more than one trolley. Anyhow, in most of the iterations, i.e. in about 95% of the iterations, no improvement can be achieved by these neighborhoods. Therefore, some dynamic order mechanism guaranteeing that neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ are primarily examined during the beginning phase of VND while being applied less frequently during the later iterations seems to be important.

In contrast to self-adaptive VND as proposed by Hu and Raidl [11], we do not punish or reward neighborhood structures based on their examination times, but reorder the neighborhoods according to their success rates, i.e. the ratios of improvements over examinations, only. The neighborhood order is updated each time an improvement on the current solution could be achieved.

In addition, we adapted VND such that not only improvements on the current solution are accepted but also moves can be applied which leave the current objective value unchanged. To avoid infinite loops, at most ten non-improving subsequent moves are allowed in our version of VND, whereas the i-th non-improving move is accepted with probability $1 - (i - 1) \cdot 0.1$, only. All counters regarding the acceptance of non-improving moves are reset as soon as an improvement could be achieved. As step function a next improvement strategy was implemented, whereas a random examination order was chosen to uniformly sample the current neighborhood.

4.3 Computing Individual Tours

Another crucial point of the proposed algorithm is the computation of concrete tours which will be used by the warehousemen for collecting a specific set of ordered items. Although the decision which article to collect next will finally be made by the workers themselves, the system provides them a suggestion. Further, the evaluation of the assignment of articles to selections is based on the shortest possible tours, and therefore an efficient tour computation is needed.

Please note that a tour as used within this work does not correspond to tours as used within works related to the traveling salesman problem or the VRP. In fact, the main difference lies therein that tours within a storage are allowed to visit each point of interest, i.e. among others the *packing station*, crossings of aisles and rack positions, more than once. This is simply induced by the circumstance that in most cases no direct connection between two points of interest exists. Consequently, paths between points of interest can be walked along more than once within one tour. Anyhow, an upper bound for the number of times the same passage is walked can be provided based on the following two observations.

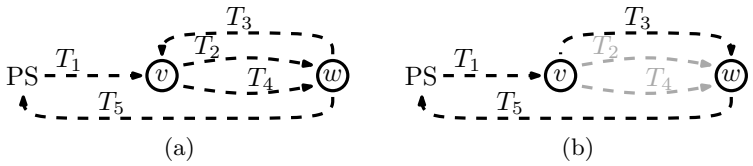

Fig. 2: How to construct a tour T' from a given tour T under the assumption that T visits two times location w immediately after location v.

Theorem 1. *Given is a tour T, which is of shortest length with respect to a set of points of interest, i.e. all of these points are visited by T. Further, we assume that there exist two adjacent points of interest v and w which are twice visited immediately consecutively in T. Then the passage between v and w is once traversed from v to w and once vice versa in T.*

Proof. Let us assume that the passage between points v and w is traversed twice in the same direction. Then, we can split tour T into five subwalks T_1, T_2, T_3, T_4 and T_5 as shown in Fig. 2a, whereas PS denotes the packing station. A new tour T' can be built by passing segment T_1 from PS to v followed by traversing walk T_3 from v to w and finally walking along T_5 from w to PS, see Fig. 2b. Since v and w are adjacent, i.e. no other point of interest has to be visited when walking from v to w, T' visits the same points of interest as T. Furthermore, since subwalks T_2 and T_4 are not traversed within T', T' is shorter than T, which is a contradiction to the assumption that T is optimal.

Lemma 1. *Given is an optimal tour T with respect to a set of points of interest. Then any two adjacent points v and w are visited at most twice immediately consecutively by T.*

Proof. This lemma directly follows from Theorem 1. Under the assumption that points v and w are visited more than two times immediately consecutively the passage between these two points has to be traversed at least twice in the same direction.

Based on the special structure induced by warehouse layouts similar to that shown in Fig. 1, we define *aisle operations* (AOs) and *inter-aisle operations* (IOs). While AOs are representations of the walks to be performed within rack aisles, IOs correspond to movements in main aisles. In Fig. 3 the sets of basic AOs and IOs are shown. By appropriately combining these basic operations, so-called modules can be defined, which will then be used for representing parts of tours, for an example see Fig. 4a. Based on Theorem 1 it can be concluded that the number of different module types needed for representing a tour is limited.

Although it is now obvious that tours can be built by selecting an appropriate module for each aisle to visit, it can be observed that the resulting tours may contain subtours, which are not connected to the rest of the tour, see for example Fig. 4b. Unfortunately, as shown in Fig. 4c, the decision whether a combination

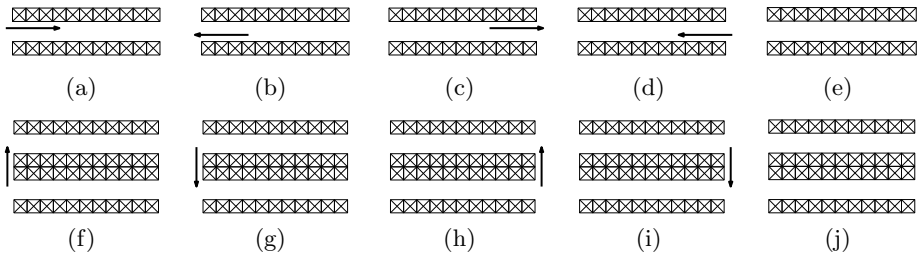

Fig. 3: In (a)–(e) the five basic aisle operations (AOs) are presented, whereas (f)–(j) show basic inter-aisle operations (IOs).

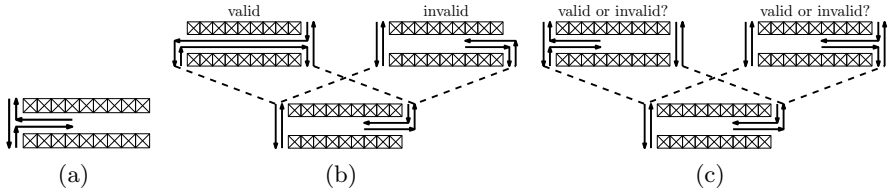

Fig. 4: Figure (a) shows a module representing that part of a tour entering and leaving the RA from and to the left side. This aisle operation is then suitably joined with the rest of the tour by appropriate inter-aisle operations. For some combinations of modules (b) it can be directly decided whether or not they are valid. In other cases (c) this decision has to be postponed.

of modules is valid cannot always be made as soon as the next module is selected. Let us denote by $\mathcal{N}_{\mathrm{c}}(j)$ the set of those modules j' which might be connected with module j with respect to the IOs of j and j', i.e. all modules j' forming together with j possibly valid tour parts. Further, we denote by $\mathcal{N}_{\mathrm{v}}(j)$ the set of those modules $j' \in \mathcal{N}_{\mathrm{c}}(j)$ such that the usage of modules j and j' results in a definitely valid tour (part).

Therefore, we introduce two $(n+1) \times (\nu)$ matrices σ and τ, with n being the number of aisles containing items to be selected and ν indicating the maximum number of potentially used module types. An entry σ_{ij}, with $1 \leq i \leq n$ and $1 \leq j \leq \nu$, corresponds to the length of a valid tour T' which visits all rack locations in aisles 1 to i storing articles to be shipped to customers and performs in aisle i the operations corresponding to module j. Analogously, an entry τ_{ij} corresponds to the total length of tour parts which visit all racks in aisles 1 to i storing articles to be shipped and perform in aisle i the operations corresponding to module j. Anyhow, these tour parts need not to be connected with each other and therefore it has to be assured that they are going to be joined into one (big) tour by operations performed in any aisle $> i$. Now, let us assume that $c_i(j)$ denotes the length of the tour part(s) represented by module j when applied to aisle i and module μ represents the IOs necessary for reaching the first aisle from

the packing station. Then, the entries of σ and τ can be computed by using the following recursive functions:

$$\sigma_{0\mu} = \tau_{0\mu} = 0 \tag{2}$$

$$\sigma_{0j} = \tau_{0j} = \infty \qquad \text{for } j \in \{1,\ldots,\nu\} \setminus \{\mu\} \tag{3}$$

$$\sigma_{ij} = c_i(j) + \min \left\{ \begin{array}{l} \{\sigma_{i-1j'} : j' \in \mathcal{N}_{\mathrm{v}}(j)\} \cup \\ \{\tau_{i-1j'} : j' \in \mathcal{N}_{\mathrm{v}}(j)\} \end{array} \right\} \quad \begin{array}{l} \text{for } i \in \{1,\ldots,n\} \\ \text{for } j \in \{1,\ldots,\nu\} \end{array} \tag{4}$$

$$\tau_{ij} = c_i(j) + \min \left\{ \begin{array}{l} \{\sigma_{i-1j'} : j' \in \mathcal{N}_{\mathrm{c}}(j)\} \cup \\ \{\tau_{i-1j'} : j' \in \mathcal{N}_{\mathrm{c}}(j)\} \end{array} \right\} \quad \begin{array}{l} \text{for } i \in \{1,\ldots,n\} \\ \text{for } j \in \{1,\ldots,\nu\} \end{array} \tag{5}$$

For decoding the optimal tour, one first needs to identify module J used for aisle n in an optimal tour, i.e. $J = \arg\min_{j\in\{1,\ldots,\nu\}}\{\sigma_{nj}\}$. Then, the computations based on Eq. (4) and (5) have to be performed backwards. Anyhow, it can be easily proven that $\sigma_{nJ} \neq \infty$ definitely holds. In case of ties any module can be chosen.

4.4 Assignment of Workers to Tours

In a final step, an assignment of workers to tours has to be computed such that the latest finishing time is as early as possible while regarding the guaranteed delivery times. For this purpose, we implemented a General VNS scheme using a greedy construction heuristic and random moves for the shaking phase. There are three different move types defining neighborhood structures for VND: The first one reassigns one tour from one worker to another worker. The second one swaps two tours between two workers. The third move type rearranges the schedule of one worker such that one tour is shifted towards the beginning of the current working day. The used neighborhood order corresponds to this order and is fixed. A first improvement strategy is used as step function in VND and random moves are applied for the shaking phase of VNS.

5 Experimental Results

For evaluating the performance of the proposed method several test runs were performed. Our algorithm was implemented in Java and all tests were run on a single core of a Dual Opteron with 2.6GHz and 4GB of RAM. All instances were randomly generated based on the characteristics of real-world data, i.e. storage layouts and typical customer orders, provided by our industry partner Dataphone GmbH.

For each of the 20 instances, we performed 40 independent runs, whereas for the half of those runs we allowed that workers may reverse within one aisle, while for the remaining ones we assured that once an aisle is entered, it is completely traversed by the warehousemen. To avoid excessive computation times, a time limit of 1200 seconds for VNS was applied. See Tab. 1 for a detailed listing of the obtained results. The column labled init presents the initial values

Table 1: Average results over 20 runs for 20 instances. The initial values, the objective values (including standard deviations in parentheses), the number of tours (n_T), the average filling degree of the trolleys used (quota) and the average computation times in seconds are opposed for instances allowing to reverse in aisle and disallowing turning around. The last column presents the p-values of an unpaired Wilcoxon rank sum test, for evaluating whether the tours with turning around are 20% shorter than those without reversing.

		reversing disabled				reversing enabled				
inst.	init	objective	n_T	quota	time	objective	n_T	quota	time	p-Val
(01)	3750	3059.0 (132.6)	3.0	74.2	37.5	2281.0 (44.7)	3.0	74.2	35.5	<0.01
(02)	4200	3213.0 (156.5)	3.7	79.3	29.2	2348.0 (85.6)	3.5	85.0	59.1	<0.01
(03)	4260	3560.0 (109.0)	4.0	75.5	47.5	2541.5 (80.0)	4.0	75.5	78.7	<0.01
(04)	3210	2962.0 (35.8)	3.0	91.8	22.6	2298.5 (4.9)	3.0	91.8	30.6	<0.01
(05)	4020	3605.0 (105.6)	4.1	88.1	37.8	2466.5 (39.1)	4.0	90.3	55.5	<0.01
(06)	5060	4671.5 (106.7)	5.7	78.3	61.2	3576.0 (74.9)	5.4	82.0	114.6	<0.01
(07)	6050	5575.5 (79.3)	7.0	81.7	83.8	4052.5 (73.5)	6.8	82.9	192.7	<0.01
(08)	5650	5602.5 (155.8)	7.0	84.9	95.6	4159.5 (98.9)	7.0	85.5	212.6	<0.01
(09)	7000	6583.0 (246.9)	8.0	86.4	132.5	4887.5 (117.6)	8.0	86.4	334.1	<0.01
(10)	5070	4995.5 (252.1)	6.0	78.8	88.2	3589.5 (104.1)	5.8	82.2	128.8	<0.01
(11)	10740	9254.0 (268.4)	12.2	81.9	391.4	6158.5 (149.3)	11.1	90.4	845.4	<0.01
(12)	9350	8155.0 (175.9)	12.6	79.3	255.4	5952.0 (136.9)	11.7	85.4	689.0	<0.01
(13)	9970	8939.0 (256.2)	12.0	83.2	323.9	6102.0 (156.7)	11.3	88.0	715.7	<0.01
(14)	9520	9082.5 (246.5)	12.6	79.6	370.7	6165.5 (181.2)	11.7	85.8	864.3	<0.01
(15)	7690	7473.0 (270.4)	11.5	86.9	279.2	5860.5 (74.9)	11.2	89.2	673.0	0.02
(16)	11510	8878.0 (240.3)	12.2	81.9	716.4	6465.0 (165.9)	11.7	85.8	1200.0	<0.01
(17)	11460	8251.5 (216.7)	12.3	80.9	782.2	6261.5 (161.7)	11.7	85.4	1200.0	<0.01
(18)	11740	8520.0 (187.8)	12.6	79.3	748.5	6238.5 (159.9)	11.5	86.9	1200.0	<0.01
(19)	11480	8990.0 (216.8)	12.1	82.6	828.3	6349.0 (207.0)	11.7	85.8	1200.0	<0.01
(20)	12260	9644.5 (286.9)	12.6	79.6	819.0	6635.0 (164.4)	11.8	85.0	1200.0	<0.01

obtained by the so called s-shaped heuristic [1], whereas we simply try to collect as much articles as possible during one tour regarding the capacities of the trolleys used. The objective values provided within the nex columns correspond to the weighted sum as presented in Eq. 1. The standard deviations (presented in parentheses) are relatively low with respect to the tour lengths. Taking a look at these average values it can be observed that the tour lengths can be reduced by about 20% on average when it is allowed to reverse within an aisle. To statistically confirm this observation, we performed an unpaired Wilcoxon rank sum test. The corresponding p-values are shown in the last column of Tab. 1. Regarding the number of tours as well as the filling degree of the trolleys, no significant difference between those runs allowing reversing and those forbidding it can be identified. Finally, taking a closer look at the computation times, it can be observed that for those instances including the option to reverse the running times are longer. This can be reasoned by the fact that the number of aisle operations is more restricted for those runs forbidding reversing. Since the available computation time was limited, the results obtained for instances 16–20 might

Table 2: Average contributions of the individual neighborhood structures to the final solutions. The numbers represent the average ratios of improvements over examinations over 20 runs for 20 instances with 25, 50, 100, 200 different items to be collected (#it.).

		reversing disabled						reversing enabled					
inst.	#it.	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\mathcal{N}_5$	$\mathcal{N}_6$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\mathcal{N}_5$	$\mathcal{N}_6$
(01)	25	3.6	10.6	60.2	34.4	46.9	0.2	5.9	15.9	70.2	61.5	18.2	0.0
(02)	25	4.1	12.0	61.0	32.2	33.8	0.0	5.6	19.7	73.0	46.2	16.3	1.1
(03)	25	3.6	11.8	60.5	40.6	49.9	0.1	4.4	17.1	73.4	55.2	27.9	0.0
(04)	25	4.5	11.1	52.1	27.4	34.6	0.0	7.4	18.2	70.9	42.8	6.9	0.4
(05)	25	6.6	15.2	65.9	29.3	45.2	0.8	9.1	19.9	74.2	58.0	32.3	0.0
(06)	50	10.4	17.6	69.6	3.7	18.7	0.0	10.7	23.3	81.8	53.5	7.6	0.0
(07)	50	12.2	20.7	74.3	6.0	42.0	0.0	12.9	25.7	85.6	59.6	48.2	0.0
(08)	50	13.5	19.5	72.3	1.7	55.9	0.0	19.1	28.6	87.9	29.9	40.2	0.0
(09)	50	13.9	20.3	74.4	41.2	46.3	0.0	17.8	27.8	85.8	46.8	37.4	0.0
(10)	50	9.8	17.4	70.8	9.2	22.3	0.0	12.6	23.1	81.8	66.9	18.5	0.0
(11)	100	16.0	27.1	76.3	7.9	24.6	0.0	21.6	29.4	88.5	44.0	37.0	0.0
(12)	100	18.9	29.9	77.8	0.9	24.7	0.0	18.4	28.8	88.3	47.2	39.2	0.0
(13)	100	16.5	26.2	75.9	5.9	26.7	0.0	23.5	29.3	87.3	60.4	44.9	0.0
(14)	100	14.3	27.6	78.2	1.1	19.5	0.0	19.4	27.2	88.0	50.5	33.8	0.0
(15)	100	18.2	23.2	73.4	1.1	12.8	0.0	20.1	26.7	85.3	51.1	30.8	0.0
(16)	200	18.0	30.3	75.3	2.9	8.1	0.0	25.7	31.5	88.8	46.4	30.5	0.0
(17)	200	17.9	28.8	75.7	0.0	7.9	0.0	27.0	31.8	88.8	39.3	49.2	0.0
(18)	200	19.2	28.6	76.0	3.5	15.0	0.0	28.1	32.2	88.8	53.8	38.6	0.0
(19)	200	16.1	28.1	74.5	1.8	17.3	0.0	26.2	29.8	88.4	35.9	35.1	0.0
(20)	200	16.4	28.2	75.4	1.1	6.7	0.0	27.5	34.7	90.2	46.6	32.8	0.0

be further improved when using looser time limits. Regarding the last step of the algorithm, i.e. the assignment of tours to workers, all violations of the time constraints could be resolved within a few iterations of the corresponding VNS procedure.

Regarding the performance of the proposed neighborhoods, i.e. the ratio of improvements over examinations, we observed that all of them except $\mathcal{N}_6$ contribute substantially to the final solution whereas neighborhood $\mathcal{N}_6$ did almost never add an improvement (see Tab. 2). Nevertheless, for the small instances with 25 articles to be shipped, some improvements could be achieved even by $\mathcal{N}_6$, and therefore we included it here. Regarding the other neighborhood structures, $\mathcal{N}_3$, i.e. the neighborhoods based on the ShiftArticle move, performed best. It is interesting that neighborhood structure $\mathcal{N}_4$ did worse for those instances forbidding turning around. However, this can be explained by the fact that for these instances the degree of freedom is less than for the others which results therein that once an aisle i has to be entered all ordered articles stored within this aisle can be collected with less expenses from aisle i than from any other aisle. The same observation holds for neighborhood structure $\mathcal{N}_5$, although this effect is less prominent for the underlying move type.

6 Conclusions

In this paper, we proposed a new hybrid algorithm combining *variable neighborhood search* (VNS) with dynamic programming (DP) for solving a real-world scheduling and tour finding problem within a spare parts warehouse. For boosting the performance of the neighborhood structures used within *variable neighborhood descent* (VND), we used a new self-adaptive VND rearranging the neighborhoods according to their success rates. Individual optimal tours within the spare parts warehouse are computed by means of dynamic programming, whereas the special structure of the storage is exploited.

Experimental results showed that this approach performs good for instances based on real-world characteristics. Further, we showed that the total tour lengths can be reduced by about 20% on average when reversing within aisles is allowed. Regarding the computation times, our approach is able to provide good results within 1200 seconds which correspond to acceptable time limits in real-world scenarios.

References

1. de Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research **182**(2) (2007) 481–501
2. Toth, P., Vigo, D.: The Vehicle Routing Problem. Number 9 in Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia (2002)
3. Dror, M., Laporte, G., Trudeau, P.: Vehicle routing with split deliveries. Discrete Applied Mathematics **50**(3) (1994) 239–254
4. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research **35**(2) (1987) 254–265
5. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: On the capacitated vehicle routing problem. Mathematical Programming **94**(2–3) (2003) 343–359
6. Feremans, C., Labbe, M., Laporte, G.: Generalized network design problems. European Journal of Operational Research **148**(1) (2003) 1–13
7. Hansen, P., Mladenović, N.: Variable neighborhood search. In Glover, Kochenberger, eds.: Handbook of Metaheuristics. Kluwer Academic Publisher, New York (2003) 145–184
8. Hemmelmayr, V.C., Doerner, K.F., Hartl, R.F.: A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research (2007) In Press. doi:10.1016/j.ejor.2007.08.048.
9. Ostertag, A., Dörner, K.F., Hartl, R.F.: A variable neighborhood search integrated in the POPMUSIC framework for solving large scale vehicle routing problems. In Blesa, M.J., et al., eds.: Hybrid Metaheuristics. Volume 5296 of LNCS., Springer (2008) 29–42
10. Bellman, R.E.: Dynamic Programming. Dover Publications Inc. (2003)
11. Hu, B., Raidl, G.R.: Variable neighborhood descent with self-adaptive neighborhood-ordering. In Cotta, C., Fernandez, A.J., Gallardo, J.E., eds.: Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics. (2006)
12. Puchinger, J., Raidl, G.R.: Relaxation guided variable neighborhood search. In: Proceedings of the XVIII Mini EURO Conference on VNS. (2005)