

NYILATKOZAT

Név: Gál Boglárka

ELTE Természettudományi Kar, szak: Alkalmazott matematikus

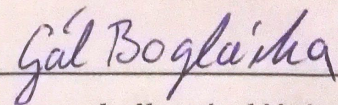
NEPTUN azonosító: D7DSQO

Szakdolgozat címe:

Algorithmic solutions for logistic problems

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2020.12.29.



a hallgató aláírása

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF SCIENCE

Boglárka Gál

**ALGORITHMIC SOLUTIONS FOR LOGISTIC
PROBLEMS**

MSc Thesis

Consultant:

Alpár Jüttner

Department of Operations Research



Budapest, 2020

Contents

1	Introduction	4
2	Problem description	5
3	Dividing into subtasks	6
4	Routing in a warehouse	8
4.1	Solving TSP in a Warehouse	8
4.2	Conclusions	15
4.3	Heuristic solutions by using cross aisles	16
4.3.1	S-shape heuristic	16
4.3.2	Largest gap heuristic	17
4.3.3	Combined heuristic	17
4.3.4	Aisle-by-aisle	20
4.3.5	Conclusions	21
5	Vehicle Routing Problem	23
5.1	What is VRP?	23
5.2	VRP in a warehouse	24
5.3	Examples for solving VRP in a warehouse	25
5.3.1	Branch and Price algorithm	25
5.3.2	Two-phase heuristics	29
6	Batching algorithms	32
6.1	First-Come, First-Served heuristic	32
6.2	2-Dimensional spacefilling curve	32
6.3	4-Dimensional spacefilling curve	35
6.4	Sequential minimum distance heuristic	35
6.5	Order batching heuristic	36
6.6	Conclusions	38
7	Replenishment methods	40
7.1	Storage Replenishment Problem	41
7.2	Prioritizing replenishment	46
7.2.1	Stock-Needs Rule	49

7.2.2	Order-Quantity-Based Rule	49
7.2.3	Conclusions	52
8	Summary	53

1 Introduction

Nowadays, it is a great challenge of business that how companies are able to serve the ever-growing needs of this consumer society in the fastest and the most cost-effective way, as well to gain the biggest profit. To do that, it is obviously required to store goods or products in huge quantities, which are accessible enough. Therefore it's not surprising, that nowadays warehouse management problems get a lot of attention, because the layout of a warehouse, the improvement of picking out and replenishment methods can result in a great benefits.

But why is it so complicated to manage warehouse problems effectively? The complexity of questions is clearly seen if you think about that the most effective way of storage of products (for example on the smallest floor area) may be against to be easy-to-pick in the storage. For example if the floor area of the warehouse is small, then the products can fit it only if the shelves are tall enough. But picking out from a very high shelf is almost impossible. Similarly if the storage area is too large, then the time we need for travelling to pick the items out will increase significantly. Naturally, even if the sizes of the warehouse is given, many further problems will be raised, how to execute the picking out of items in the fastest way, which way the forklifts should go, which products should be collected together, and what happens if the shelf goes out of stock. The problem has become more and more serious by the increasing quantity of products, which has to be picked out.

The goal of this thesis is to provide an insight of the algorithmic methods of warehouse management, and to investigate the solutions of each subtasks by seeking the solution of a greater comprehensive problem.

During solving warehouse management problems, the interests of business cannot be ignored and also, current operation of the warehouse, the capability of storage, tools and the available labor could be essential. These parameters can completely change the constraints and objectives of the problem, therefore business decisions have a big impact on what algorithm will be the most suitable to meet demands. Thus we are often unable to determine the absolute best algorithm, because if a method is appropriate for a certain environment, it may be not applicable for another one. In this thesis I did not insist on fitting a certain business demand, but I made an effort to investigate the algorithms more extensively, and we will discussed which methods for what business needs are suitable.

2 Problem description

In this thesis I have raised a relatively actual problem, as a task to be solved. But course of solving the problem, I examined a more comprehensive range of possibilities, and so provide an insight of a certain part of warehouse management. Namely, I divide the main problem into smaller subtasks, which are individual problems in the literature. After examination of the solutions of subtasks, I have proposed how to connect the sub-solutions in order to give a solution to the main problem.

The problem mentioned in the introduction is that the storage of goods with large quantities makes picking operations difficult. This is usually tackled by dividing of the storage area into two parts: to a *forward* and *reserve area*. The reserve area is suitable for storing large quantities of products. Picking out products from here is rare but if it is necessary, a large amount of products will be picked out. In the forward area we store a smaller amount of products, usually in a smaller place. In the forward area, picking operations are frequently performed in a smaller quantity, therefore the idea is to design this section so that the picking operation can be performed as quickly as possible. By the division of storage area, it can be solved how to store great quantities of products and picking them quite quickly in the same time. But the disadvantage of this layout is that we must frequently replenish products in the forward area from the reserve area, which raises another logistic problem. Namely, we should perform the replenishment coordinated with the picking operation, because it would not be well advised to shut off or impede picking operations to frequently, still we want the access of products to be ensured continuously.

We focus on a certain warehouse where the layout of storage (forward and reserve area), the location of shelves, the location of products on shelves, and the capacity of locations are already fixed. The capacity of a location means how many items can be placed on a given location. It is important to stipulate this in advance, because it plays a major role in the optimization of warehouse management operations. (If you are interested in this, you can see more about what size to choose for the forward and reserve area in [5], [4], [7], [8], how to place the shelves, namely how much aisle and cross aisle are suggested to create in [10], [11], and how to assign products to item locations and how much quantity of them should be stored in the forward area in [1], [2], [3], [6], [9].)

There are given a set of products I stored in the warehouse. We refer to these products as *items*. Each item can be found both in the reserve and the forward area. The coordinates of locations of items are given and known. In each area, one item can be found only at one location, and at each location can be found only one kind of items. All items have a

known weight and size (i.e. width, height, and depth). Both the size and the load capacity of the shelves are known, therefore we can determine what amount of item can be placed on each shelves (i.e. locations). This is called the capacity of the location.

The items are collected and replenished to the forward storage area by some kind *vehicles* (for example forklifts, trolleys and push carts). The vehicles can differ both in size and load capacity, and they are stored in different places in warehouse depending on its type. We use different vehicles for replenishment and picking. This is necessary because the amount of the delivered products can be greater during replenishment than picking. This constraint that the vehicles used for picking is not involved in replenishment operations and vice versa.

There are given a set of *orders* M . An order includes a list of items with their quantities. For orders, all we enforce is that an order from a product can not contain more than the location capacity in the forward area. There are given a certain time interval T (i.e. the planning horizon), in which the items from orders in M can be picked. The main task is to collect items from each order in the planning horizon in the forward area with vehicles so that the quantity of products carried by a vehicle must not exceed the capacity of this vehicle. In order to ensure the availability of items on each order, we have to replenish the forward area from the reserve area using the charging vehicles. The goal is to minimize the travel time needed to collect the items, such that one vehicle can be used more than once. Items have to be shipped to a pre-designated location. In other words, the task is to collect orders to be served on a given day as quickly as possible. Or the task can be considered so that we want to serve as many orders as possible within a fixed time interval (for example one day).

We also have to determine how to deal with the situation if we can not pick as much as item out we want, because it is sold out in forward area. We may set up a constraint which guarantees that such a case can not happen or we can make an objective to minimize the number of situations, when picking can not be performed because of stock out. Both cases will be discussed later.

3 Dividing into subtasks

Note that our problem is closely related to the Vehicle Routing Problem (VRP). The most basic type of the VRP is the capacitated VRP (CVRP), where we have to serve the demands of the costumers from a central depot with vehicles of uniform capacity, and

then return such that we have to minimize the sum of the travel time. (See more [12], and in section 5). In our problem, customers correspond to the locations of items, and the demands of the customers correspond to the amount of items to be picked. Obviously, it can be seen that our problem to be solved is more complex than a CVRP. Although many heuristics are known for trying to solve each type of VRP (e.g. [12]), these ones are still not appropriate for solving warehouse management problems. Namely, these can not handle the questions in what sequence the orders should be picked, which orders should be picked together, and how to solve replenishment problems. So, the solutions given for VRP can be used effectively for warehouse management (see later in chapter 5), but these solutions are not sufficient to answer all questions in warehouse management. We need to look for other options.

On the other hand, it can be seen that it is not a good idea to model the complete task with one IP problem. This is because the size of the IP soon becomes unmanageable. For example the Vehicle flow models used by [12] even the routing constraints yield exponential number of constraints, and because of the large number of orders, we may expect a great amount of variables. Because of the complexity of the problem, it seems to be the best possibility if we divide the problem into several subtasks, and we examine them as individual problems.

But what kind of subtasks can be identified? Once we have determined what products a vehicle should collect in each round, then we have to solve a Travelling Salesman Problem. In this case the only question is in which sequence the items have to be collected, namely in which sequence the item locations have to be visited in order to minimize the total travelling time. This will be discussed in Section 4. If the total weight or size of items corresponding to one order exceeds the vehicle capacity, we have to use more vehicle to collect them. This is a typical Vehicle Routing Problem, which will be discussed in Section 5. If orders are small enough to collect them with one vehicle, we have to determine which orders should be collected together to pick them as soon as possible. The *batching algorithms* that can be used for this will be discussed in the 6 chapter. Finally, the replenishment policies and procedures are described in the 7 Section. It will be discussed how to link these subtasks together in Chapter 8.

4 Routing in a warehouse

In this section, we try to solve the following problem: let us given a warehouse with known layout of the shelves, and the locations of the items. We have a single vehicle, which we want to use to collect a pre-recorded list of products. We can ignore the capacity of the vehicle (namely, we can assume that the total weight of items does not exceed the vehicle capacity). The question is in which sequence the item locations should be visited to minimize the total travelling time.

The problem described above resembles to a TSP problem. However, we want to solve a special case of TSP rather than a general TSP. Namely, we can use the fact that the vehicle can move only between the shelves of the storage and its arrangement is known in advance. In this special case there is an algorithm with polynomial running time (in fact it is linear in the number of aisles) which have been proposed by Ratliff and Rosenthal in 1983 [13].

But what is that certain special arrangement that characterizes a storage? Most warehouses have a rectangular shape and shelves are arranged side by side in parallel. Products can be placed on both sides of a shelves, and we can travel on the aisles between these shelves. It is called *pick aisle*. Changing pick aisle can take place in horizontally located corridors, of which there are basically two in the warehouse: one in front of the shelves and one behind them. It may occur more horizontal aisle in a storage called *cross aisle*. (See later is chapter 4.3 and in [11]). There is no picking in the cross aisle, this can only be used for changing corridors. In the following, it will be described how this arrangement can help to solve the TSP task.

4.1 Solving TSP in a Warehouse

In this section will be described, how to solve the problem in a warehouse where there are only two cross aisles: behind and front of the shelves. We assume that the vehicle starts and arrives at the same place called *depot*. The procedure presented now comes from [13]. The floor map of the warehouse described above can be seen in Figure 1. The rectangles are the shelves and the little circles correspond to the item locations to be visited. The vehicle can move on aisles between the rectangles. The depot is denoted by a hatched rectangle under the fourth aisle.

Suppose we have to collect an order m items. We can represent the warehouse with a graph G with vertex set $V = \{v_0, v_1, \dots, v_m, a_1, b_1, a_2, b_2, \dots, a_n, b_n\}$ where v_0 denotes the

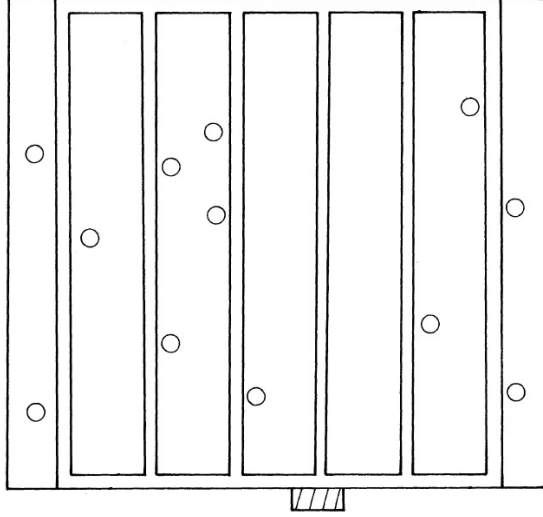


Figure 1: Layout of the considered warehouse from [13]

depot, the items to be collected are v_i where $(i = 1, \dots, m)$, and n corresponds the number of aisles, and a_j and b_j are the endpoints of the aisle j where $j = 1, \dots, n$. In this graph two vertices are connected if they are found on the same aisle and they can be collected one after the other with the vehicle. The first and last item on aisle j are connected to a_j and b_j . In addition, there is an edge between all vertices a_j and a_{j+1} and similarly between b_j and b_{j+1} where $(i = 1, \dots, n - 1)$. Between the locations there can be quite a lot of number of edges. The weight of an edge is defined as the corresponding distance between the two locations in the warehouse. Vertex v_0 is placed on that aisle nearest to the depot. The weight of the corresponding edge is zero. So the resulting graph can be seen in Figure 2.

Thus our task is to find a circle containing all vertices v_i at least once. A circle may contain all edges at most once. Such a circle will be called *tour*. The goal is to find a tour of minimum length.

Definition 4.1. We call a subgraph $T \subset G$ spanning all vertices v_i a *tour subgraph* if it contains a tour as subgraph.

Theorem 4.2. A subgraph $T \subset G$ is a tour subgraph if and only if the following statements are true.

- All vertices v_i have a positive degree in T where $(i = 0, \dots, m)$
- T has only one component, which is not an isolated point.

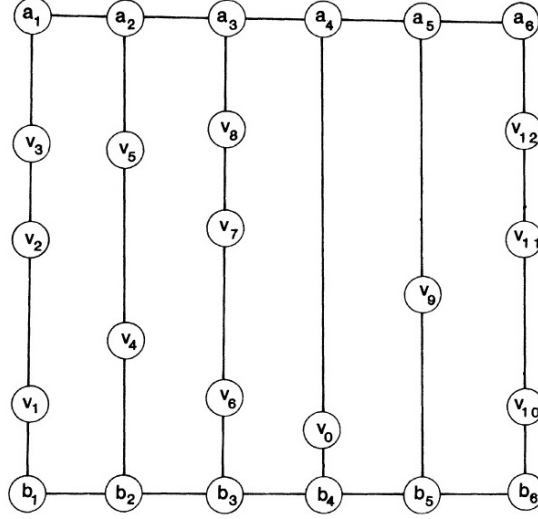


Figure 2: Graph representation of the warehouse from [13]

- All vertices in T have an even number of degrees (it means zero, too).

Indeed, correctness of the theorem can easily be seen, because it is a special case of the Euler theorem for graphs (see more in [14]). Later we will prove if we find a minimal length tour subgraph then we can construct easily a minimal length order-picking tour what we are looking for. Therefore, in the following we will construct a minimal length tour subgraph to get a minimal length order-picking tour.

Corollary 4.3. A tour subgraph with minimal length does not contain more than two edges between each pair of vertices.

Theorem 4.4. If we take a (P, P') partition of vertices of a tour subgraph, there exists an even number of edges, which has an endpoint in P and another endpoint in P' .

We will use the following concepts

Definition 4.5. For a fixed subgraph $L \subset G$ we call a $T_j \subset L$ subgraph L *partial tour subgraph* (L PTS for short), if there exists a $C_j \subset G - L$ such that $T_j \cup C_j$ is a tour subgraph in G . We call the subgraph C_j the *completion* of T_j .

Let L_j^- denote the subgraph of G which contains vertices a_j and b_j and includes all points of G , which can be found left to a_j and b_j . Let A_j be the subgraph of G , which contains both a_j and b_j together with all vertices between them. Furthermore let L_j^+ to be equal to $L_j^- \cup A_j$. The use of notation L_j in the statements below means it is true for both L_j^+ and L_j^- .

Theorem 4.6. *A $T_j \subset L_j$ is a L_j PTS if and only if the following assumptions hold:*

- *All vertices v_i in L_j have a positive degree in T_j .*
- *All vertices of T_j (maybe except for a_j and b_j) have an even degree or zero degree.*
- *Except for the vertices with zero degree T_j does not have connected component, or it has one connected component, which contains at least one of a_j and b_j , or it has two connected component where the one contains a_j and the other contains b_j .*

Proof.

Note that the total degree of T_j is even, thus if either of degree of a_j or b_j is odd, then both have odd degree.

First we show sufficiency. Suppose that a_j and b_j have odd degree in T_j . Let C_j be that subgraph, which contains all vertices of $G - L_j$, and has exactly two edges between each pair of vertices in A_k , where $k = j, j+1, \dots, n-1$ (in case of L_j^+ : $k = j+1, j+2, \dots, n-1$), and has exactly one edge between each pair of vertices in A_n , and contains one edge between both a_k and a_{k+1} , and b_k and b_{k+1} (where $k = j, j+1, \dots, n-1$). In this case $T \cup C_j$ satisfies the constraints of , so we have found an appropriate C_j , therefore the definition of L_j PTS fulfills for T_j .

Now suppose that a_j and b_j both have even degree, or they have zero degree or the one has even degree and the other has zero degree. Let C_j be that subgraph, which contains all vertices of $G - L_j$, and all points of them are connected with exactly two edges. In this case $T_j \cup C_j$ satisfies the constraints of , and so the theorem has fulfilled based on the definition of L_j PTS.

Let see necessity. Let $T_j = T - T \cap (G - L_j)$ for a T tour subgraph. Note that there are no edges between vertices of T_j and $T \cap (G - L_j)$, maybe except for a_j and b_j . Since the first two constraints hold for T , then they have to hold for T_j , too. Except for the vertices with zero degree, T is connected, hence T_j does not have such a connected components, which do not contain a_j or b_j . \square

Definition 4.7. Two L_j PTS's, T_j^1 and T_j^2 are *equivalent* if any $C_j \subset G - L_j$ is such that if $T_j^1 \cup C_j$ is a tour subgraph, then $T_j^2 \cup C_j$ is also a tour subgraph.

The next theorem will be stated without proof. (The proof can be found in [13].)

Theorem 4.8. *Two L_j PTS T_j^1 and T_j^2 are equivalent if the following statements hold*

- *The degree parity of a_j is the same in both, and the degree parity of b_j is the same, too.*

- *Except for the zero degree vertices T_j^1 and T_j^2 do not have connected component, or they have one connected component, which contains at least one of a_j and b_j , or they have two connected component so that the one contains a_j and the other contains b_j .*

Based on this theorem we can construct equivalence classes of L_j PTS's according to the degree parity of a_j and b_j and according to the connectivity. To identify equivalence classes we use the following notations: we denote zero degree with 0, even degree with E , and odd degree with U , which corresponds to "uneven". Denote the single component $1C$, and two component $2C$. For example if we take an equivalence class in which both a_j and b_j are odd degree, and the L_j PTS consists of only one component except for the zero degree vertices, then this equivalence class will be referred to $(U, U, 1C)$.

Corollary 4.9. The all possible equivalence classes of L_j PTS's are $(U, U, 1C)$, $(0, E, 1C)$, $(E, 0, 1C)$, $(E, E, 1C)$, $(E, E, 2C)$, $(0, 0, 0C)$, $(0, 0, 1C)$.

Note that $(0, 0, 0C)$ is possible only if none of the aisles in L_j contains any item to be picked, and $(0, 0, 1C)$ is possible only if none of the aisles in $G - L_j$ contains any item to be picked.

Corollary 4.10. If T_j^1 is a minimal length L_j PTS in i . equivalence class, and its minimal length completion is C_j^i , then the minimal length tour subgraph of G is the shortest $T_j^i \cup C_j^i$ where $i = (1, 2, \dots, 7)$.

Note that in L_n^+ the minimal length completion of PTS's in equivalence classes $(0, E, 1C)$, $(E, 0, 1C)$, $(E, E, 1C)$, $(0, 0, 1C)$ is an empty graph, namely this PTS's are tour subgraphs. In this case the remaining equivalence classes do not have any completion, so it is enough to examine these equivalence classes to find the optimal tour subgraph.

In the following, a minimum length tour subgraph is constructed. To do this, take $j = 1, 2, \dots, n$ aisles, and if we find a minimum length L_j^- PTS for all equivalence classes, then we can create an L_j^+ PTS. From this, we can construct an L_{j+1}^- PTS, and so on. Finally we determine the minimum length L_n^+ PTS for all equivalence classes, and the minimum length tour subgraph will be the minimum length L_n^+ PTS from equivalence classes $(0, E, 1C)$, $(E, 0, 1C)$, $(E, E, 1C)$, $(0, 0, 1C)$.

Let us consider aisle j . Because between any two vertices at most two edges will be needed, the edges correspond to aisle j can be placed in a tour subgraph in six ways, which is shown in Figure 3.

It can be shown that if the pick aisles and cross aisles have the same length, then the cases (v) and (vi) are not to be considered, because if an aisle does not contain any item to

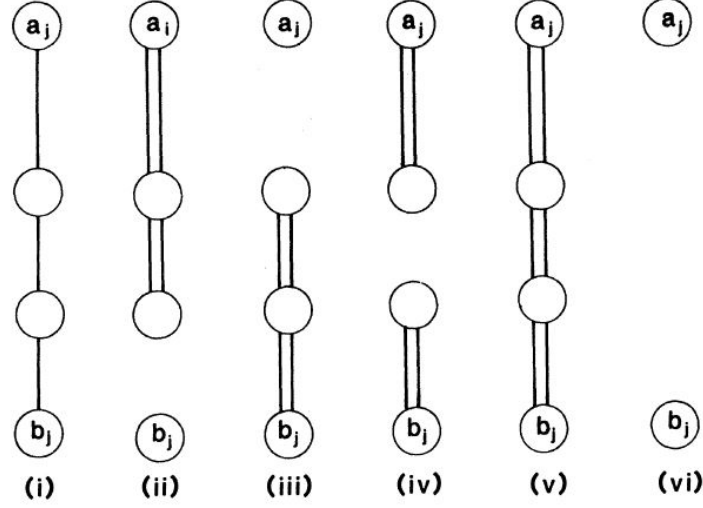


Figure 3: Possibilities of edges correspond to an aisle used by a tour subgraph from [13]

be picked, we can omit it. Case (vi) is possible only if there is no item to be picked on the aisle. The fourth case can be of several types depending on where we place the "hole". It is enough to take only the minimal case, namely if the hole is placed between two vertices where these two vertices are as far as possible to each other.

If we add a case depicted In Figure 3 to a partial tour subgraph, which belongs to a L_j^- equivalence class, then we get a case which belongs to equivalence class L_j^+ . So if we determine the minimal length L_j^- PTS for every classes, we can determine the minimal length L_j^+ PTS for every classes, too, which may result in a subgraph belonging to another equivalence class. The minimal length L_{j+1}^- PTS can be determined by adding some edges belonging to the corresponding cross aisles are added to the minimal length L_j^+ PTS. The options for adding these edges are illustrated in Figure 4.

If we fit a partial tour subgraph one of the five cases, it may happen that the PTS will belong to an other equivalence class. But this can be determined in advance and we can record it in a data table. So if we know the minimal length L_j^+ PTS's for every equivalence classes, then we can construct the minimal length L_{j+1}^- PTS's, so that we choose an option from our data table which results the shortest PTS for all equivalence classes.

As described above, we can construct for all equivalence classes the minimal length L_j^- PTS's, and then L_j^+ PTS's in cases $j = 2, 3, \dots, n$. In case $j = 1$, the six ways of layouts of edges in the aisles correspond to the six equivalence classes. The minimal length L_1^+ PTS is straight forward determine for each equivalence class. Finally, we can choose from the shortest L_n^+ PTS's the minimal length PTS, which corresponds to the minimal length

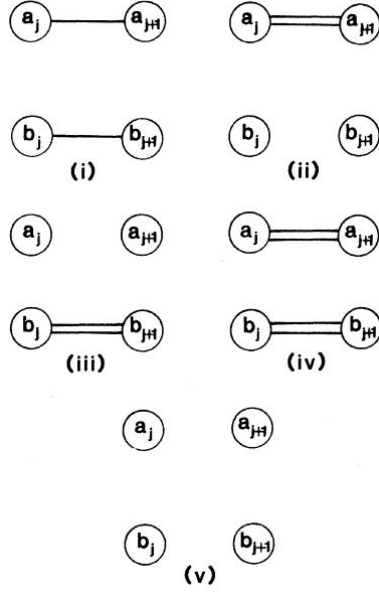


Figure 4: Possibilities of edges corresponding to a cross aisle used by a tour subgraph from [13]

order-picking tour subgraph.

It will be presented how to construct a minimal length order-picking tour in T from a minimal length tour subgraph. This can be done by the following algorithm:

Step 1. Start by visiting v_0

Step 2. Let the corresponding vertex be v^* .

Step 3. If there is an unused pair of edges incident to v^* , then use one and GO TO Step 2.

Step 4. If there is an unused single arc incident to v^* , then use it and GO TO Step 2.

Step 5. If there is a pair of edges, so that one of them is used and one is unused, then use it and GO TO Step 2.

Step 6. STOP.

Since all vertices of tour subgraph have even degree, the tour ends at v_0 so that all edges incident to v_0 are used. According to [13] it can be seen that the resulted tour includes all vertices of T .

4.2 Conclusions

In order to really apply the method described above, we have to adapt to the shaping and layout of the storage, and we have to modify the algorithm based on this. The process must be slightly modified if the depot vertex is located elsewhere. Recall that the method described above will not be changed essentially by changing the location of depot vertex. Depot vertex can be placed on any aisle. If this vertex is located on a cross aisle, for example between b_j and b_{j+1} , then only the cases should be considered in the construction of L_{j+1}^- PTS where b_j and b_{j+1} are used, and so fit them "artificially" into the solution. Alternatively, if we use a fictitious aisle by v_0 whose length is chosen to be so large that it will be not included into the tour.

The method is applicable even if the start and destination point are not the same. In this case we have to modify the concept of tour subgraph, so that we require that the degree of the start end destination vertices have to be odd (let v_0 and v_{m+1}). This change affects our algorithm that in aisles where v_0 and v_{m+1} can be found, we have to choose from other options instead of the cases seen in Figure 3. In this case v_0 and v_{m+1} can be found at the end of the aisle, so the edges of the pick-aisles have to be chosen from the cases illustrated in Figure 5, which correspond to the cases in Figure 3.

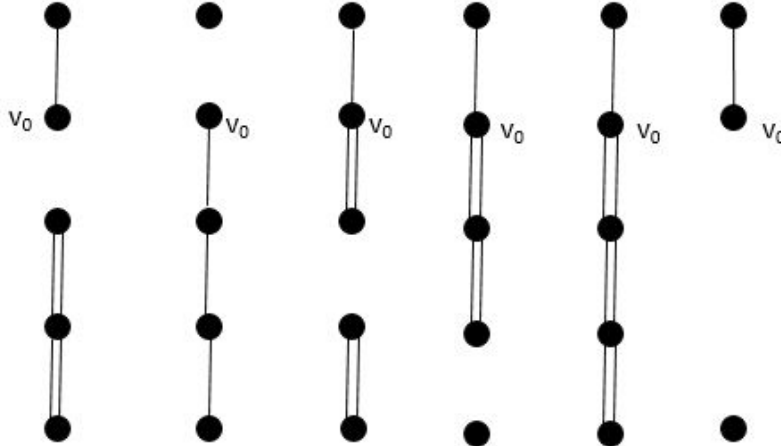


Figure 5: Possibilities of edges correspond to an aisle used by a tour subgraph

Warehouses are often designed in a way that not only are there two cross aisles perpendicularly to the pick aisles, but also there are more, in the middle of the pick aisles. In this case the method described above is still applicable, but the running time of the algorithm increases significantly, because the equivalence classes will include a lot more

elements [13]. Namely, if we have p cross aisle inside of the pick aisle, we have to examine $p + 2$ degree combination. In addition, the number of connectivity classes will increase, too, and we have to keep track of, which vertices are in the same component. The next chapter will deal with the management of cross aisles.

4.3 Heuristic solutions by using cross aisles

In the case where the warehouse has more cross aisles, the method described above will become too complicated and unmanageable, therefore more heuristic methods have been proposed to solve this, for example in [11] and [15]. Note that there is heuristic solution for the case without cross aisle in [16], which can replace the methods described above.

From now on, let us suppose that the depot is located in the lower left corner. Let the names of aisles be similar to the above, the vertical aisles are pick aisles (where picking happens), and the horizontal aisles are the *cross aisles* where picking does not happen. The area between two cross aisles is called *block*.

4.3.1 S-shape heuristic

In this section a method from [11] will be described. After starting at the depot, let us go to the first pick aisle which contains any item from the order. On this aisle go through the "upper" side of the last block, which contains item to be collected. Then go right until reaching an aisle of the current block containing item to be collected. If this is the only aisle in the block containing items to be picked, then pick them, and go to the "bottom" of the block. If there are more aisles in the block containing items to be picked, then go through the whole subaisle, and so go along aisles in "S" shape. If there are no more items to be picked in the block, go to the bottom of the block (if we are not there yet).

Then the next block will be the one we are at the "top" of. If there are items to be picked in this block, then check if the first (on the left) or the last (on the right) item to be picked is closer to the current position. Begin to collect items in "S" shape starting at the closest item similarly to that is, described above. If the block does not contain any item to be picked, then go through the closest pick aisle at the next block.

This process has to be continued until all blocks are travelled. Then, after visiting the last block, return to depot. The route obtained by the S-shape heuristic is shown in Figure 6.

4.3.2 Largest gap heuristic

Largest gap heuristic [11] begins similarly to S-shape heuristic. Namely, we go to the first pick aisle containing items to be picked, and go along it to the last block containing any items. During this method the vehicle goes through any cross aisle, and along a cross aisle enter each pick aisle, containing any item to be picked. Go along this pick aisle until reaching the largest *gap*, then return, and go out on the same side where the vehicle has entered. Gap means the distance between two items, or between one item and the end of the aisle. The last aisle, which contains item is travelled whole, and so can be reached the next block.

The next cross aisle is travelled so that pick aisles of both neighboring blocks are entered: one side first and the other side backwards, as can be seen in Figure 6. The whole last aisle containing items is travelled, and so will be reached the next cross aisle. This process is repeated until all blocks containing items are visited, then return to the depot.

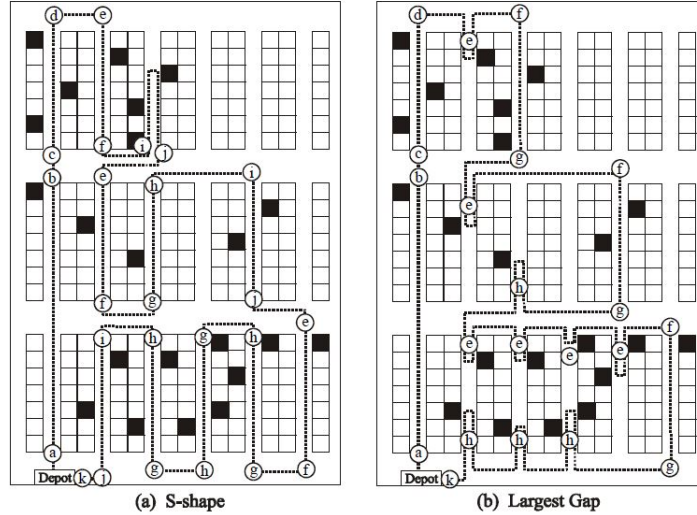


Figure 6: Paths getting by S-shape and Largest gap heuristic from [11]

4.3.3 Combined heuristic

The heuristic from [11] begins in the same way as before: go to the first pick aisle containing item, and then go through the last block containing items to be picked. The aisles are visited by one, and they will be entered only once. During this method, a small dynamic programming algorithm will be executed for all blocks.

Use the following notations: let k be the number of blocks, n is the number of aisles, a_{ij} is the upper end of j th aisle in block i , and b_{ij} is the lower end of the j th aisle in

block i , and finally denote the depot with d . It can be seen that $b_{ij} = a_{i+1,j}$ holds for $i = 1, 2, \dots, k-1$. Note that the distance between the middle of a cross aisle and the end of the pick aisle ending here is considered as the part of the pick aisle.

In case of a fixed block, l denote the first pick aisle containing items, and let r denote the last pick aisle, which contains item. Let L_j be the subpath visiting all pick locations from l to j . One can distinguish two types of such subpaths: let L_j^a be a subpath which ends on the back (or on the top) of the pick aisle j , and let L_j^b be the subpath which ends on the front (bottom) of the j th pick aisle. There are two transitions from aisle $j-1$ to aisle j : behind (above) of the block (denoted by t_a) or in front of (under) the block (denoted by t_b).

The picking of the items can be performed in four ways:

- Go through the subaisle to the next cross aisle (t_1)
- Do not enter in the aisle at all (t_2)
- Enter the aisle from the front of the block, and leave it in the same place (t_3)
- Enter the aisle from the back of the block, and leave it in the same place (t_4)

Obviously t_2 is possible only if the current subaisle does not contain any item to be picked. Denote $L_j + t_w$ the subpath which is constructed from L_j extending it with t_w where $w = 1, \dots, 4, a, b$. Finally denote the travelling time by c .

Using these notations, the algorithm can be described as follows:

First step. The current block is block i . If block i is the farthest block containing any item, then start with the following subpaths: L_l^a is a subpath which starts at point b_{il} , ends at point a_{il} and contains transition t_1 . L_j^b is the subpath which starts at point b_{il} and ends here, too (this contains transition t_3). If the block i is not the farthest block containing at least one item, then the starting subpaths are the following. L_l^a starts at a_{il} and ends here (this contains transition t_4). L_l^b starts at point a_{il} , ends at b_{il} and contains t_1 .

Second step. For each consecutive aisle j , determine the following values ($l < j < r$). If j contains any items to be picked:

$$L_j^a = \begin{cases} L_{j-1}^a + t_a + t_4, & \text{if } c(L_{j-1}^a + t_a + t_4) < c(L_{j-1}^b + t_b + t_1) \\ L_{j-1}^b + t_b + t_1, & \text{otherwise.} \end{cases}$$

$$L_j^b = \begin{cases} L_{j-1}^b + t_b + t_3, & \text{if } c(L_{j-1}^b + t_b + t_3) < c(L_{j-1}^a + t_a + t_1) \\ L_{j-1}^a + t_a + t_1, & \text{otherwise.} \end{cases}$$

If aisle j does not contain any item:

$$L_j^a = L_{j-1}^a + t_a L_j^b = L_{j-1}^b + t_b.$$

Third step. We find the following for the last aisle (r) in the block containing at least one item:

$$L_r^b = \begin{cases} L_{r-1}^b + t_b + t_3, & \text{if } c(L_{r-1}^b + t_b + t_3) < c(L_{r-1}^a + t_a + t_1) \\ L_{r-1}^a + t_a + t_1, & \text{otherwise,} \end{cases}$$

because, we want to stay at the bottom of the block after travelling the block, so that we can easily get through the next block. Then L_r^b is used for the order picking tour.

The dynamic programming algorithm described above deals only with a single block. Then we have to connect the algorithms running on different blocks, and so we can get the whole order-picking tour as a result. We do this as follows.

1. Determine the first pick aisle and the farthest block containing at least one item.
2. Go to the first pick aisle, and go to the bottom of the farthest block determined in the previous step. This block is called i_{\min} .
3. Let i be equal to i_{\min}
4. Check if block i contains any item to be picked and has not been collected in step 2. If there are no more items to be picked in block i , then go to the next block on the closest aisle, then GO TO 6. If block i contains any item to be picked, determine the first and the last aisles (called l and r) containing at least one item to be picked which has not been collected in step 2. Let j_{\min} be the aisle l or r that one, which is closer to the current position, and go there.
5. Apply the combined heuristic dynamic programming algorithm on block i . If $j_{\min} = l$, then add subpaths to the current route obtained from the dynamic programming algorithm. If $j_{\min} = r$, then reverse the subpaths obtained from dynamic programming algorithm, and then add them to the current route. This means that aisles will be visited not from left to right, but from right to left.

6. If the current block is the last block closest to the depot, then return to the depot. Otherwise $i = i + 1$, and GO TO Step 4.

Note that this combined heuristic can further be improved. For example, we can return to the depot after travelling the last block as soon as possible. Namely, if the vehicle travels in the last block from left to right, then it will finish at the right of the last block, and it has to travel to left to the depot, which is useless in terms of picking. Therefore the algorithm can be modified so that j_{\min} will be chosen in step 4 in a way, that the last block will be travelled from right to left, thus the vehicle can get out from the block on the left, closer to the depot.

Another improvement is to choose the route section is chosen more efficiently by travelling through the aisle containing the first item, in step 2. Namely, it is not necessary to travel through the whole distance of the first aisle, aisles can be changed depending on how the items to be picked are located. It can be done by the time of reaching the farthest block, the items on the first x aisles are collected, and they can be omitted by latter phases of the algorithm. The combined heuristic containing improvement methods will be referred as *combined*⁺, as it can be seen in [11].

4.3.4 Aisle-by-aisle

In the method presented in this section, the storage will be travelled by pick aisles instead of blocks. Namely, all pick aisles will be entered only once, and the cross aisle will chosen, which is best to go from a pick aisle to another. The principle of this method comes from [15]. Denote M the number of pick aisles, and let N denote the number of internal cross aisles. Let T be the length of the whole storage section of a pick aisle, so the length of storage section between two cross aisles is $L = \frac{T}{N+1}$. Let A be the width of a cross aisle, and K_m will be the number of items which has to be picked from each pick aisle, where $m = 1, 2, \dots, M$. Denote $X_m(t)$ the location of item t in pick aisle m . ($0 < X_m(t) < T$, it increases from top to bottom, and $t = 1, 2, \dots, K_m$.) Denote X_m^- the item which can be found in pick aisle m on the lowest position, and denote X_m^+ the item which is at the top of the aisle. These are not defined if $K_m = 0$. So $X_m^- = \max_t \{X_m(t)\}$, and $X_m^+ = \min_t \{X_m(t)\}$. Let $C_m(i, j)$ be the whole vertical travel distance, which is travelled if pick aisle m is entered on cross aisle i , all items are picked in this aisle, and then leave the aisle on cross aisle j . $B1_m(i, j)$ is the length of the route which is travelled if pick aisle m is entered on cross aisle i , and go to the item which can be found at the top. Similarly, let $B2_m(i, j)$ be the length of route which must be stepped back to the item that can be

found on the lowest position, before leaving on the cross aisle j . Namely:

$$B1_m(i, j) = \begin{cases} 0, & \text{if } K_m = 0 \\ 0, & \text{if } X_m^+ \geq \min(iL, jL) \\ 2 \left(\min(iL, jL) - X_m^+ + A(\frac{1}{2} + \lfloor \frac{\min(iL, jL) - X_m^+}{L} \rfloor) \right), & \text{if } X_m^+ < \min(iL, jL), \end{cases}$$

and

$$B2_m(i, j) = \begin{cases} 0, & \text{if } K_m = 0 \\ 0, & \text{if } X_m^- < \max(iL, jL) \\ 2 \left(\max(iL, jL) - X_m^- + A(\frac{1}{2} + \lfloor \frac{X_m^- - \max(iL, jL)}{L} \rfloor) \right), & \text{if } X_m^- \geq \max(iL, jL). \end{cases}$$

Denote the route $f_m(i)$ which has the minimal length, needed to pick items on aisles $1, 2, \dots, m-1, m$, if the pick aisle m is entered on cross aisle i . Namely:

$$f_m(i) = \min_j \{C_m(i, j) + f_{m-1}(j)\},$$

where $f_1(i) = C_1(i, N+1)$. So to find the minimal length route, $f_M(N+1)$ has to be determined.

4.3.5 Conclusions

In [11], the authors gave a detailed numerical evaluation of heuristics described above. During comparison, several cases have been examined. They differ from each other in length and number of pick aisles and in the amount of items to be picked. It has been observed, that S-shape heuristic practically never gives the best solution. The intuitive reason for this is that, compared with combined heuristic, in S-shape heuristic the vehicle always goes through the subaisle, which is entered, but combined heuristic always makes a decision whether we should go through or return. It can also be observed that heuristic *combined*⁺ gives almost always the best solution (compared to other heuristics).

To see the difference of the length of route obtained by heuristics, the following tables are presented [11]. The first table in Figure 7 shows the case when there are two cross aisles, and the table presented in Figure 8 demonstrates the case of four cross aisles. The first three columns describe the warehouse layout. The remaining columns shows heuristics, and the length of route obtained by the corresponding algorithm. In the last column the optimum values are presented, too. These have been calculated by branch-and

#aisles	Length	#items	S-shape	Largest gap	Combined	Combined+	Aisle-by-aisle	Optimal
7	10	10	165.1	146.6	148.5	148.5	148.5	138.7
7	10	30	203.5	208.6	192.1	192.1	192.1	186.6
15	10	10	266.2	227.3	235.2	235.2	235.2	219.6
15	10	30	391.3	357.5	356.7	356.7	356.7	337.5
7	30	10	353.1	295.1	304.7	304.7	304.7	269.6
7	30	30	452.0	451.7	418.8	418.8	418.8	398.3
15	30	10	517.6	401.0	427.2	427.2	427.2	377.3
15	30	30	833.3	715.6	732.7	732.7	732.7	665.5

Figure 7: Length of tours getting by heuristics using 2 cross aisles. Datas are from [11]

#aisles	Length	#items	S-shape	Largest gap	Combined	Combined+	Aisle-by-aisle	Optimal
7	10	10	152.6	164.6	145.4	136.2	153.7	131.5
7	10	30	250.1	273.9	236.1	224.7	227.0	198.6
15	10	10	245.0	287.2	235.3	214.5	229.4	201.4
15	10	30	431.1	484.5	402.5	373.3	369.1	311.6
7	30	10	256.5	250.7	235.1	219.3	268.4	211.1
7	30	30	438.2	425.7	397.4	379.3	422.8	342.9
15	30	10	361.0	379.1	332.5	305.3	358.6	290.9
15	30	30	688.2	665.4	605.6	567.6	642.8	495.9

Figure 8: Length of tours getting by heuristics using 4 cross aisles. Datas are from [11]

bound algorithm presented in [17]. Bold letters denote the best result obtained to the corresponding case.

In [11] test cases can be seen for more cross aisles, and it can be observed that for more than four cross aisles heuristic *combined*⁺ always gives the best solution.

But can these heuristics be really applied in practice? If the layout of pick and cross aisles are similar to the one described above, it is not difficult to customize heuristics for a actual storage. Namely, an alternative location of depot does not change the methods, is easily-to-handle, if start location is not the same as the destination. Notice, the processes described above, -except for aisle-by-aisle- start from top left corner of a rectangle, which limits the items to be picked. The direction of travel is determined so that the blocks are examined from top to bottom, so picking ends in a point close to depot. So, if depot is placed in another corner, the case is analogous to the above, because blocks have to be visited only in another sequence, for example from bottom to top. Visiting aisles can be performed from right to left, so aisle-by-aisle can be changed logically.

The answer is not clear if depot is not located in the corner, but somewhere at the edge of the warehouse. Recall that not the whole storage has to be visited during heuristics, but also only the rectangle which bounds the items to be picked. The route can start for

example from the corner of the rectangle which is the closest to the depot, and can end in the other corner on the edge which is the closest to the depot, or vice versa. It is useful to choose corner so that when the vehicle reaches the starting corner, it is able to pick some items on the way. Namely, in the example described above where depot can be found on the bottom left corner, it seems to be no difference between starting from the bottom right or from the top left corner. But starting from the bottom right corner, the vehicle travels through a cross aisle first, which is useless in terms of picking, and by starting the heuristic there are no picked items yet. Starting from the top left corner, picking can be started on the way to that corner, and it may happen that at the end of the algorithm the vehicle does not have to travel through the whole cross aisle by returning to the depot. (Namely, the "longer" route is preferred on aisle where picking can be performed, than on such one, where not.)

5 Vehicle Routing Problem

Algorithms and heuristics so far are able to solve problems where only one vehicle is used, and items to be picked are given in advance, which total weight does not exceed the vehicle capacity. These were essentially special cases of TSP. In this section the problem will be further generalized, and it will be examined, which methods should be used, if more vehicles can be used to collect items where the total weight delivered by one vehicle is limited. Vehicle Routing Problems deal with such questions, for the solution of which many algorithmic methods have been developed. There is an extensive collection of these in [12].

5.1 What is VRP?

Vehicle Routing Problems (VRP) deal with the question how to serve the demands of each customer by using a certain number of vehicles so that, the total length of routes traveling by vehicles should be minimal. Namely, given one or more depot stations, which can be used as a start or destination point of vehicles. Given the customers, whose exact location is known, and it is known what quantity of each product they need. In addition, a certain number of vehicles is available, which can be used for serving the demands of customers. All vehicles have a capacity value, which gives what quantity fits on the vehicle from the product to be served.

Many types of VRP can be distinguished. Most of the time these differ in that how

many depots exist, whether it can be distinguished by points of departure and points of arrival, how many types of vehicles can be used according to their capacity, and whether a fixed number of vehicles is available, or this number can be changed. It means further VRP types, if costumers can be visited only in a given time interval (called time windows), or if during the way vehicles have to deal not only with serving, but also with collecting products, too. (See more in [12].) The simplest form of VRP is called capacitated VRP (CVRP), where there is only one central depot, all vehicles' capacity are the same, and the demands of costumers are given in advance.

Since the solution of VRP is closely related to TSP, the problem is NP-complete, which means that there is no polynomial running time algorithm known for solving it. So there have been many process and heuristic developed to get an approximation of the optimal solution in relative short time.

In general case, VRP is handled so that, depot and costumers are represent with a complete graph where vertices correspond to costumers and depot, and the weight of edges means the distance of two points connected by it. In case of CVRP, the goal is to find K circle (where K means the number of vehicles) in graph so that, all circles have to contain the depot, all costumers are included in exactly one circle, all circles must have the property that the total demand of costumers visiting by them does not exceed the vehicle capacity, and the total length of circles has to be minimal.

5.2 VRP in a warehouse

During solving a warehouse management problem, it seems to be a good idea to consider the problem as a VRP task. Since in the current problem there are more available vehicles, with which locations are to be visited, and products are to be delivered, and these have to be done so that, the total travel time has to be minimal. The difference is that, products have to be not served, but also they have to be collected. But this is irrelevant for the solution, because the amount of products to be served by a single vehicle is limited in VRP, and the same limit can be used, if we perform picking instead of delivery.

So the map in VRP (namely the complete graph) is mapped now to the floor area of the storage. Depots correspond to the places in the warehouse where forklifts are stored, and where the picked products have to be delivered. Costumers in VRP correspond to item locations in the warehouse, from which products have to be collected with vehicles. The amount of products to be picked is known for all item location, which corresponds to costumer demands in VRP. Forklifts have capacity too, so the total weight of products

picked from item locations, which are visited by a single vehicle can not exceed the vehicle capacity.

What other differences can be encountered when solving a warehouse management problem? Obviously, we want to take advantage of the facility coming from the layout of warehouse, which is described in section 4. In addition, parameters given in section 4 have to be taken into account. Since a more complex problem has to be solved, compared with CVRP. Weight of items, and the capacity of vehicles, cannot be characterized by a single value, because a product has width, height, depth and weight, too. All four of them the vehicle capacity is given, namely multiple capacity constraints have to be determined during picking. The vehicles used are not the same, they can differ in their capacity. Vehicles start from different stations, which depends on their type, and they return not necessarily here, so the destination places can depend on vehicle's types too.

5.3 Examples for solving VRP in a warehouse

To solve VRP type problems, plenty of methods have been already developed. These include heuristic solutions, and processes, which start by describing the problem as an IP model, and try to solve this. In the following sections, some examples will be presented how to apply these methods, if warehouse management problems are to be solved.

5.3.1 Branch and Price algorithm

In this section, a solution method will be described, which was inspired by the algorithm from [18]. In the following an overall describing of the algorithm from [18] will be presented, then this will be transformed into such a method, which can be applied for the solution of the problem, which is described in section 2.

The article [18] gives an efficient solution method for Vehicle Routing with Demand Allocation Problem (VRDAP). In this problem there is only one depot, and all vehicle capacities have the same value (Q). The demand d_k is given for all costumers $k \in K$, where K denotes the set of costumers. Beside that the set of delivery sites S is given, too. The task is to transport products to delivery sites using the vehicles, and to assign costumers to delivery sites, in which they can receive their products in amount d_k , which they need. This has to be done that the total weight of products delivered by one vehicle should not exceed the vehicle capacity Q , one delivery site has to be visited at most by one vehicle, and to all costumers k should be assigned one delivery site s , in which they can receive the whole amount of the required products. The objective is to minimize the total weight

of routes travelled by vehicles and by costumers.

The article [18] describes the above problem as a set partitioning model, which is tried to solve by the so-called Branch and Price algorithm. During the process, the optimum of LP relaxation of the original problem will be found by using column generation method. The main steps of the algorithm are the following:

1. An initial feasible solution is constructed, which gives an upper bound to the optimal solution.
2. Solve the LP relaxation of the original problem with column generation, which gives a lower bound to the optimal solution.
3. If the upper bound is equal to the lower bound, then STOP.
4. If the upper bound is not equal to the lower bound, branching will be performed on the fractional values.

To describe the problem as a set partitioning model, use the following notations. Let V be the maximal number of vehicles, which can be used, and let $N = S \cup \{0\}$ be, where 0 denotes the depot. The edges corresponding to delivery sites are $E = \{(i, j) : i, j \in N, i \neq j\}$, and their cost is c_{ij} , and the edges corresponding to costumers are $E' = \{(k, j) : k \in K, j \in S\}$, and their cost is f_{ij} . Let H be the set of routes (or columns in IP model), which are feasible tours, namely the total demand of costumers, which are assigned to the delivery sites visiting by this tour does not exceed the vehicle capacity. P_h denotes the 0-1 vector, which gives the costumers assigned to $h \in H$, and let Q_h be such a 0-1 vector, which gives the delivery sites assigned to the tour $h \in H$. Let ρ_h be the cost of the route travelled by all costumers and vehicles in tour h . The variable corresponding to tour h will be denoted as z_h whose value is 1, if tour h is chosen in the solution, and 0 otherwise. So the problem can be described as follows:

$$\sum_{h \in H} P_k^h z_h = 1 \quad \forall k = 1, \dots, |K| \quad (1)$$

$$\sum_{h \in H} Q_i^h z_h \leq 1 \quad \forall i = 1, \dots, |S| \quad (2)$$

$$\sum_{h \in H} z_h = |V| \quad (3)$$

$$z \in \{0; 1\} \quad (4)$$

$$\min \sum_{h \in H} \rho_h z_h \quad (5)$$

Now the second step of the process will be explained in detail, which is the solving of LP problem by using column generation. To generate columns, the so-called Pricing Subproblem will be invoked, by which routes can be constructed, whose reduced cost is minimal. If one of these routes has negative cost, then that column will be inserted in LP, which correspond to this route. If such a route cannot be found, then terminate with the optimum solution.

The essence of the Pricing subproblem is that the routes to be constructed are not treated together in a common IP model by many constraints corresponding to them, but they will be determined one by one. Namely, an IP problem will be created whose constraints apply to which delivery sites will be visited in a certain route, and which costumers will be served by this. This will be performed so that not only the total weight of used edges are to be minimized by the objective function, but the number of served costumers are to be maximized, too. The corresponding variables can be added to the objective function with negative notation.

The Pricing Subproblem is solved by dynamic programming algorithm. In the process proposed by article [18], one exact dynamic programming algorithm and more heuristic methods will be invoked. The goal of this is the improvement in the running time. Since, if the column with negative reduced cost is obtained during the heuristic methods, then the running of the algorithm is substantially faster, than it would be solved by the exact method. Obviously, if none of the heuristic methods gives a column with negative cost, then it must be found by the exact algorithm.

The substance of the exact dynamic programming algorithm is that the routes constructed so far are labelled, and by the extension of routes the labels are dynamic updated. Before the algorithm a pre-processing method is performed whose goal is to except the delivery sites and costumers, which would not yield a negative value in the reduced cost.

A label consists of the following: $\mathcal{L} = (P_l, A_l, C_l, Q_l)$ where $P_l = (0, v_1, \dots, v_l)$ denotes the set of delivery sites, which are visited by the current tour, $A_l = (\emptyset, a_1, \dots, a_l)$ denotes the set of served costumers (a_i is the set of costumers assigned to v_i), C_l is the cost of the route constructed so far, and Q_l denotes the demand of costumers, which are served by this. An \mathcal{L} route is called feasible, if $Q_l \leq Q$, $\bigcap_{i=1}^l a_i = \emptyset$, and all delivery sites are visited in P_l by exactly once. Let σ_l and χ_l be the set of delivery sites and costumers, which was visited by \mathcal{L} . Let $\tilde{\sigma}_l$ and $\tilde{\chi}_l$ be the set of delivery sites and costumers, which is not included by \mathcal{L} , and cannot be inserted later, otherwise they would yield dominated labels. For all labels $\mathcal{L}^* \neq \mathcal{L}$ we say that \mathcal{L}^* dominates \mathcal{L} if the followings hold:

1. $C_{l^*} \leq C_l$, namely the cost of \mathcal{L} is not less than the cost of \mathcal{L}^* .
2. $Q_{l^*} \leq Q_l$, namely the products delivered on route \mathcal{L} take up not less space in the vehicle than on the route \mathcal{L}^* .
3. $\sigma_{l^*} \subseteq \sigma_l \cup \tilde{\sigma}_l$ és $\chi_{l^*} \subseteq \chi_l \cup \tilde{\chi}_l$, namely which delivery sites and costumers are visited by \mathcal{L}^* , these are visited by \mathcal{L} , too, or it does not visit them later.

New labels are constructed by the existing labels by extending them. This will be performed so that new delivery sites are added to the label of \mathcal{L} , and a set of costumers will be added to them. The assignment of costumers to a delivery site v_l is performed so that the subsets of costumers will be examined, which are not served so far, and whose total demand is not too large. (Namely, it does not exceed the value $Q - Q_l$.) All assignments of a subset to v_l yield a new label. In order to increase efficiency of the examine of subsets, the set of possible costumers will be decreased by the removing of costumers, whose assignment to v_l would yield a dominated label. To do this, many methods can be applied, which are now not discussed. (For details see [18].)

Returning to the original problem, it can be observed that it is a special case of the VRDAP. So the solution idea described above can be utilized in the current problem, which is described in 2. This can be done as follows. The locations of items, which are to be collected, correspond to the delivery sites included in VRDAP. After this, the problem can be seen as the assignment of costumers to delivery sites would already be performed, so this part of the problem has not to be considered, only the appropriate amount of products should be delivered, or more precisely collected. So the modelling of the problem by set partitioning model can be performed similarly as in the VRDAP. The difference is that the constraint (1) can be omitted, since the costumers assigned to delivery sites are now not considered. In addition, the fact cannot be ignored that in the problem examined now, the capacity of the vehicles are different, and they start and arrive in different places, depending on their type. Therefore instead of creating the set of feasible tours H , the sets H_1, H_2, \dots, H_t are to be constructed where H_i denotes the set of feasible tours according to i th vehicle, and t denotes the number of vehicle types. Namely, the constraint (2) changes as follows:

$$\sum_{i=1}^t \sum_{h_i \in H} Q_i^h z_h \leq 1.$$

Instead of (3) more constraints have to be considered, since only a certain number of vehicles is available from all vehicle types. If V_i denotes the amount, which can be used

from vehicle type i , then the following constraints have to be inserted instead of (3):

$$\sum_{h \in H_1} z_h \leq |V_1|, \dots, \sum_{h \in H_t} z_h \leq |V_t|.$$

The solving of Pricing Subproblem is to be changed similarly as above, too. In column generation method the column with the smallest reduced cost is to be found regardless of which feasible route corresponds to this according the vehicle type. A constraint corresponding to the feasibility of the routes is included in the conditions of Pricing Subproblem, which uses the vehicle capacity Q described in VRDAP. But in the current problem there are more vehicle types with different capacities. Namely, if the Pricing Subproblem will be described with Q_i capacities where Q_i denotes the capacity of vehicle i , then the result will be the route with the smallest reduced cost from the feasible routes according to vehicle i . But the current problem is to find from all feasible routes that one whose reduced cost is negative. So the Pricing Subproblem is to be solved starting with $i = 1$ for all vehicle types, until a negative cost route is found for one type. If there is no such an i , the algorithm terminates with the optimum.

5.3.2 Two-phase heuristics

For solving the CVRP problem, many heuristics have been developed, and a significant part of these ones are the two-phase heuristics. The basic idea of the two-phase heuristics is that they try to solve CVRP such that they part the problem into two subproblems. One of them is that the costumers will be distributed into groups so that costumers belonging to one group will be served by one vehicle. The other subproblem is to determine for all groups, in which sequence the vehicle should visit the costumers, namely a TSP problem will be solved for all vehicles.

In this thesis this type of heuristics is highlighted, because if this type of heuristics will be chosen for solving warehouse management problems, then the techniques described in section 4 and the heuristic methods given for VRP can be combined. It can be observed, if in the first phase of the heuristic the groups of item locations are determined, which have to be collected by a single vehicle, then the only remaining task is to solve TSP problem, for all vehicles, and that problem is exactly the same, which have been already discussed.

To find a feasible solution of the examined problem, a heuristic developed for a CVRP problem, the petal algorithm is analyzed, whose basic idea came from [19]. The main steps of the petal algorithm are the following:

1. The products are arranged according to their angle of rotation around the depot station.
2. In the ordered sequence of products the i th item will be assigned to the vehicle and then the next items after product i will be examined in turn, and we will find the latest of them, which can already fit on the vehicle. Let j be the first product, which cannot fit on the vehicle.
3. For all i , store the products between items i and k where $i \leq k < j$ (including i and k , too). These will be called feasible petals.
4. Feasible petals will be represented by a graph. All products correspond to a vertex in the graph. Vertices will be placed in a circle. Two vertices i and j are connected if and only if $i, \dots, j - 1$ products fit on the vehicle together. The cost of an edge is the length of the cheapest TSP tour between the corresponding products.
5. Designate a vertex k , which will be splitted: from k_s all arcs start, which have k as a source point, and in k_d all arcs arrive, which have k as target. Then find a minimum length route between k_s and k_d . So a petal solution can be obtained. Then the vertex k is to find, for which this petal solution is the cheapest.
6. It can be seen, if there is no arc between point i and $i + j$, then the optimal petal solution can be obtained from solving $i + j - 1$ minimal path problem. (See more in [19])

Obviously the process has to be changed so that it should give a feasible solution for the problem described in section 2. In the first steps of the petal algorithm, it can be decisive, which sequence of products will be chosen for running the algorithm. Probably the start and destination places of vehicles are not in the centre of the warehouse, but it can be conjectured if two products are placed near each other, then they will be picked by different vehicle with low possibility. So intuitively it can be thought that initially such a sequence should be chosen, which is obtained by Nearest Neighbor heuristic for the TSP problem. At this point a thought arises, whether a heuristic gives a better solution for TSP, then the sequence corresponding to them would yield a better solution for problem examined now. Therefore finding the best sequence can give a further research direction, which can be answered by testing the algorithm.

The next question is how to handle the case, if the start and destination points of the vehicle are different. This is not relevant by searching for feasible petals in petal algorithm,

only the calculation of the cost of the arcs corresponding to petals have to be changed, because the length of routes travelled by each vehicle changes. Therefore only the TSP heuristic used for products including by each petal has to be changed. For example, if we chose the Farthest Insertion heuristic for TSP, in which a circle tour is to be found (since the start and destination point are the same in CVRP), then this heuristic can be used now, if for a circle tour is searched where the start and destination points are placed next to each other. This can be reached so that the algorithm is initialized by a tour consisting only these two points, and by inserting the next point, a constraint regulates that the new point cannot be inserted between the start and destination point.

Finally, it will be discussed how to treat the case, if more types of vehicle are used. To do this, it has to be determined for all vehicle types, which petals are feasible. This can be represented in the graph so that for all feasible petals correspond an arc, whose color denotes, for which vehicle type it is a feasible petal. Similarly to the original petal algorithm, minimal length routes are to be found in the graph. But now it should be done so that the minimal route has to be chosen only from the feasible routes. We call a route feasible, if it uses from all colors no more as a upper bound given in advance. This upper bound means what amount is available from each vehicle types. Obviously such a solution is to be found, which uses no more vehicle from a type than it is available. To do this, a basic Dijkstra algorithm is run on the graph with the addition that if a minimal route to vertex v is found, then it has to be fixed how much arcs are used from each color in this route. If it has used from a color (for example red) a maximal amount, then the red arcs will be removed from the graph, which are incident to v and which are different from arcs on which vertex v is reached. So it can be seen that a minimal route leading to a given point is feasible, if it exists.

But why is it a minimal route between the feasible routes? Obviously, if the route leading to a given vertex was feasible in the original graph, then the changed Dijkstra algorithm will find this route, too. By removing an arc uv , a feasible minimal route cannot be "made ruined", since all routes using arc uv goes through u . If the arc uv was removed, this was the reason for that all arcs are already used from the color of uv , if u was reached. In the minimal length routes using arc uv , the subpaths before u are also minimal, and all minimal length routes using uv use of all arcs with the same color as uv before reaching u . Therefore non of the minimal length routes using uv can be feasible. Finally, it follows from the correctness of Dijkstra algorithm that a route leading to a given point, which is obtained by the algorithm, is minimal in the truncated graph.

6 Batching algorithms

As described in Chapter 2, items to be picked are given in orders as the input of the problem to be solved. An order is a fixed list of items to be picked. The set of orders to be collected during a planning horizon is known in advance. During picking, orders need to be considered, which means that the items contained by orders cannot be handled together in bulk, and consider only this great set. Picking has to be performed so that, by handing down the picked items, bags determined by orders should be collected easily. Namely, if items fit on a single vehicle, it is preferred to collect them by a single vehicle.

Method given for VRP described above can be applicable, if items from one order do not fit on a single vehicle. In such case the input of the problem, namely the set of items to be picked, is the content of a actual order, which should be collected by more vehicles. In this section the case will be examined when contents of more orders can fit on a single vehicle. The goal is to group orders, and assign them to vehicles, namely to determine for each vehicle, which of the orders should be collected. Such a group of orders will be called as *batch*. Constructing of batches has to be done so, that the total length of routes traveling by vehicles has to be minimal.

In the next, suppose that all orders are small enough to fit on a single vehicle. Each order can be included by exactly one batch, and each batch can contain only so much order to the total content of which does not exceed the vehicle capacity. In the following some heuristic solutions will be presented, which can support the solution of batching problem.

6.1 First-Come, First-Served heuristic

The First-Come, First-Served heuristic (FCFS) from [20] is a naive batching algorithm. The implementation of this is easy, and it can be used for measure of the quality of other heuristics.

The heuristic is the following: sort the first n orders into a batch so, that the weight of batch is close enough to the upper limit of batches. Then take the next m order, and sort them into batch so, that the weight of batch is close enough to the upper limit of batches. This will be repeated, until all orders are in a batch.

6.2 2-Dimensional spacefilling curve

The basic idea of the heuristic described in [20] comes from the article [21]. [21] presents a heuristic process to solve TSP. The basis of the method is the creation of a continuous

mapping between the points of a unit circle, and the internal points of a unit square. This mapping will be called spacefilling curve. The internal points of the unit square will be visited in such sequence, in which the corresponding points are in the unit circle.

Let the point of the circle be clockwise: $C = \{\vartheta \mid 0 \leq \vartheta < 1\}$. And let the internal points of the unit square be: $S = \{(x; y) \mid 0 \leq x \leq 1; 0 \leq y \leq 1\}$. Let Ψ be a continuous mapping from C to S . This mapping comes from Peano and Hilbert from 1890, see more in [22]. It is claimed for Ψ that $\lim_{\vartheta \rightarrow 1} \Psi(\vartheta) = \Psi(0)$. So if ϑ goes from 0 to 1, $\Psi(\vartheta)$ makes a tour on points of S . Namely, if there are given N points in S , which have to be visited, this can be done in such sequence, in which they are in tour $\Psi(\vartheta)$. So they can be sorted based on the inverse of Ψ .

Ψ has to be chosen so that it must have the following properties:

(P1) The inverse of Ψ has to be easily calculated. Namely, if x and y have k bit binary representation, then ϑ , which satisfies $(x, y) = \Psi(\vartheta)$, and has to be calculated from $O(k)$ operations.

(P2) There is a concave function f on interval $[0, 1]$ that

- $f(0) = 0$,
- $f(\Delta) = f(1 - \Delta)$, and
- $\|\Psi(\vartheta) - \Psi(\vartheta')\| \leq f(|\vartheta - \vartheta'|)$,

where $\|\cdot\|$ denotes the metric on S that the tour should be minimized.

The objective is to solve the problem on C instead of S . Using the properties described above, it can be seen that the following statements hold (see more in [21]).

Theorem 6.1 (Triangular inequality). *In case of $\vartheta_1 \leq \vartheta_2 \leq \vartheta_3$*

$$f(\vartheta_2 - \vartheta_1) + f(\vartheta_3 - \vartheta_2) \geq f(\vartheta_3 - \vartheta_1).$$

Theorem 6.2 (Crossing elimination). *In case of $\vartheta_1 \leq \vartheta_2 \leq \vartheta_3 \leq \vartheta_4$*

1. $f(\vartheta_3 - \vartheta_1) + f(\vartheta_4 - \vartheta_2) \geq f(\vartheta_2 - \vartheta_1) + f(\vartheta_4 - \vartheta_3)$
2. $f(\vartheta_3 - \vartheta_1) + f(\vartheta_4 - \vartheta_2) \geq f(\vartheta_3 - \vartheta_2) + f(\vartheta_4 - \vartheta_1).$

So from this properties follows that the optimal tour on C according to metric f can be obtained, if the points are visited in a row from the smallest ϑ value to the largest.

The spacfilling curve will be constructed similarly to the Sierpinski curve in a recursive way. The square will be partitioned in four part, and all of them will be full with a curve, which will be oriented so that they form a circle, see in Figure 9. Intuitively it will be appropriate, because if a point has been visited, then their neighbours will be visited, too, before going to the next region.

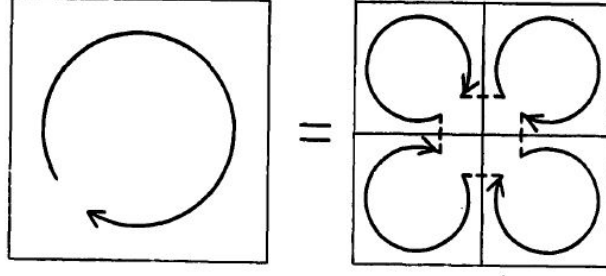


Figure 9: Construct spacfeilling curve from [21]

The corners of the square can be denoted as $q_0 = (0, 0)$; $q_1 = (0, 1)$; $q_2 = (1, 1)$; $q_3 = (1, 0)$. The curve crawls around the internal points of the square so that $\Psi(\vartheta)$ covers the subsquares, which include points q_i . The sequence of the points is $\Psi\left(\frac{i}{4}\right) = q_i$, $i = 0, 1, 2, 3$, if $|\vartheta - \frac{i}{4}| \leq \frac{1}{8}$, and in case $i = 0$, $|\vartheta - \frac{i}{4}| \geq \frac{7}{8}$. The curve has to be oriented in all squares so that it starts from the center of S , and returns here, too. So Psi can be determined as follows:

$$\Psi(\vartheta) = \frac{1}{2} \left[\Psi \left(\left\{ 4\vartheta + \frac{6-i}{4} \right\} \right) + q_i \right] \Big|_{i=\lfloor 4\vartheta + \frac{1}{2} \rfloor \bmod 4},$$

where $\{\cdot\}$ denotes the fractional part, and $\lfloor \cdot \rfloor$ denotes the floor. So the set of point on S are to be visited by using spacefilling curve, which can be done by evaluating of Ψ^{-1} .

The idea from [20] is that the spacefilling curve method summarized above can be used to create a batching algorithm. This can work intuitively, because points whom ϑ values are close enough to each other are located in the warehouse close to each other, too, so it can be used to determine a batching. Namely, products are proposed collecting together, which can be found relatively close to each other, so hopefully the vehicle has not to travel the whole warehouse.

Namely, the algorithm can be described as follows:

Step 1. Determine ϑ values for each item (this will be called as item ϑ values).

Step 2. For all orders determine locations whom have the smallest and larges ϑ value.

Step 3. For all item ϑ values of each order determine the smallest and the largest one. These two values are considered as coordinates of a point, and the ϑ value of this point is to be determined (this will be called as order ϑ values).

Step 4. Order the orders according there ϑ values in increasing sequence.

Step 5. According to the sequence obtained by Step 4, orders can be sorted in batches, as it was done in First-Come, First-Served heuristic.

6.3 4-Dimensional spacefilling curve

An improved version of method described above is expounded in [20], too, which is called 4-Dimensional spacefilling curve heuristic. This differ from the previous method that the spacefilling curve maps the points from four dimensional space to the unit circle (see more in [23]). This can be used, if the minimal and maximal x and y points are determined from the coordinates of items in all orders, denote them as x_{min} , y_{min} , x_{max} , y_{max} . In this case the rectangle determined by points (x_{min}, y_{min}) and (x_{max}, y_{max}) includes all items from the order. By using four dimensional mapping method calculate the ϑ values of points $(x_{min}, y_{min}, x_{max}, y_{max})$, this number will be the order ϑ value of the order. So the same process can be used as described in two dimensional case.

6.4 Sequential minimum distance heuristic

This heuristic, which comes from [20], based on the definition of the distance between two orders. Let (x_{im}, y_{im}) be the coordinates of the item i in order m . Furthermore:

$$L_{imn} = \min_j [(x_{im} - x_{jn})^2 + (y_{im} - y_{jn})^2]^{\frac{1}{2}}.$$

L_{imn} denotes the distance between the i . item of the order m , and the item from order n , which is closest to it. Define the distance between order m and order n as

$$d(m, n) = \sum_i L_{imn}.$$

Such a definition of the distance is not necessarily unique, and not necessarily symmetric. Namely, $d(m, n) \neq d(n, m)$ can occur.

To define the distance of orders it is another possible way, if there will be only measured that how much has to be travelled on cross aisles. The definition of distance in such a way

promotes creation of such batches, in which products from same pick aisles will be included in one tour. In this case define:

$$L_{imn} = \min_j |a_{im} - a_{jn}|.$$

The value of $d(m, n)$ can be defined in the same way described above.

After selecting, which distance definition will be applied, the following algorithm can be used:

- Step 1. Take the first order, it will be the "seed" of the first batch.
- Step 2. Calculate the distance of each order and the seed.
- Step 3. Orders, which are closest to the seed, will be inserted into the batch. So much orders can be inserted that the capacity of the batch should not be exceeded.
- Step 4. From the remaining orders choose a seed for the next batch, and GO TO Step 2. If all orders are grouped in batches, then STOP.

6.5 Order batching heuristic

The next process proposed by [24] uses the idea of the heuristic saving algorithm of Clarke and Wright from [25]. The saving algorithm is developed for VRP. The basic idea of this is that it is initialized by many short tours (namely all vehicles visit only one customer, and return to the depot), and it examines in all steps, which two short tours are worthy to connect, and connect them. So if two subtours can be connected (namely capacity constraints are not violated), then that one has to be connected from these pairs of tours, which results the shortest tour.

Let A_j be the set of pick aisles, which are to be visited in order to collect items from order j . Suppose that items are sorted in an order in such a sequence that the distance of them from the depot is monotone increasing. Suppose that items are collected in "S" shape, and there are only two cross aisles (front and back).

A pick aisle is to be travelled, if it contains at least one item, which is included by one order, whose content is to be picked in the current tour. Let L be the length of the pick aisle, so the length of the route, which has to be travelled to collect order j , is $L|A_j| + 2b_j$, where b_j denotes the distance of the farthest pick aisle and the depot.

If the content of orders i and j are to be picked, then treat the items mixed together, and sort them in increasing order starting from the depot. By merging orders i and j ,

the distance travelled has grown by the number of aisles to be visited compared collecting order i , and i and j together. This difference is to be minimized. If $A_i \subseteq A_j$, then there are no aisles, which have to be added to order j . The distance of two orders can be defined as follows (called minimum additional aisle distance):

$$s_{ij} = \min \{ |(A_i \cup A_j) \setminus A_i|, |(A_i \cup A_j) \setminus A_j| \}.$$

Another way to define the distance is the principle of "centre-of-gravity", which comes from Golden 1987 [26]. Define the gravity centre of order i as follows:

$$g_i = \sum_{k \in I} \frac{a_k}{I},$$

where I denotes the list of items contained by order i , and a_k denotes the index of pick aisle, which contains item k . The distance between orders i and j can be defined as follows:

$$s_{ij} = g_i - g_j.$$

Each distance definition has that property if the items of two orders are scattered, then the distance of them is greater, and it is smaller, if items of the orders can be found close to each other.

The heuristic algorithm can be described as follows:

- Step 1. Initially each order is contained by exactly one tour, namely it is supposed that all orders are collected in different turn. B denotes the set of orders. Calculate each s_{ij} value according to one distance definition for all $i = 1, 2, \dots, |B| - 1$ and $j = i + 1, i + 2, \dots, |B|$. Then GO TO Step 3.
- Step 2. Evaluate $s_{ij'}$ according to one distance definition for all $i \in B - \{j'\}$ where j' denoted the large order obtained in Step 4.
- Step 3. Let \bar{i} and \bar{j} be the pair of orders, which fulfills the constraint $w_i + w_j \leq C$, and between pairs fulfilling this, the value of s_{ij} is minimal. Here w_i denotes the weight of the order i , and C denotes the capacity of the vehicle. If there are more minimum, the pair of orders has to be chosen as \bar{i} and \bar{j} , which is equal to the following maximum value:

$$\max \{ w_i + w_j \mid (i, j) \text{ is minimal between pairs, where } w_i + w_j \leq C \text{ holds} \}.$$

Step 4. If there is no such a \bar{i} and \bar{j} pair, then STOP. Otherwise connect \bar{i} and \bar{j} as a larger j' batch. Expand the set B with j' , and remove \bar{i} and \bar{j} from this. If $w_j = C$, then GO TO Step 3, namely j' tour is finished. Otherwise $w_j < C$, and in such a case j' is still a subtour, and GO TO Step 2.

Note that if in step 3 the minimum value is taken on more pairs, then the process described above is recommended, because if the order with smaller weight is added to the batch before the larger one, then the larger can not be added to some batch easily later. Namely, in the algorithm, if an order is classified to one batch, then it cannot be taken out later.

6.6 Conclusions

To compare the heuristics described above, Gibson and Sharp performed experiments, whom results can be found in [20]. This does not include the Order batching heuristic, namely it is developed later by Rosewein. To test the latter heuristic, experiments have been performed, too, whom results can be found in [24]. During these tests the Order batching heuristic have been compared essentially with the heuristics proposed by Gibson and Sharp.

It is clear that by comparing batching heuristics, more parameters play a role, not only the heuristic, which have been chosen. Since the goal is to create such batches that the total length of travelled tour needed to collect all items is minimal, it is necessary to treat TSP heuristics with batching heuristics together during experiments. In fact, according to [20] a lot more parameters play a role in what results can be obtained. In this article Gibson and Sharp took into account the following parameters:

1. Batching heuristic
2. Travel metric: it plays a role in solution of TSP. During experiments more metrics have been applied to measure the distance between two items:
 - Euclidean distance: $d_E = [(x_1 - x_2)^2 + (y_1 - y_2)^2]^{\frac{1}{2}}$ (it is rarely used in warehouses).
 - Rectilinear distance: $d_R = |x_1 - x_2| + |y_1 - y_2|$.
 - Chebyshev distance: $d_C = \max \{|x_1 - x_2|, |y_1 - y_2|\}$.

3. Storage region representation: there are two different cases. The items to be picked are in a continuous way located anywhere, which yield an infinite number of possible item locations, or the items to be picked can be placed only on finite number of item locations, which are known in advance.
4. Item location assignment: it is the fixing of information, with what probability can be found items to be picked on the item locations. This probability distribution can be uniform or arbitrary skewed.
5. Fixed or variable number of items on one order: in case of fixed number of items all orders include the same number of them. In case of variable number of items it was assumed during experiments that items follow positive geometric distribution.
6. Number of orders
7. Number of replications to minimize random bias.

During experiments it turned out that FCFS heuristic does not result essential improvement in length of tours. Using aisle travel metrics and uniform item location assignment, it performs the worst solutions. In case of skewed distribution, which is given by [20], significant improvements can be achieved (according to experience 15-19%). But the application of Spacefilling curve and Sequential minimum distance heuristics resulted a more remarkable improvement in case of skewed distribution. (This can be result even 27-44% improvement in case of skewed distribution, see more in [20].)

In [24] the comparison of Order batching heuristic with heuristics of Gibson and Sharp has been performed. Here it was assumed after constructing batching that the aisles are in "S" shape travelled, so the comparing of obtained tours can be performed by the number of aisles, which has been travelled. The length of routes travelled on cross aisles was separately observed, namely it is the distance between the farthest visited aisle, and back to the depot, and this value was kept count for all obtained tour, which supported the comparison.

It was observed, what distance metric should be used to get a better performance of the Order batching heuristic. According to experiments it turned out that more shorter tours can be obtained by using minimum additional aisle distance, in fact by using the principle centre of gravity Order batching heuristic gives no better solutions in practice than heuristics of Gibson and Sharp. The table in Figure 10 is trying to show the difference between Order batching heuristic and the heuristics from Gibson and Sharp, which can be

experienced. The table summarizes the average of the results, concerning to the obtained picking tours, the number of travelled pick aisles, the distance travelled on cross aisles, and the running times of the CPU.

n	Avg. # of Picking Tours			Avg. # of Aisles			Avg. Cross-Aisle Traversal Effort			Avg. CPU Time (MM:SS)		
	OB	GS	%Decrease	OB	GS	%Decrease	OB	GS	%Decrease	OB	GS	%Increase
100	5.4	6.4	15.6	12.2	18.2	33.0	21.6	27.4	21.2	0:05	0:05	–
200	10.0	12.6	20.6	20.4	33.0	38.2	31.4	44.6	29.6	0:11	0:08	37.5
300	14.8	17.2	14.0	30.0	44.0	31.8	41.6	56.4	26.2	0:28	0:16	75.0
400	18.6	21.4	13.1	38.2	56.4	32.3	49.0	68.4	28.4	0:38	0:30	26.7
500	21.2	24.6	13.8	41.4	63.0	34.3	52.6	74.8	29.7	1:41	0:56	80.4
600	24.6	27.8	11.5	45.0	67.4	33.2	56.8	80.2	29.2	3:30	1:33	125.8
700	27.6	30.8	10.4	48.4	72.4	33.1	60.0	85.2	29.6	5:38	2:25	133.1
800	32.0	35.6	10.1	55.2	79.6	30.7	67.2	91.6	26.6	8:17	3:36	130.1
900	38.3	40.8	8.3	62.2	90.2	31.0	75.4	103.2	26.9	11:37	5:02	130.8
1000	42.6	46.0	7.4	67.4	96.4	30.1	80.4	110.6	27.3	16:21	6:53	137.5

Figure 10: Comparison of Order batching (OB) and Gibson and Sharp heuristics (GS) from [24]

7 Replenishment methods

As it has already been mentioned in section 2, the problem to be solved consists not only of the carrying out of picking in the warehouse using the appropriate methods, but taking care of the storage *replenishment*. To do this, a reserve storage area is available, and a forward area where picking can be performed and, which has to be replenished from the reserve area.

Logically a replenishment procedure is appropriate, if during picking no product will be out of stock, namely no such case occurs that at the moment of performing picking, in which the vehicle has just reached the item location, where there are not enough products on the shelf, therefore the vehicle cannot pick the needed amount. So the precise harmonization of the picking and replenishment procedures are indispensable, and the appropriate timing of replenishment is also necessary. But even if the picking is precisely planned, it cannot be determined exactly, when each vehicle will reach a given item location, namely what is the moment of time, for which replenishment has to be performed. The reason of this is that the vehicles are leaded by peoples, which results unpredictability in speed of vehicles and in time of the departure, and the probability of mistakes cannot be neglected. In addition, applying the methods described above the case of congestion cannot be excepted. This is the case when more vehicles reach the same location in the same time, therefore picking cannot be performed in the planned time, because vehicles have to wait for each

other. Therefore the strategy of harmonisation picking and replenishment has to be chosen appropriately, depending on what are the business needs, and what kind of information is available.

If from business aspect it is required that stock out (namely 0-pick) can never occur, then it seems the best solution to perform picking and replenishment in waves. This means that time intervals (for example an hour long) will be determined so that in one hour only picking can be performed in forward area, and in the other hour only replenishment will be executed. So the picking procedures performed in an hour can be planned so that at most so many items should be picked in one wave, which can fit on a shelf. Obviously the dividing of picking and replenishment procedures in waves can significantly increase the total time of picking, since picking process must be again and again stopped in order to execute replenishment. So it is important to make effort to optimize the replenishment processes. In case of such problems the one objective can be to perform replenishment as soon as possible in order to minimize the length of time period breaking of the picking procedure. Another goal can be to increase efficiency of replenishment, so picking should be only rarely interrupted, namely such an amount should be delivered to shelves, which is enough to increase the length of picking period.

If because of business aspects using waves cannot be allowed, since picking process cannot be interrupted, it is necessary to execute picking and replenishment in the same time. But in this case the probability of 0-picks will be positive because of arguments described above. In this method the objective can be to perform replenishment so that the expected value of number of 0-picks should be minimal. For this, and for case of using waves, examples will be presented in the next sections.

7.1 Storage Replenishment Problem

In this section a method is proposed for solving the problem, when picking and replenishment are in waves performed. For the time of replenishment an upper limit can be determined, which has to be observed, and beside that the travelling costs should be minimal. The Storage Replenishment Problem (SRP) deals with two questions: when and how many should be replenished from each item, if it has to be available in the next wave, and what ways the vehicles should travel during replenishment.

The article [27] deals with the solution of SRP. It assumes that only one vehicle is used for replenishment, which does not have capacity, and the initial inventory levels in forward and reserved storage area and the amounts to be picked from each item in a given wave

are known in advance, except for the layout of the storage, the location, weight and size of items. The objective is to minimize the total travel time. SRP is an NP-hard problem, whose proof can be found in [27]. Because of NP-hardness the computational time can significantly increase, if the size of the problem become large, therefore it is recommended to search heuristic solutions.

The heuristic proposed by [27] is the A Priori Route-Based Heuristic. The first step of heuristic is the constructing of a tour so that it can be seen in [13], and was presented in section 4, or it can be obtained by heuristic way based on [16]. The depot vertex corresponds to the place in the warehouse where the replenishment starts, and where the vehicle arrives after replenishment (namely the place where forward area can be reached from reserved area). After constructing the a priori route, fix the sequence of the items to be picked according to this. For all items i , the sets α_i and β_i can be defined where α_i contains items, which precede i on the a priori route, and β_i contains items, which follow i on this route.

In the following it will be tried to answer the question, which items should be replenished in each wave. Obviously, if it is fixed when and what should be replenished, then the only remaining task is to find the shortest path to load this.

Use the following notations:

- M : set of item locations
- $M' = M \cup v_0$ where v_0 denotes the depot vertex
- T : set of waves
- $T' = T \cup \{|T| + 1\}$
- $T^* = T \cup \{0\}$
- c_{ij} : the travel time needed to go from $i \in M'$ to $j \in M'$. It contains the time of replenishment, too.
- p_{it} : the amount of products $i \in M$, which arrives to reserve storage area in wave $t \in T$.
- b_{ikt} : the amount of product $i \in M$ to be delivered in wave $t \in T'$ to shelves, if the last replenishment was in wave $k \in T^*$ performed.
- r_{it} : the amount of product $i \in M$, which has to be picked in wave $t \in T$.

- π_{it} : the first wave for $i \in M$, for which the demand between π_{it} and t can be satisfied.
- μ_{ik} : the last wave for $i \in M$, whose demand can be satisfied from wave $k \in T$.
- C : time limit for all waves.
- I_{i1} : initial inventory level from product $i \in M$ in forward storage area.
- I'_{i1} : initial inventory level from product $i \in M$ in reserve storage area.

Use the following decision variables:

- z_{it} : binary variable, which gives whether a location $i \in M'$ will be visited in wave $t \in T$.
- \hat{y}_{it}^t : binary variable, which gives whether a location $j \in M'$ follows the location $i \in M'$ on the route for wave $t \in T$.
- I_{it} : the amount of product $i \in M$ stored in forward area in wave $t \in T'$.
- I'_{it} : the amount of product $i \in M$ stored in reserve storage area in wave $t \in T'$.
- w_{ikt} : binary variable, which gives whether product $i \in M$ is replenished in wave $t \in T'$, if the last replenishment was performed in wave k (where $\pi_{it} \leq k \leq t-1$).

If the amount of product i currently stored in forward area (order-up-to level) is denoted by U_i , then the values of b_{ikt} satisfies the following equations:

$$b_{i0t} = U_i - I_{i1} + \sum_{j=1}^{t-1} r_{ij}, \quad (6)$$

$$b_{ikt} = \sum_{j=k}^{t-1} r_{ij} \text{ for all } k \in T, \quad (7)$$

and the values of π_{it} and μ_{kt} can be described as follows:

$$\pi_{it} = \max_{0 \leq k \leq t-1} \{k : b_{ikt} \leq U_i\}, \quad (8)$$

$$\mu_{kt} = \max_{k+1 \leq t \leq |T|+1} \{t : b_{ikt} \leq U_i\} \quad (9)$$

For a fixed item i according to values w_{ikt} can be determined, in which waves the product was replenished. In fact, the possibilities when the item i can be replenished, can be represented by a graph. The vertices of a graph correspond to each wave, and the

vertex l is connected by an edge to vertex n (where $l < n$), if it is possible that item i was replenished in wave n , and before it, it was replenished last in wave l . So the edge connecting vertices l and n corresponds to variable w_{iln} . The fact, whether there is an edge between l and n , depends on the values of π_{in} and μ_{il} . The replenishment strategy of item i , namely in which waves i will be replenished, can be represented in the graph as a route between the 0th and last waves. The Figure 11 shows an example for a graph representation coming from [27] where the initial inventory level from item i is 20 units, the capacity of shelves is 45 units, and 20 units will be picked in every waves.

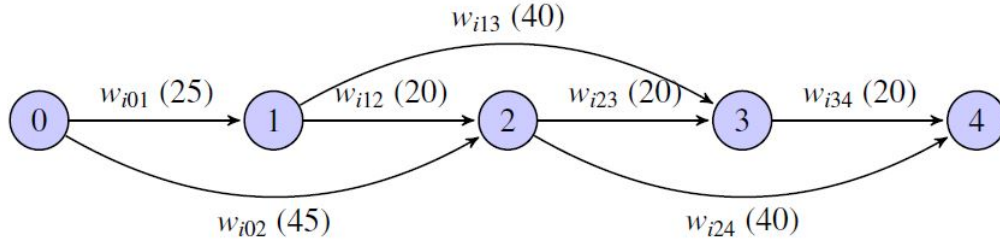


Figure 11: Example for graph representation of replenishment item i from [27]

To determine, which products should be replenished in each wave, the following problem should be solved.

$$\min \sum_{i \in M'} \sum_{j \in M'} \sum_{t \in T} c_{ij} \hat{y}_{ij}^t \quad (10)$$

$$\text{such that } I'_{i,t+1} = I'_{it} + p_{it} - \sum_{k=\pi_{it}}^{t-1} b_{ikt} w_{ikt} \quad \forall i \in M, t \in T \quad (11)$$

$$I'_{it} \geq \sum_{k=\pi_{it}}^{t-1} b_{ikt} w_{ikt} \quad \forall i \in M, t \in T \quad (12)$$

$$I_{i,t+1} = I_{it} + \sum_{k=\pi_{it}}^{t-1} b_{ikt} w_{ikt} - r_{it} \quad \forall i \in M, t \in T \quad (13)$$

$$\sum_{k=1}^{\mu_{i0}} w_{i0k} = 1 \quad \forall i \in M \quad (14)$$

$$\sum_{k=t+1}^{\mu_{it}} w_{itk} - \sum_{k=\pi_{it}}^{t-1} w_{ikt} = 0 \quad \forall i \in M, t \in T \quad (15)$$

$$\sum_{k=\pi_{it}}^{|T|} w_{i,k,|T|+1} = 1 \quad \forall i \in M \quad (16)$$

$$\sum_{k=\pi_{it}}^{t-1} w_{ikt} = z_{it} \quad \forall i \in M, t \in T \quad (17)$$

$$z_{it} \leq z_{0t} \quad \forall i \in M, t \in T \quad (18)$$

$$\sum_{j \in \alpha_i} \hat{y}_{ij}^t = z_{it} \quad \forall i \in M', t \in T \quad (19)$$

$$\sum_{j \in \beta_i} \hat{y}_{ji}^t = z_{it} \quad \forall i \in M', t \in T \quad (20)$$

$$\sum_{i \in M'} \sum_{j \in M'} c_{ij} \hat{y}_{ij}^t \leq C \quad \forall t \in T \quad (21)$$

$$w_{ikt} \in \{0; 1\} \quad \forall i \in M, t \in T', \pi_{it} \leq k \leq t-1 \quad (22)$$

$$\hat{y}_{ij}^t \in \{0; 1\} \quad \forall i \in M', j \in M', t \in T \quad (23)$$

$$z_{it} \in \{0; 1\} \quad \forall i \in M', t \in T \quad (24)$$

$$I_{it}, I'_{it} \geq 0 \quad \forall i \in M, t \in T \quad (25)$$

The value $c_{ij}\hat{y}_{ij}^t$ included in expression 10 gives the cost of the route between items i and j , provided that they follow each other in the solution. The objective is to minimize the total length of travelled route during replenishment, which is expressed by 10. The condition 11 fixed that the amount of item i stored in reserve storage area after wave $t+1$ can be obtained, if the arriving amount in reserved storage area in wave t is added to the amount stored after wave t , and the value is subtracted, which was delivered after wave t . The constraint 12 requires the availability of products in reserved storage area in all waves, which has to be delivered to forward storage area. Because of 13 so many products will be delivered into the forward storage area from item i for wave t , which are enough to perform picking in wave t . The constraints 14, 15 and 16 regulate that values w_{ikt} indeed give the loading strategies of the i th item, namely the variables w_{ikt} with value of 1 determine a route in graph representation described above, which starts from wave 0, and ends at the last wave. The values of variables z_{it} should be harmonized to this, which is required by 17 and 18. In the case, where the item i is visited in wave t , exactly one item appears before and after i in the sequence of visiting, this is described by constraints 19 and 20. Finally 21 corresponds to observe time limits of the wave, and constraints 22-25 regulate the possible values of variables.

The values of w_{ikt} can be determined by using the model described above, namely considering the constraints it can be decided, which edges of the graph will be used. After

determining, which items will be replenished in each wave, the only remaining task is to find the routes travelled by vehicles, which can be done according to some method from section 4.

According to [27] the process described above ran on more cases, which were different in the applied heuristic used for constructing a priori route, in the size of the problem, in the probabilities whether an item is included by an order (it can be uniform or skewed), and in the distribution of waves (namely how many days are included by the planning horizon, and in how many waves a day is divided). The test cases came from articles [28] and [29]. The running time of the algorithm stayed under 5 minutes in all cases on a computer used nowadays (for the details see [27]). The results of comparing the test cases are presented in Figure 12.

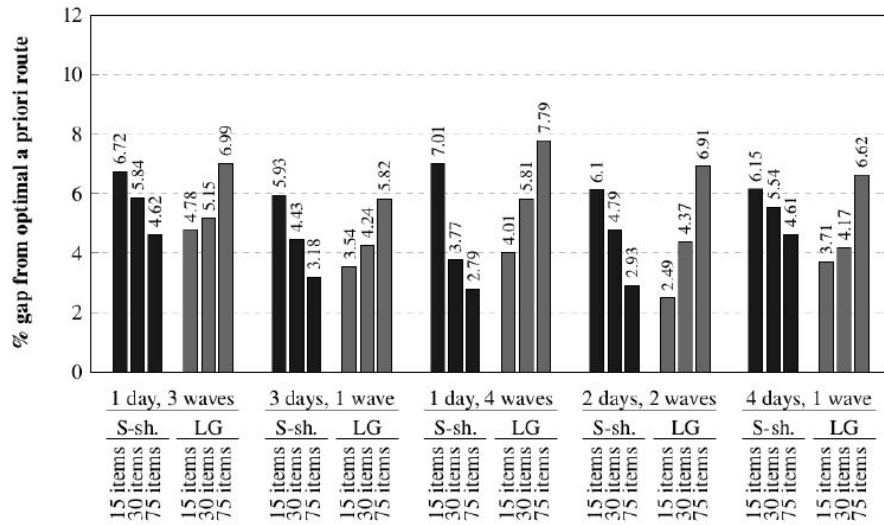


Figure 12: Experimental differences between instances from [27]

The examination, how this method can be applied for the problem described in section 2, will be discussed in section 8.

7.2 Prioritizing replenishment

In this section the case will be discussed, when the replenishment should be performed parallel with picking process, namely picking will be not stopped for the sake of replenishment. In [30] suggestions can be found, in which sequence the products should be replenished to minimize the number of 0-picks.

The article [30] assumes that only one "wave" is examined, namely the whole planning horizon. This can be for example one day. About planning horizon only one assumption is

made, namely the orders, which have to be picked in this period, and the items included by them are known, so it can be determined in advance, how many pieces from each item will be picked in that period. However it is not supposed that so many products fit on the shelves in the forward storage area, that picking can be performed. In fact it cannot be determined in advance, which product in exactly what time will be run out on the shelves.

For this type of problems the (S, s) rule can be applied, which means that a product will be replenished, if the amount on the shelves has decreased under value s , and in this case the shelf will be replenished to S pieces of items (S can be meant as the capacity of shelves). To do this, a warehouse management system is needed, which automatically gives actually reports about the inventory levels of products. So if it can be detected that the inventory level has decreased under s , then this item will be put on the top of the list to be replenished, and it will be replenished according to principle first-in-first-out. However applying this rule is not enough to avoid 0-picks (although by proper determining S and s can increase the chances), especially in cases, from which a greater amount will be picked often. Besides the fact, that a product will run out of stock sooner or later, it can be easily noted, because the currently inventory level of warehouse and the amounts to be picked in the current period are already known. So intuitively it can be seen as a good solution, if these items are "out of turn" replenished, and such replenishments have to be well prioritized.

In the following, as "replenishment order" will be called the replenishment of one item in one time. The products will be called "emergency products", from which it was determined that they will run out of stock in the planning horizon, these items have to be replenished urgently. Replenishment orders can be determined continuously, according to the current states, or they can be determined for all planning horizons periodically, namely they can be fixed for the examined time period, and cannot be changed later. In the continuous case the probability of 0-picks is less, however this case can be implemented harder, and it would often cause changes in replenishment processes, which yield more stress for the workers. So in the following, cases will be examined, in which priorities can be determined in advance.

In that case, if it can be known what amount will be replenished in each product in the next period, and it is known in what time each replenishment will be performed, then the problem in which priorities have to be assigned to emergency product is the same with that one where emergency products have to be assigned to each replenishment (since the most urgent product has to be replenished first, the second most urgent product has to

be replenished second time, etc.)

In the next the following notations are used:

- T : length of planning horizon,
- H : set of emergency products. The element are: $h \in \{1, 2, \dots, |H|\}$,
- N_h : the number of picking orders, which includes product h ,
- Π_h : the set containing all possible sequences, in any way the N_h orders containing h can be collected. This means $N_h!$ opportunities, and an element of the set is: π_h .
- $Q_h(k)$: The requested amount of product h , if the product is picked for the k th time ($1 \leq k \leq N_h$). Obviously this depends on value π_h .
- $S_h(k)$: The inventory level of product h , before it is picked for the k th time. Obviously this depends on value π_h .
- I : set of replenishment orders, and its elements: $i \in \{1, 2, \dots, |I|\}$.
- t_i : the time at which the replenishment order i is performed.
- $X_h(t_i)$: the number of 0-picks applied to product h , if product h is replenished at time t_i .
- $f_{hi} = E(X_h(t_i))$: the expected value of number of 0-picks applied to product h .
- $Y_h(t_i)$: The number of picks from product h before time t_i , including 0-picks, too.

In addition, the following decision variable is used:

$$z_{hi} = \begin{cases} 1, & \text{if emergency product } h \text{ is assigned to replenishment order } i \\ 0, & \text{otherwise} \end{cases}$$

In the following some proposals will be presented for prioritizing emergency products, which came from [30].

7.2.1 Stock-Needs Rule

During this process priorities will be assigned to products according to the following: the lower the inventory level of the product on the shelf is compared to the further demands, the larger the possibility of 0-picks is. So the value

$$\frac{\text{stock}}{\text{needs}} = \frac{S_h(1)}{\sum_{k=1}^{N_h} Q_h(k)}$$

is used for determining priorities of emergency products. The product, for which the value of the fraction is the smallest, gets the largest priority, the second smallest gets the second largest priority, etc. Priorities are determined for the planning horizon, and they cannot be changed later. Replenishment orders are ordered in batches, and replenishment will be performed so. After loading all emergency products, the replenishment according to (s, S) rule can be continue.

7.2.2 Order-Quantity-Based Rule

To apply this method, the following assumptions have to be made:

1. The length of one wave is known in advance.
2. The time at which picking orders are performed has uniform distribution between the beginning and the end of the planning horizon.
3. The moments are independent to each other, at which picking orders are performed, which contain a certain product.
4. The batching methods of replenishment orders are independent to the cost of the replenishment.
5. The moment t_i of the loading of the replenishment order i can be determined at the beginning of the planning horizon, and it is independent to the product, which is replenished.
6. The stocks of the warehouse are enough to satisfy the picking orders.
7. After loading a product, the inventory level in the forward storage area is enough for the remaining part in the planning horizon (namely, all products are replenished only once).

8. If the inventory level of the product on the shelves is too small to pick the requested amount, then none of this will be picked.

Between the assumptions there are more (for example 1, 3, ??), which cannot be provided in the reality, but they have to be assumed to create the model. But most of them cause no problem in a real case, for example for the length of the planning horizon, and the moments of the replenishment estimated values can be used. In case of 2 no picking probabilities are distinguished for all time units. This can be used, if we have only the information that picking will happen somewhere between the beginning and the end of the planning horizon. This assumption and ?? can be used in that case, if the length of the planning horizon is not too large. 4 has to be assumed, because we want to keep minimizing of the number of 0-picks as a main objective. If beside that the cost has to be minimized, too, then the process will be too complete, which we want to use.

The problem will be again solved so that instead of assigning priorities to the emergency products, the emergency products will be assigned to the replenishment phases defined in advance. This is equivalent to the original problem because of the assumptions described above, and it can be described as the following assignment problem:

$$\min \sum_{h \in H} \sum_{i \in I} z_{hi} f_{hi} \quad (26)$$

$$\sum_{h \in H} z_{hi} \leq 1 \quad \forall i \in I \quad (27)$$

$$\sum_{i \in I} z_{hi} = 1 \quad (28)$$

$$z_{hi} \in \{0, 1\} \quad \forall i \in I, \forall h \in H \quad (29)$$

If $|H| > |I|$ hold, then the constraint 27 can be replaced by the expression $\sum_{h \in H} z_{hi} = 1$, and instead of constraint 28, the expression $\sum_{i \in I} z_{hi} \leq 1$ can be used. The assignment problem described above is NP hard (see more: [31]), and it is hard to implement, if a warehouse management system is used. But the following can be observed:

1. In case of $|H| \leq |I|$, the emergency products will be assigned to the first H replenishment. So the other replenishment orders can be omitted. Let I^* be the new set of replenishment orders, which satisfies $|H| = |I^*|$. So in 27 the inequality can be replaced by an equality. So the problem turns into a classic Linear Sum Assignment Problem (LSAP).

2. If $|H| > |I|$, the problem can be modelled as LSAP, too, if $|H| - |I|$ further imaginary replenishments are added to the existing replenishment phases. Let I^{**} be the new set of replenishments. If this change is applied, then $|H| = |I^{**}|$, and the model can become again an LSAP. After determining the assignments, the imaginary replenishments have to be removed.

In the model described above, the values f_{hi} were used in the objective function, which is the expected number of 0-picks at product h . These values can be obtained using the assumptions described above as follows:

$$f_{hi} = E(X_h(t_i)) = \quad (30)$$

$$= \sum_{j=1}^{N_h} E(X_h(t_i) \mid Y_h(t_i) = j) \cdot P(Y_h(t_i) = j) = \quad (31)$$

$$= \sum_{j=1}^{N_h} E(X_h(t_i) \mid Y_h(t_i) = j) \binom{N_h}{j} \cdot \left(\frac{t_i}{T}\right)^j \cdot \left(\frac{T-t_i}{T}\right)^{N_h-j} = \quad (32)$$

$$= \sum_{j=1}^{N_h} \sum_{\pi_h \in \Pi_h} E(X_h(t_i) \mid Y_h(t_i) = j, \pi_h) \cdot P(\pi_h) \cdot \binom{N_h}{j} \cdot \left(\frac{t_i}{T}\right)^j \cdot \left(\frac{T-t_i}{T}\right)^{N_h-j} = \quad (33)$$

$$= \sum_{j=1}^{N_h} \sum_{\pi_h \in \Pi_h} E(X_h(t_i) \mid Y_h(t_i) = j, \pi_h) \cdot \frac{1}{N_h!} \cdot \binom{N_h}{j} \cdot \left(\frac{t_i}{T}\right)^j \cdot \left(\frac{T-t_i}{T}\right)^{N_h-j} = \quad (34)$$

$$= \sum_{j=1}^{N_h} \sum_{\pi_h \in \Pi_h} \sum_{k=1}^j \mathbb{1}_{S_h(k) < Q_h(k)} \cdot \frac{1}{N_h!} \cdot \binom{N_h}{j} \cdot \left(\frac{t_i}{T}\right)^j \cdot \left(\frac{T-t_i}{T}\right)^{N_h-j}. \quad (35)$$

Here, in 31 the possibility $P(Y_h(t_i) = j)$ can be counted as binomial distribution because of assumptions 1-3, and so, if N_h test are performed, then $\frac{t_i}{T}$ is the possibility of successful cases, and $\frac{T-t_i}{T}$ is the unsuccessful ones. In 33 the expected value is limited in that case, if the orders including by N_h are in sequence π_h collected, and it is multiplied by the probability, by which it can occur. Finally, for fixed $Y_h(t_i)$ and π_h values, the number of 0-picks can be determined, this is denoted by the indicator function $\mathbb{1}_{S_h(k) < Q_h(k)}$.

So it can be stated, by keeping assumptions described above, and after calculating values f_{hi} , that assignment problem described by 26-29 gives the optimal solution of the problem.

But it can be observed that the evaluating of values f_{ih} needs a larger calculation time. In the article [30] a proposal can be found how to save time in the calculation, but for this, it has to be assumed that for all orders, which contain product h , the same amount

has to be picked, which is estimated by the value

$$\bar{Q}_h = \frac{\sum_{k=1}^{N_h} Q_h(k)}{N_h}.$$

In this case, the counting time of values f_{ih} decreases, however the new assumption takes the problem further to the reality, so it can happen that in a real case, the results are less satisfying.

7.2.3 Conclusions

The [30] tried to compare the methods described above with each other by running tests. These were happening in an imaginary warehouse by using such orders, which satisfy the conditions described above. During experiments the (S, s) rule was also tested, and that case, too, in which the assumption holds that for all orders containing item h the same amount will be picked. (This case will be called as Order-Based Rule, and denoted by OBR.)

From the result it can be clearly detectable that (S, s) rule on its own does not prevent the 0-picks by emergency products in fact, since in this case the replenishment is not planned forward, and it can occur that an emergency product will be replenished later then a not emergency product. According to the results, the Stock-Needs rule (SNR) can provide significant improvements, but it does not give such a good solutions as the Order-Quantity-Based Rule (OQBR), and OBR methods. The reason for this can be that the SNR method is less able to make such decisions, which provide good solutions in long term.

The difference between these methods are well presented on table in Figure 13, which came from [30].

Replenishment policy	Mean # 0-picks per wave	Mean % diff. with the OQBR	Std. of % diff. with the OQBR	95% Conf. interval of mean % diff. with the OQBR
(s, S)	53.59	491.5	86.73	[486.2, 496.9]
SNR	12.56	35.5	14.24	[34.6, 36.4]
OBR	9.61	3.7	2.83	[3.5, 3.9]
OQBR	9.29	–	–	–

Figure 13: Results of comparison from [30]

8 Summary

In this section, it will be discussed, how to connect the solution methods given for subproblems described above, and how to use them in a solution of a greater problem. About the correlation of the subproblems was already talked by describing the methods, but now let see the problem from a higher level, and so develop a comprehensive view about the opportunities.

As it was described in sections 4 and 5, in solution of TSP and VRP can be a tight connection observed, so the solution methods giving for routing problems, can be effectively applied in two phases VRP heuristics, and so it can be taken advantage of the facility of the warehouse, instead of the solving original TSP and VRP problems. Obviously such a method can be chosen, too, where not all subproblems have to be solved, which was described above. For example, if the Branch and Price algorithm described in section 5 is to be used, it can be done without using some solution method of the routing problems described in section 4.

Similarly, it can be occurred that the business demands need to solve the problem by a VRP heuristic such that the using of batching algorithms are unnecessary. However this algorithms can support the replenishment processes, so in this case the batching subproblem should not to be omitted.

Suppose that the planning horizon is given, and the orders are known, whose items are to be picked in this planning horizon. The parameters of the problem are given as described in section 2. In addition, the replenishment processes should not to be omitted in order to perform the picking. By choosing the solution strategy, it has to be taken into account among others the size of the orders, the capacity of the shelves in the forward storage area, and the length of the planning horizon.

In the case, in which the size of the orders are typically small enough to fit more of them on one vehicle, then it is recommended to organize orders on vehicles using some batching heuristic, which was described in section 6. Since the vehicles will probably travel in more turns in the period of the planning horizon, then batches can be considered as a set of items, which will be picked by a single vehicle in one turn. If in addition the 0-picks have to be excepted, then such a method should be used for replenishment, in which loading happens by using waves. In this case the length of a wave has to be chosen so that the problem can be solved that during the picking period no more products have to be picked, than the amount, which can be fit on the shelves. The problem, in which batches are to be assigned to the waves, can be modelled as an IP problem. Since the picking has

to be performed during the planning horizon, the picking should be interrupted as rarely as possible by the loading periods, so let the objective of the problem be the minimizing of the number of waves.

Denotes I the set of items, B denotes the set of batches, and H will be the set of waves. Furthermore let the following notations be used: S_i denotes the amount, which can be stored on shelves from the item i , x_{kj} is a variable, which has the value one, if the batch k will be picked in wave j , and let b_{ik} be a parameter corresponding to the amount, which has to be picked in batch k from item i . In addition, the variable y_j has the value one, if in the wave a picking is performed. So the following IP model can be constructed:

$$\min \sum_{j=0}^n m_j y_j \quad (36)$$

$$\sum_{k=1}^n x_{kj} b_{ik} \leq S_i \quad \forall i \in I, j \in H \quad (37)$$

$$\sum_{k=0}^n x_{kj} - y_j \geq 0 \quad \forall j \in H \quad (38)$$

$$\sum_{k=0}^n x_{kj} - n y_j \leq 0 \quad \forall j \in H \quad (39)$$

where n is an upper estimation for the number of waves. The parameters m_j are constant, which have a large value for a larger value of j . This have to be used, because so the picking will be performed in earlier waves, and the later waves will remain "empty". So it can be determined, which batch should be picked in each wave, and so it will be clear, what amount has to be picked from item i in each wave. This can be an input data for the algorithm described in section 7.1.

If the orders have a larger size, and they typically cannot fit on a single vehicle, then the problem has to be thought in another way. It can be assumed that the collecting items is performed so that the content of one order can be distributed for more vehicles, but one vehicle can transport the content of at least one order. On its own by using VRP methods the problem can be solved, how to collect items including by one order. Obviously it can happen that not all vehicles are to be used for picking an order, and in this case the remaining vehicles can perform the picking of other orders. It is to be determined, which orders should be collected parallelly, and how they can be organized in waves in order to perform replenishment.

Suppose that it is known for all orders, how many vehicles are needed to collect them. To do this, such a VRP problem can be solved for all orders, in which the number of vehicles

is also minimized, or the problem can be treated as a Bin Packing Problem (BPP), too. If it is determined, how many vehicle will be pick each order, by solving another BPP, they can be assigned for the available vehicles. So all bins correspond to one turn. These turns can determine the waves for the replenishment problem, but in this case a new condition is needed, namely in one turn (or wave) no more product should be collected than the capacity of the shelves. This can be modelled by a BPP with multiple constraint, where one group of the conditions corresponds to the requirement that no more order should be assigned to one turn, as it can be served by the available vehicles, and the other group of the conditions corresponds to the demand that no more order should be assigned in one turn as it can be picked from the shelves.

Finally let see the case, in which the replenishment and picking should be performed parallely. Here, it can be observed that in section 7.2.2 relatively strict assumptions were determined in order to model the problem easily. Among these there are more assumption, which can be supposed in reality only in the case, if the planning horizon is small enough. (for example the distribution of picking times, and the assumption that all items should be replenished at most once in the period). So a mixed solution can be constructed, which means that picking will be distributed in waves (whose length is small enough), and the problem of parallel replenishment will be solved for all waves separately. In this case there are no replenishment periods, only the assignment of batches to waves have to be well determined. In this case it has not to be required that in a wave there should be no more items picked as it fits on the shelves. But it can be a good idea to chose the assignments so that the congestion should have less probability, which can decrease the probability of 0-picks, too. This can give a further research direction in this topic.

The solution proposals described in this thesis tried to give effective methods for a wider range of warehouse management problems, so these are suitable for not only to solve a certain business demand, but also we can get inspiration from this in many different cases. To do this, the mathematical models and algorithms have to be changed such that they will be suitable for solving more specific problems. Overall it can say that the warehouse management problems hide lots of research directions, as far as special business demands and improvement of algorithmic solutions and finding new ideas are concerned.

References

- [1] J.A. White, H.D. Kinney, Storage and Warehousing, Handbook of Industrial Engineering, 1982.
- [2] S. Nahmias, Production and Operation analysis, volume third ed., McGraw-Hill International Editions, 1997.
- [3] Y.A. Bozer, Optimizing Throughput Performance in Designing Order Picking Systems, Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1985.
- [4] Moncer A. Hariga, Peter L. Jackson, The warehouse scheduling problem: Formulation and algorithms, IIE Transactions 28 (1996) 115–127.
- [5] B. Rouwenhorst, B. Reuter, V. Stockrahm, G.J. van Houtum, R.J. Mantel, W.H.M. Zijm, Warehouse design and control: Framework and literature review, European Journal of Operational Research 122 (2000) 515–533.
- [6] S. Hackman, J. Rosenblatt, Allocating items to an automated storage and retrieval system, IIE Transactions 22 (1990) 7–14.
- [7] E.H. Frazelle, S.T. Hackman, U. Passy, L.K. Platzman, The forward–reserve problem, Optimization in Industry 2 (1994) 43–61.
- [8] Jeroen P. Van den Berg, Gunter P. Sharp, A.J.R.M. Gademann, Yves Pochet, Forward–reserve allocation in a warehouse with unit-load replenishments, European Journal of Operational Research 111 (1998) 98–113.
- [9] Strack, G.; Pochet, Y. 2010. An integrated model for warehouse and inventory planning, European Journal of Operational Research 204(1): 35–50.
- [10] Roodbergen, K.J. and De Koster, R. (2001), Routing order pickers in a warehouse with a middle aisle. European Journal of Operational Research 133(1), 32-43.
- [11] Roodbergen, K.J. and De Koster, R. (2001), Routing methods for warehouses with multiple cross aisles. International Journal of Production Research 39(9), 1865-1883.
- [12] Paolo Toth, Daniele Vigo. The Vehicle Routing Problem. Università degli Studi di Bologna, Bologna, Italy, 2002.

- [13] H. D. Ratliff and A. S. Rosenthal (1983). "Order picking in a rectangular warehouse: A solvable case of the Traveling Salesman Problem." *Operations Research* 31(3), 507–521.
- [14] Christofides, N. 1975. *Graph Theory: An Algorithmic Approach*. Academic Press, London.
- [15] T. S. Vaughan (1999) The effect of warehouse cross aisles on order picking efficiency, *International Journal of Production Research*, 37:4, 881-897, DOI: 10.1080/002075499191580
- [16] R. W. Hall (1993). "Distance approximations for routing manual pickers in a warehouse." *IIE Transactions* 25(4), 76–87.
- [17] Little, J. D. C., Murty, K. G., Sweeney, D. W., and Karel, C., 1963, An algorithm for the traveling salesman problem. *Operations Research*, 11, 972-989.
- [18] Mohammad Reihaneh, Ahmed Ghoniem, A Branch-and-Price Algorithm for a Vehicle Routing with Demand Allocation Problem, *European Journal of Operational Research* (2018), doi: 10.1016/j.ejor.2018.06.049
- [19] D. M. Ryan, C. Hjorring, and F. Glover. Extensions of the petal method for the vehicle routing problem. *Journal of Operational Research Society*, 44:289-296, 1993.
- [20] Gibson , D.R. and Sharp , G.P. (1992) Order batching procedures. *European Journal of Operational Research*, 58 (1), 57–67.
- [21] Bartholdi, J.J., and Platzman, L.K. (1982), "An $O(N \log N)$ planar travelling salesman heuristic based on spacefilling curves". *Operations Research Letters* 1/4, 121-125.
- [22] H. Abelson and A. disessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. MIT Press. Cambridse, MA (1982).
- [23] Bartholdi, J.J., and Platzman, L.K. (1988), "Heuristics based on spacefilling curves for combinatorial problems in euclidean space", *Management Science* 34/3, 291-305.
- [24] M. B. Rosewein (1996) A comparison of heuristics for the problem of batching orders for warehouse selection, *International Journal of Production Research*, 34:3, 657-664, DOI: 10.1080/00207549608904926

- [25] G. Clarke, J. V. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568-581, 1964.
- [26] Golden. B. L., Levy. L. and Vohra. R., 1987, The orienteering problem. *Naval Research Logistics*, 34, 307-318.
- [27] Sural, Haldun; Archetti, Claudia; and Celik, Melih, "The Storage Replenishment Problem in Rectangular Warehouses" (2016). 14th IMHRC Proceedings (Karlsruhe, Germany – 2016). 29.
- [28] O. Solyalı and H. Süral (2011). "A branch-and-cut algorithm using a strong formulation and an a priori tour-based heuristic for an inventory-routing problem." *Transportation Science* 45(3), 335–345.
- [29] C. Archetti, L. Bertazzi, G. Laporte, and M. G. Speranza (2007). "A branch-and-cut algorithm for a vendor-managed inventory-routing problem." *Transportation Science* 41(3), 382–391.
- [30] de Vries, H, Carrasco-Gallego, R, Farenhorst-Yuan, T, & Dekker, R. (2014). Prioritizing replenishments of the piece picking area. *European Journal of Operational Research*, 236(1), 126–134. doi:10.1016/j.ejor.2013.12.045
- [31] Martello, S., & Toth, P. (1987). Linear assignment problems. In S. Martello, G. Laporte, M. Minoux, & C. Ribeiro (Eds.). *Surveys in Combinatorial Optimization, Annals of Discrete Mathematics* (Vol. 31, pp. 259–282). Amsterdam, North-Holland: Elsevier Science Publishers B.V..