CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Control Engineering

Master's Thesis

Strategies for the Forklift Scheduling Problem

Jan Piskáček
Open Informatics, Computer Engineering
piskaja2@fel.cvut.cz

May 2018
Advisor: Ing. Antonín Novák

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Piskáček Jan** Personal ID number: **406396**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Open Informatics**

Branch of study: **Computer Engineering**

II. Master's thesis details

Master's thesis title in English:

Strategies for Forklift Scheduling Problem

Master's thesis title in Czech:

Strategie pro rozvrhování zásobovacích vozíků

Guidelines:

1. Study the problem of Forklift Scheduling and the related ones in the literature.
2. Improve the model assuming multiple forklifts with a renewable resource constraining each forklift.
3. Propose and implement an efficient algorithm solving the problem.
4. Evaluate list-scheduling and constructive strategies solving the problem; evaluate the proposed algorithm and the impact of the key parameters values; compare the proposed algorithm to the existing works.

Bibliography / sources:

[1] ADULYASAK, Yossiri, Jean-Francois CORDEAU a Raf JANS. The production routing problem: A review of formulations and solution algorithms. Computers and Operations Research. 2015, 55, 141-152. DOI: 10.1016/j.cor.2014.01.011. ISSN 03050548.
[2] COELHO, Leandro C., Jean-Francois CORDEAU a Gilbert LAPORTE. Thirty Years of Inventory Routing. Transportation Science. 2014, 48(1), 1- 19. DOI: 10.1287/trsc.2013.0472. ISSN 0041-1655.
[3] Desrosiers J., Lübbecke M.E. (2005) A Primer in Column Generation. In: Desaulniers G., Desrosiers J., Solomon M.M. (eds) Column Generation. Springer, Boston, MA
[4] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. Operations Research 199846:3 , 316-329

Name and workplace of master's thesis supervisor:

Ing. Antonín Novák, Department of Control Engineering, FEL

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **17.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Antonín Novák	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

25. 4. 2018 | Date of assignment receipt | Student's signature

Acknowledgement / Declaration

I would like to thank my family for support, encouragement and many pieces of advice throughout my studies. I would also like to thank my thesis supervisor Antonín Novák for key insights and commitment.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. 5. 2018

.....

Abstrakt / Abstract

Tato práce popisuje nový problém rozvrhování zásobovacích vozíků. Problém rozvrhování zásobovacích vozíků je minimalizační problém o nalezení periodického rozvrhu a plánu jízd vozíků v továrně, za dodržení materiálových požadavků všech strojů. V práci jsou popsány dva způsoby řešení. První je založen na konstruktivní heuristice. Druhý způsob je založen na metodě generování sloupců a modelu celočíselného programování. Kvalita obou způsobů řešení je otestována na vygenerovaných instancích problému. Dále jsou výsledky diskutovány z hlediska dopadu různých vstupních parametrů na kvalitu řešení. Výsledky ukazují, že volba vstupních parametrů silně ovlivňuje kvalitu různých způsobů řešení a že prezentované metody jsou schopny vyřešit instance až s 12 stroji a 10 periodami s kvalitou výsledku 25% od vypočítané dolní hranice.

Klíčová slova: problém rozvrhování zásobovacích vozíků, konstruktivní heuristiky, generování sloupců, lineární optimalizace

Překlad titulu: Strategie pro problém rozvrhování zásobovacích vozíků

This thesis introduces a new problem called the Forklift Scheduling Problem. The Forklift Scheduling Problem is a minimization problem of finding a periodic schedule and routing plan of forklift vehicles subject to material demands of machines in a factory. The thesis proposes two algorithms to solve the Forklift Scheduling Problem, the first is based on a constructive heuristic approach and the second is based on column generation method and an Integer Linear Programming model. We evaluate both algorithms on generated instances and discuss the effect of different parameter combinations on the quality of solutions. The results show that the parameters of instances heavily affect the quality of different strategies and that our methods can solve instances with up to 12 machines and 10 periods within 25% from our computed lower bound.

Keywords: forklift scheduling problem, constructive heuristics, column generation, linear optimization

Contents /

Tables /

Chapter 1
Introduction

Flexible manufacturing leads to a large number of product types with lower volume batches. A large number of product types brings considerable challenges in material supplying because the type and amount of materials needed is changing rapidly, as the machines on the shop floor are continually reconfigured and supplied with new material during production.

The objective of this work is to study and propose new strategies for supplying the material for production machines on the shop floor with varying and periodic material consumption. The informal problem statement could be summarized as follows: Given a shop floor layout and the material demand profile of machines across the scheduling horizon, build a schedule and a routing plan for the material supply with available forklift fleet such that the required material demand of each machine is satisfied and the forklift traveling time is minimized. It is also important to note that the problem statement assumes electrically powered rechargeable forklifts which adds complexity to the problem.

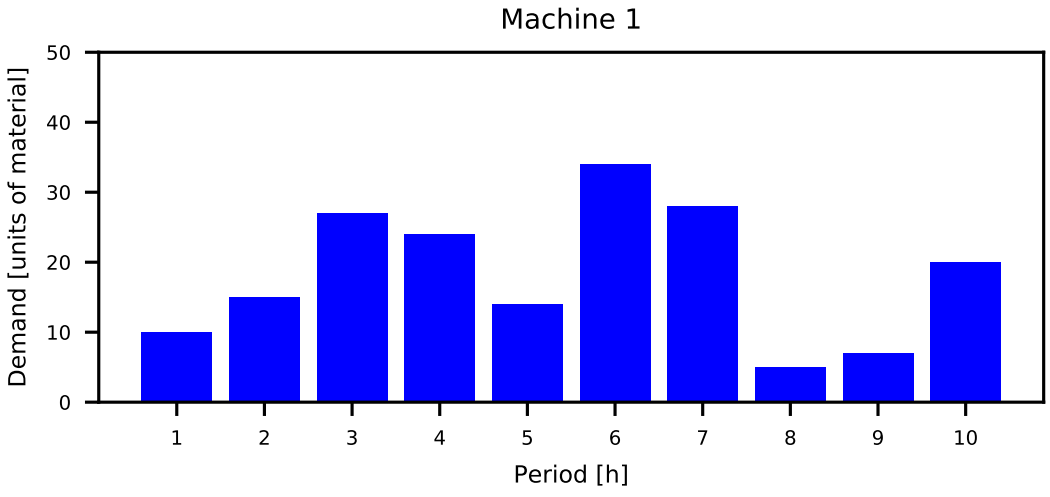

Figure 1.1. Example of machine demand over a 10-period horizon. The length of the period is one hour.

The travel time of forklift fleet is affected by the factory layout design. It is desirable to minimize the travel time of forklifts to gain greater efficiency. By minimizing the travel time, the forklift can be used for other activities in the factory or the forklift operator can continue with other tasks sooner. Improving overall efficiency is one of the major goals of warehouse management systems and reducing the travel time of forklifts can help this goal.

Although we are trying to find a strategy for the problem in the context of a factory and scheduling of forklift trucks, the context can be entirely different. In fact, the overwhelming majority of work related to our problem is in the context of Vendor Managed Inventory in Supply Chain Management [1]. Related work and key differences between forklift scheduling and similar problems are discussed in chapters 1.3 and 1.4.

The *Forklift Scheduling Problem* (*FSP*) naturally includes the *Traveling Salesman Problem* as a subproblem, and therefore it is $\mathcal{NP}$-hard. That means that there should be no algorithm with polynomial time complexity that can solve the problem to optimality unless $\mathcal{P} = \mathcal{NP}$. The formal definition of the problem and proof of its $\mathcal{NP}$-hardness is given in the following chapters.

1.1 Problem definition

An input instance of *FSP* is defined as follows:

- Graph $G = (V, E, t)$, where:
  - $V = \{0, ..., n\}$ is a set of vertices
  - $E = \{i, j\} : i, j \in V, i \neq j\}$ is a set of edges
  - $t : E \to \mathbb{R}_+$ is a mapping from the set of edges to non-negative real numbers

Graph G is:

- complete
- undirected
- simple
- weighted
- metric - satisfies the triangle inequality

The given graph represents the layout of the machine on the shop floor. Vertex 0 represents the parking and charging station of the forklift fleet. The set $V' = V \setminus \{0\}$ represents machines that consume material periodically. The number of periods is p. Mapping t represents the distances between vertices in meters. For every machine $i \in V'$ a maximum storage capacity for material W_i is given and also a p-dimensional vector

$D_i = (d_1, ..., d_p)$ that represents the material demands that need to be present in the storage area of the machine i at the end of every period. There can be more material in the storage area of a machine than is demanded; the material stays there until the end of a period, then the amount of material demanded by the machine is consumed.

The number of forklifts is P. All forklifts have the same maximum loading capacity C and also the same maximum battery capacity B. The battery discharge rate is dependent on the speed of the forklift. The current battery charge state can never go below 0 or above B. If a forklift has a low battery, it needs to be charged in the parking station that is represented by vertex 0.

The forklift fleet delivers material to the machines. Every forklift can execute multiple routes in a period. The routes have to start and end in the parking station, so every route is a cycle. There can be multiple routes scheduled for a forklift during a period but the sum of the traveling times cannot exceed the length of the period. The length of periods is H and is set to be one hour.

The material demands, maximum storage capacity and forklift loading capacity all have the same units.

The objective is to find a schedule and routing of the forklift fleet such that the sum of individual traveling times is minimized and the material demands of the machines satisfied. The output includes the set of routes for each forklift. Each route has to specify the following information:

- time needed to execute the route
- sequence of visited vertices
- amount of material to unload at every visited vertex

The output also includes a set of charging sessions for each forklift. Every session contains the time needed to complete the charging and the amount of battery capacity that was charged.

1.2 Complexity

To prove $\mathcal{NP}$-hardness of the Forklift Scheduling Problem we need to show a polynomial-time reduction from a problem that is known to be $\mathcal{NP}$-hard to our Forklift Scheduling Problem. In other words, for every input instance of the chosen $\mathcal{NP}$-hard problem, we have to be able to construct the corresponding instance of the Forklift Scheduling Problem in polynomial time. We will use the fact that *TSP* is $\mathcal{NP}$-hard, which was proven in the work of Richard M. Karp - *Reducibility among Combinatorial Problems* [2].

Proposition:

$$TSP \lhd_p FSP$$

Proof: Let's take an arbitrary instance of *TSP* defined by an undirected weighted graph that has the properties stated at the beginning of chapter 1.1, the goal is to find a cycle visiting every vertex of the graph with a minimal sum of traversed edges. We will show how to construct an instance of *FSP* from a *TSP* instance such that a solution that is given by a black box that can solve the *FSP* is optimal if and only if the solution to the corresponding instance of *TSP* is optimal.

For every vertex of *TSP* instance, we will create a corresponing vertex in the *FSP*, from these vertices we arbitrarily choose one to represent the vertex number 0 (forklift station). For every edge in the *TSP* instance, we add an edge between the corresponding vertices in *FSP* and we will also give it the same distance $t_{i,j}$. We will set the following parameters to the *FSP* instance:

- Number of periods $p = 1$
- For every machine $i \in V' : D_i = (1), W_i = 1$
- Number of forklifts $P = 1$
- Maximum loading capacity of a forklift $C = |V'|$ (just enough to carry one unit of material to every machine)
- Maximum battery capacity $B = \infty$
- Length of period $H = \sum_{i,j \in V, i \neq j} t_{i,j}$ (enough time to traverse all of the edges in the graph)

Since both graphs have the same weights on all edges and the only allowed speed is 1 m/s, the time it will take to travel between any two vertices in the *FSP* instance will be equal to the corresponding weight in the *TSP* instance.

Now let's take an instance of *FSP* constructed by the previous procedure and solve it with a black box that can solve any instance of *FSP*. By definition, the solution will be an optimal one. We know that vertex 0 has to be visited by definition and the rest of vertices too because they have a demand for one unit of material. The travel time of the forklift will be minimal, and that can only be achieved by taking the shortest cycle that visits all the vertices. The shortest cycle found by *FSP* black box corresponds to the optimal route of the *TSP* instance we began with. The time it takes to construct the *FSP* instance is polynomial in the number of vertices. ■

1.3 Related work

FSP is related to *Production Routing Problem* (*PRP*) [3]. *PRP* is described in more detail in Chapter 1.3.1.

The *Production Routing Problem* has a simplified version where some aspects of the problem are omitted, and it is called *Inventory Routing Problem* (*IRP*) [4]. *IRP* is the closest problem to *FSP* mentioned in scientific papers that we found. We believe that it can be considered as a modification to the *IRP*. The motivation behind these problems is to minimize supply chain costs.

Supply Chain Management includes planning of all activities from producing goods or services to transporting the produced goods from the supplier to customers (in this context, the customers are retail stores). Traditionally, the replenishment decisions for products are made by the customers, but in Vendor Managed Inventory (VMI) [5] practice it is the supplier that makes these decisions. VMI aims to reduce all costs involved in the supply chain. These costs include unit production cost, fixed production setup cost, inventory holding costs and transportation costs [4]. More detailed description of *PRP* and *IRP* is presented in the following chapters. The last related problem we present is *Periodic Vehicle Routing Problem* (*PVRP*) [6].

1.3.1 Production Routing Problem

The *Production Routing Problem* involves four layers of decision making in the supply chain [7]:

- How many products to manufacture and when?
- How many products to store, where to store them and for how long?
- How many products to deliver and to whom?
- Which route to take when delivering the products?

PRP combines *Lot-Sizing Problem* (*LSP*) [8] and *Vehicle Routing Problem* (*VRP*) [9]. The decisions stated were usually planned and optimized step by step. As previously stated VMI takes all of these decisions into consideration when creating a plan to execute. Because of that, it is possible to reduce costs even more than if those decisions were dealt with step by step.

An instance of *PRP* is defined on a complete directed graph, where one node represents a plant and rest represent customers. There is also a finite set of periods. Each customer has a demand in every period that has to be met. Each customer also has some limited inventory space that cannot be exceeded. Products can be produced at the plant and then at a later time distributed between customers by a set of identical vehicles, these vehicles have their own maximum capacity they can handle. The

production of products has a setup cost and also cost per unit produced. Holding the products at the plant and at the customers incurs inventory holding costs. There are also transportation costs associated with every edge of the graph.

The objective is to minimize the total production, inventory and transportation costs. The output to an instance of the *PRP* has to specify the production quantity at the plant in every period, the number of products that will be stored at the plant and at the customers, the number of products to distribute between the customers and also the routes the vehicles take to deliver them. Note that our description is very condensed and states only the necessary information and details to understand what the *Production Routing Problem* is about.

The *Forklift Scheduling Problem* has many similarities with the *PRP*. Both problems are defined on a graph; both involve a set of periods and demands of some form (the customer demand for products in *PRP* corresponds with the machine demands for material in *FSP*). Both problems also include vehicles that can distribute the demanded items (vehicles in *PRP*, forklifts in *FSP*).

Numerous solution approaches are mentioned in the review paper by Adulyasak, Cordeau and Jans - *The production routing problem: A review of formulations and solution algorithms* [4]. The solution approaches include a branch-and-cut algorithm, decomposition-based heuristics, branch-and-price and mixed integer programming heuristics and also metaheuristic approaches such as Adaptive Large Neighborhood Search (ALNS) and memetic algorithms.

The size of solved benchmark instances introduced by Boudia et al. is 50 to 200 customers and 20 time periods [4]. Another benchmark was created by Archetti et al. which includes instances up to 100 customers and 6 periods. Numerous papers on the *PRP* used the mentioned benchmarks to compare their solution with other researchers. The best average objective value on the Boudia et al. benchmark was achieved by Adulyasak et al. [7]. It uses a modified Adaptive Large Neighborhood Search to solve the problem.

Note that compared to the benchmarks introduced by Boudia et al. we generate instances of *FSP* smaller in terms of vertices (interpreted as customers or machines).

1.3.2 Inventory Routing Problem

Instead of describing the *Inventory Routing Problem* from scratch, we will show the differences between *PRP*. We end up with the *IRP* when we modify *PRP* not to involve decisions on the production amount. It is assumed that there are just enough products to satisfy all the customer demands in all periods.

decision level	PRP	IRP	FSP	VRP	LSP
production	x				x
inventory and storage	x	x			x
distribution	x	x	x		
routing plan	x	x	x	x	

Table 1.1. Illustrating the various levels of decision making in different problems with *FSP* added [7].

According to Coelho [1], the study of *IRP* is rooted in the paper of Bell et al. - *Improving the Distribution of Industrial Gases with an On-Line Computerized Routing and Scheduling Optimizer* [10].

Since the *IRP* is the closest problem to *FSP*, we will describe the work of Bell et al. in more detail. The problem involves one central depot and multiple vehicles and customers. The customers have fixed daily demand - each customer can have different demand but the demand of one customer does not change from day to day, that is a difference from *FSP*. The vehicle fleet is heterogeneous; every vehicle has different capacity and operational costs. Usually, the delivery for one customer must be shipped only by one vehicle. There are numerous additional constraints that are sometimes not present in a typical *IRP* formulation, these constraints include:

- cannot let the customer supplies drop down to zero, therefore it is needed to maintain a base level of supply at the customer
- daily demand can sometimes change due to unexpected events
- the delivery of products can arrive at the customer only in certain time windows
- some vehicles cannot be used for delivery to certain customers

The total cost of distribution includes the salary of the drivers, toll, vehicle maintenance costs, depreciation, prices of fuel and other costs independent of distance traveled. The objective is to maximize the price of delivered goods minus the total costs.

The solution method used by Bell et al. involves Lagrangian relaxation algorithm to solve Mixed Integer Programs (MIP). The MIP gave solutions proven to be near optimality, i.e., 0.5 - 2.0 % from the optimum. Typical problem instance has from 150 to 400 customers and up to 30 vehicles. The size of MIPs is up to 800000 variables and 200000 constraints. Their solution also includes a module able to change the resulting schedule if the schedule becomes undesirable due to unexpected events.

The paper also states that Mixed Integer Programs solved by their algorithm were *"apparently the largest integer programs ever solved to near optimality on a routine basis."* and that the Lagrangian relaxation algorithm developed by them *"extended the*

theory of optimization". The optimization was saving between 6% - 10% of operating costs of the company.

The paper written by Coelho, Cordeau and Laporte [1] provides a comprehensive review of literature about *IRP* and presents a way to categorize different variants of the problem. We will now summarize the categorization approach and try to describe our *Forklift Scheduling Problem* with it. The variants of *IRP* are classified according to two schemes: the first considers the structure of the problem and the second considers the availability of information on the demand. There are also different modifications of *IRP* with multiple types of delivered products and additional constraints to make the results more practical in real-world applications.

The entries in **bold** in table 1.2 are the ones that are present in our *Forklift Scheduling Problem*.

criterion			
time horizon	**finite**	infinite	
number of customers and plants	one to one	**one to many**	many to many
routing method	direct	**multiple**	
vehicle composition	**homogenous**	heterogenous	
number of vehicles	single	**many**	infinite
demand information	**deterministic**	stochastic	dynamic

Table 1.2. Classification according to Coelho, Cordeau and Laporte in the paper *Thirty Years of Inventory Routing* [1].

1.3.3 Periodic Vehicle Routing Problem

The PVRP is the last related problem we will mention. It is probably the least similar problem presented. However, it includes the factor of periodicity and routing decisions which makes it worth mentioning.

According to paper *Forty Years of Periodic Vehicle Routing* written by A. M. Campbell and J. H. Wilson, which reviews the applications and solution methods of *PVRP* [11], the *PVRP* has first appeared in 1974 in a paper focusing on garbage collection (Beltrami and Bodin, Networks 4, 65–74). Many varieties of the problem have been proposed; we will try to state a short informal description of the problem according to a paper *The Period Routing Problem* [12] written by N. Christofides and J.E. Beasley.

We are given a central facility and a set of customers that must be supplied by vehicles starting and ending in the central facility. There is also a set of p days, and

each customer requires a certain number of visits by a vehicle during the p days. The visits of a customer may only occur in predefined sets of k-day combinations. The goal is to find a set of routes with the minimum cost of distribution that satisfies the number of visits required by each customer and uses only the predefined k-day combinations for delivery. Slightly different problem statements of the *PVRP* also exists, and the main difference is in the definition of the periodicity of the deliveries.

Similarly to *PRP* and *IRP*, *PVRP* includes routing and it includes decisions on more levels that heavily affect each other. The first choice in *PVRP* is between the available k-day combinations, then we know which customers are we visiting on each day, then we solve *VRP* in each day. Joining those decisions can lead to significant improvements as in *PRP* and *IRP*, but the problem becomes very hard. Because of this, a number of relaxations and heuristics of the *PVRP* were proposed by Christofides et al. [12].

Numerous varieties of *PVRP* were studied, for example, one with only one vehicle, which makes it effectively a *Periodic Travelling Salesman Problem*, or other varieties with time windows or multiple central depots.

1.4 Key distinction between similar problems

There are key differences between the *Inventory Routing Problem* and our *Forklift Scheduling Problem*:

- the objective function
  - *IRP* minimizes the total cost
  - *FSP* minimizes the total travel time of forklifts
- *IRP* includes the inventory holding costs, *FSP* has no corresponding element
- all of the studied *IRP* formulations permit only one trip of a vehicle in a period, whereas in *FSP* the number of trips is only limited by the constraint that all of the trips have to be finished before the end of the current period
- *FSP* includes a renewable resource constraining each forklift - the battery

1.5 Contributions

There are five main contributions of the thesis. We defined the Forklift Scheduling Problem and explained the main similarities and differences from other problems such as Production Routing Problem and Inventory Routing Problem. We also developed a constructive heuristic algorithm with a modular design that can solve FSP. Then we created an LP formulation of the problem and applied column generation algorithm

on it to obtain lower bounds. Furthermore, we created an ILP formulation that takes a non-integer solution from the column generation algorithm and produces an integer solution which is better than solutions given by the constructive heuristic algorithm in approximately 50% of measured instances. Lastly, we tested the developed algorithms on a broad set of generated problem instances and analyzed the results.

1.6 Summary and outline

So far we have defined the *Forklift Scheduling Problem* and presented the motivation behind studying efficient algorithms to solve the problem. Similar problems, namely *Production Routing Problem*, *Inventory Routing Problem* and *Periodic Vehicle Routing Problem*, were described and the relation to our *FSP* explained and key differences between those problems were shown. Furthermore, we classified *FSP* according to scheme created by Coelho et al. [1].

In Chapter 2 we show a generator of problem instances for *FSP*. In Chapters 3 and 4 we present two methods to solve FSP and obtain a lower bound, namely the constructive greedy heuristic similar to list scheduling approach and column generation method based on LP formulation of the problem. We evaluate the methods in Chapter 5 and conclude the thesis in Chapter 6.

Chapter 2

Instance builder

In this chapter, we describe the main features of the software that was used to generate problem instances. An instance of *FSP* is represented as a Java class. As we have seen in chapter 1.1 the structure of the instances is fairly complicated and extensive. The `Instance` class has the following members:

- vertex count
- hour count
- width and height (in meters)
- minimum vertex distance
- object of class `Vertex` that represents the station
- array of `Machine` objects
- array of `Vehicle` objects

It would be very inconvenient to construct an `Instance` object with a constructor because of the number of fields and also because some fields include other objects that need to be properly instantiated too. When faced with a creation of a complex object it is useful to use the *Builder* design pattern.

The Builder design pattern is creational design pattern. It is one of the patterns presented in the influential book *Design Patterns: Elements of Reusable Object-Oriented Software* written by E. Gamma, R. Helm, R. Johnson and J. Vlissides. Builder pattern is used to separate the construction of a complex object from its representation.

However, there are still problems with the pattern, because the creation steps of the `Instance` object need to proceed in correct order. Fortunately, an extension of the Builder pattern exists - the `Step Builder` pattern. The Step Builder enforces an order of the builder methods, which is very convenient. If the Step Builder is set up correctly, the programmer cannot make a mistake when creating a new instance with it; the Java source code would not compile if there was an error in the order of steps. Another advantage of Step Builder is being able to offer multiple alternatives for a building step. For example, in the context of FSP, we would sometimes like to set the coordinates of the vehicle station by hand, but sometimes generate it randomly. Step Builder supports that option.

The key to Step Builder implementation is to have a Builder, and an interface for every step of the Builder, the enforcing of the order of steps is achieved by having the methods return only a specific interface type and having the Builder implement all the interfaces. To give the reader a better idea, here is a short code snippet to demonstrate the concept. Note, that only a few steps are shown in the snippet.

```
private static class StepBuilder implements VertexCountStep,
            HourCountStep, HeightStep, WidthStep, ..., BuildStep {
            /* private fields, omitted */

           @Override
            public HourCountStep setVertexCount(int vertexCount) {
                this.vertexCount = vertexCount;
                return this;
            }

            @Override
            public WidthStep setHourCount(int hourCount) {
                this.hourCount = hourCount;
                return this;
            }

            @Override
            public HeightStep setWidth(double width) {
                this.width = width;
                return this;
            }
}
```

Listing 2.1. Short code snippet showing the idea of StepBuilder.

The building steps also include options on setting a sensible default values, for example, there is an option to set the battery capacity to 100.0 units, but if the user wants to experiment with different capacities, there is also an option to set arbitrary capacity. The same option is available for setting the battery charge and discharge rates of vehicles. The is also an option to set the random coordinates of vehicle station or to set it by hand, the same can be done with machines and their demands. Another notable building step is setting a minimal vertex (machine) distance between each other, because when we let the builder generate the vertex coordinates at random, we want to avoid the cases where the random coordinates end up clustered near each other, so by adding that option, we can force the vertices to be more spread out.

The full list of possible building steps of the *FSP* instance Step Builder is:

- setVertexCount
- setHourCount
- setWidth

- setHeight
- setMinVertexDistance
- setStation or setStationRandom
- setMachines or setMachinesCapacityRange
- setVehicleCapacity
- setBatteryCapacity
- setChargeRateFactor or setDefaultChargeRateFactor
- setDischargeRateFactor or setDefaultDischargeRateFactor
- setVehicleCount
- build

An important thing to note that when computing the distance matrix of vertices from the generated coordinates, we use Manhattan distance (1) rather than Euclidean distance (2). This choice was made to better reflect the layout of factories and warehouses. The routes the forklifts take to deliver the material to the machines is very likely to be rectangular due to the grid structure of factories and warehouses. A good example of a warehouse layout showed in a paper by Davide Giglio [13] can be seen in Figure 2.3.

$$d_{manhattan}(x, y) = |x_1 - y_1| + |x_2 - y_2| \tag{1}$$

$$d_{euclidean}(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{2}$$

To see how the resulting *FSP* instance looks like, we use Python script to plot the factory layout and bar chart of the demand vectors of individual machines. The grey transparent circles around the vertices on Figure 2.1 show the minimum vertex distance, so it is easy to see that the required minimum distance is satisfied.

The Figure 2.2 shows a randomly generated vector of demands of a machine. The storage capacity is always bigger than the periodic demand, otherwise, the instance would be infeasible - the machine would require more material than it is physically possible to fit into its storage area.

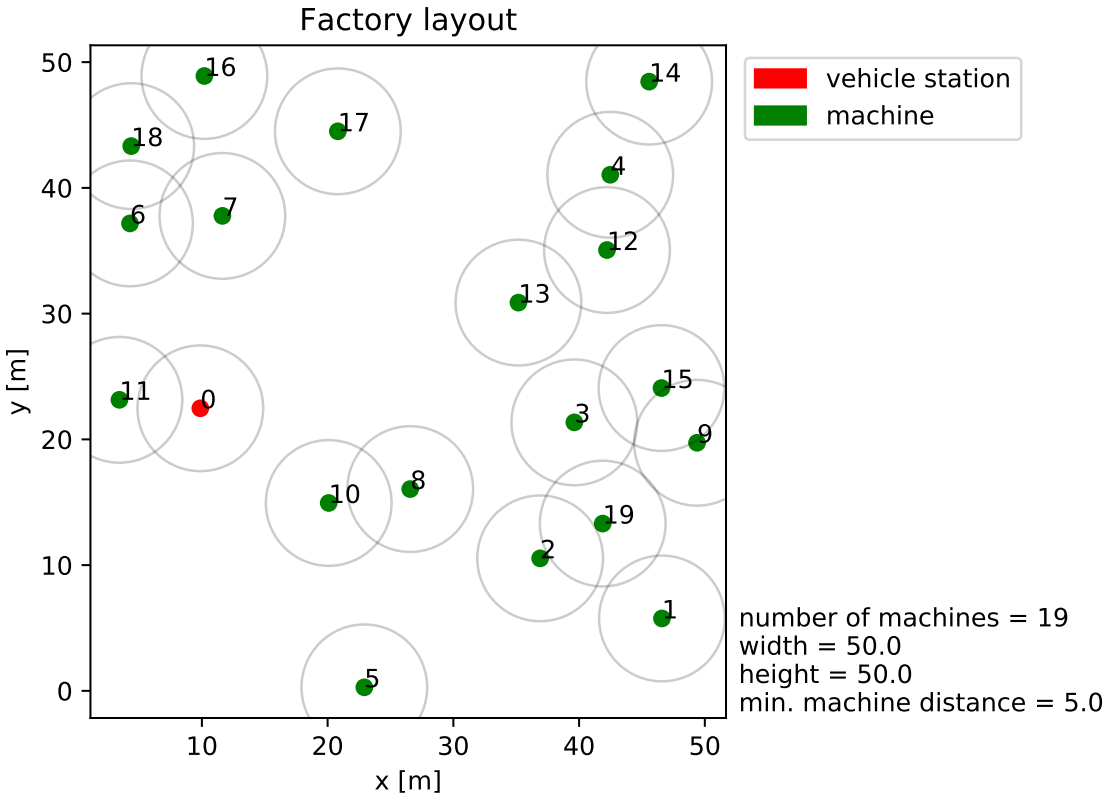

Figure 2.1. Example of a factory layout with 20 vertices, the forklift station is colored in red, the machines are colored in green. Minimum vertex distance is set to 5 meters.

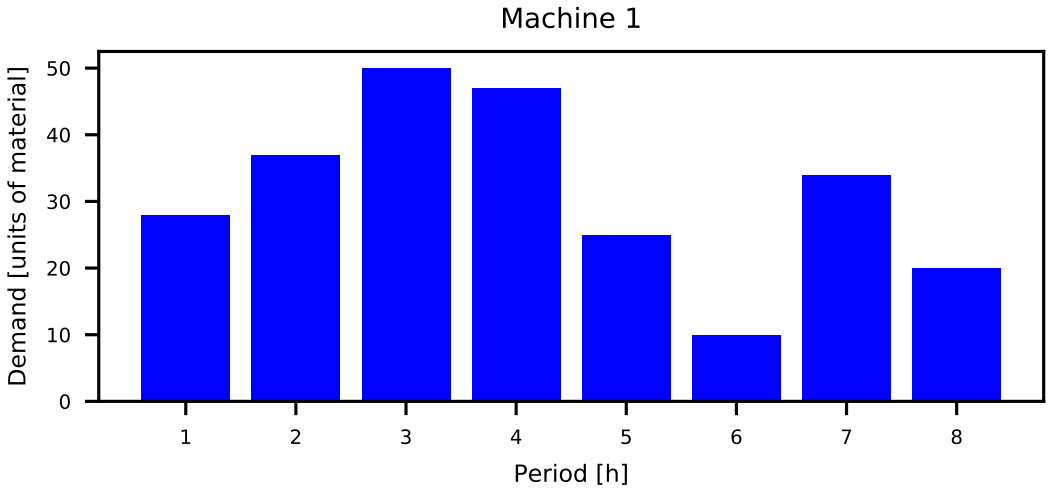

Figure 2.2. Example of machine demand over an eight period horizon. This periodic demand was generated randomly between 5 and 50 units of material per period.

A full example of *FSP* executing all the required steps and building the instance looks as follows:

```
Instance instance = Instance.Builder.getStepBuilder()
                            .setVertexCount(10)
                            .setHourCount(24)
                            .setWidth(30)
                            .setHeight(30)
                            .setMinVertexDistance(5)
                            .setStationRandom()
                            .setMachinesCapacityRange(30,50)
                            .setVehicleCapacity(300)
                            .setBatteryCapacity(100.0)
                            .setDefaultChargeRateFactor()
                            .setDefaultDischargeRateFactor()
                            .setVehicleCount(1)
                            .build();
```

Listing 2.2. Creating an instance of *FSP* with the StepBuilder.

In the next chapter, a greedy constructive heuristic algorithm will be described.

2.1 Implementation details

Initially, the implementation of both the instance builder and the algorithms was written in *Python*. *Python* enabled faster implementation and also convenient libraries for plotting, namely *matplotlib* and *plotly*, *numpy* library is also used. For easy conversion from and to JSON, the *json* module is used. We decided to switch from *Python* to *Java*, because of personal preference and strong typing of *Java* that is less error-prone. The *Java* project uses only one external library, and it is *Gson* to easily convert an object to and from JSON. Only the parts of code used for visualization remained in use in the *Python* project. The solver used in both projects is *Gurobi*. *Gurobi* was chosen as the solver because of previous experiences with it and *Gurobi* also offers free academic licenses.

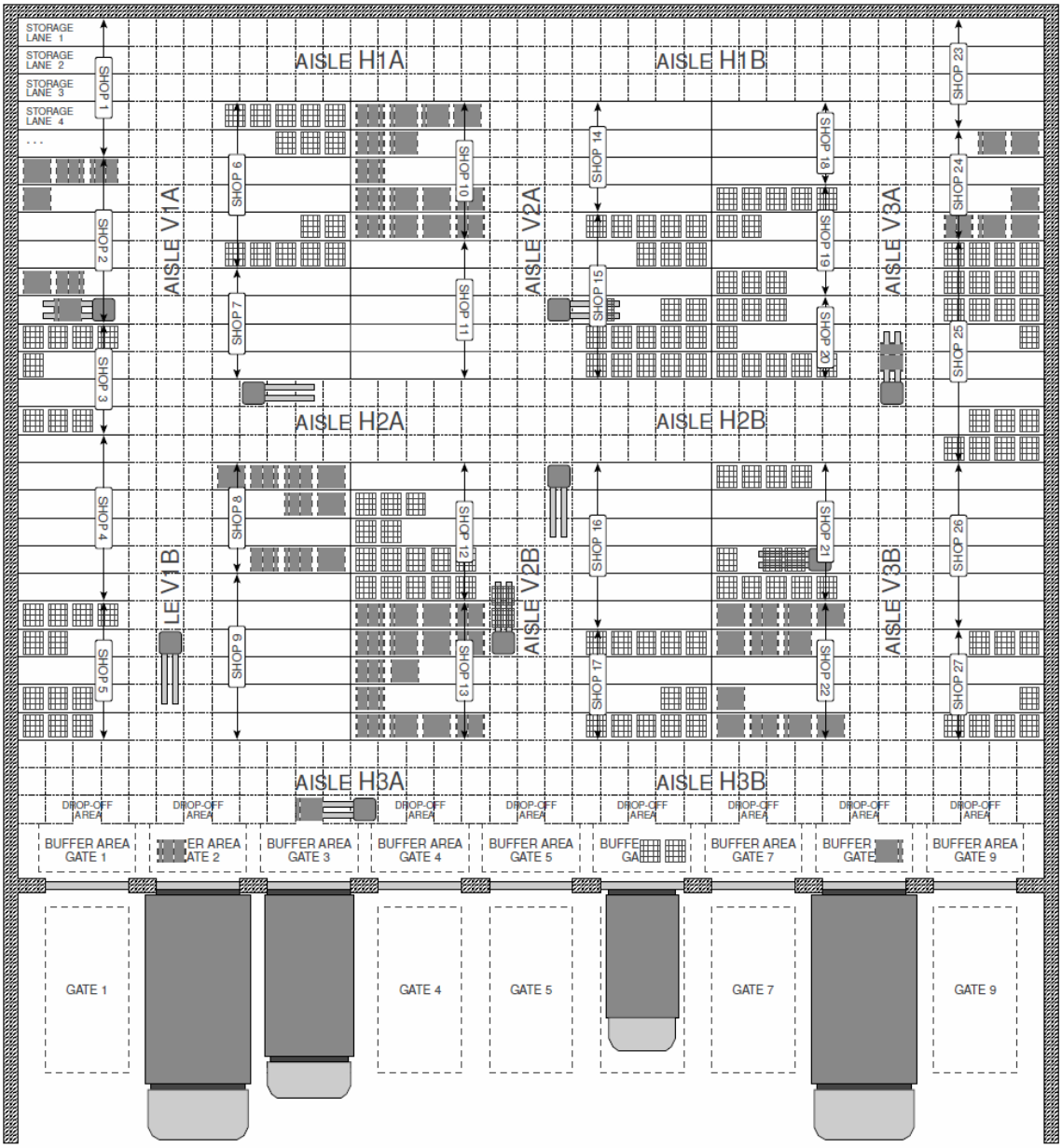

Figure 2.3. Example of a warehouse layout with a grid structure [13].

Chapter 3

Constructive heuristics

The first solution method to create an algorithm to solve FSP was done through a constructive approach. In general, a constructive approach constructs a solution step by step, always looking at the current state of the solution and making a new decision based on the current state. We terminate the algorithm when the solution is fully constructed.

The constructive approach can also be a greedy one, that means that from our current state, we are trying to make the best possible progression towards achieving the optimal solution. However, there is no guarantee that the greedy approach always gives the optimal solution. This fact is related to the problem of local optima, which is one of the themes in the study of local search methods. Only problems with a certain structure can be solved to the optimum by a greedy algorithm.

There are two main options we considered for creating a constructive algorithm to solve FSP. The first one is to define a time quantum and update the current state of the solution in those time quanta. The time quantum does not have to be always the same length, but it has to be no longer than the time we could decide that a new action should be performed, e.g., routing or charging. For example, we start in time 0, we decide to send forklift v_1 to machines m_1, m_2, m_3, and forklift v_2 to machines m_4, m_5. We know how much time those trips take so that we can compute the greatest common divisor of those times and that would be our quantum. We would update the current state of the whole algorithm according to this time quantum until we could make a new decision on routing or charging a vehicle. By current state, we mean the current position of all forklifts, their battery state, and the current state of all storage areas of machines. When we reach the point when a new decision can be made, we take all the ongoing actions and the newly created action and again recompute the time quantum and proceed to update the solution according to it. However, we did not take this approach, because it seems unnecessarily complicated, so we came up with a less complicated, yet comparably capable method.

The second method is to generate events and handle them in the correct order. Handling of an event can produce another event, and the algorithm continues to handle events until there are no more events to handle. To implement this algorithm, we used a priority queue implemented with a well-known data structure called *heap*, we used a

min-heap in particular. Heap is a tree-based data structure that supports deleting the minimum element in $\Theta(log\ n)$ and inserting new elements in $O(log\ n)$.

The elements we insert into the heap are *events*. The events are extracted and inserted into the heap with the key being their starting time. There are two types of events. The first type is *end of period* event; it checks the amount of material that is present in the storage area of all machines and makes sure that it is at least the demanded amount for the current period and at most the maximum storage capacity. The second is *vehicle ready* event, it looks at the current state of machines and computes a new route to execute. The *vehicle ready* event also inserts new *vehicle ready* events into the heap, because immediately after the completion of one route, the forklift is once again ready to execute another.

The general outline of the algorithm is simple. First, we insert initial events into the heap, which includes *vehicle ready* events, and for the end of every period, we add the *end of period* events. The number of initial *vehicle ready* events, of course, depends on the number of vehicles available in the problem instance. By handling the events, new events get inserted into the heap. We always extract one event at a time from the heap (the one with minimum start time) and handle it; we repeat this process until there are no events left.

A route is defined by the path it takes and also by the amount of material it unloads at each visited machine, all other parameters such as the time it takes to execute it, and the battery that is drained by performing the route can be easily computed. It is important to note, that when the algorithm decides to execute a certain route, it does not wait for the end of the execution to update the vehicle battery or to update the amount of material in storage areas. It simply updates those parameters immediately after it is decided that a particular route will be taken.

This kind of ”eager” updating eliminates the problem of two forklifts being ready at the same time. Suppose we are at the start of the algorithm and two forklifts are now ready. So, we extract the most recent event from the heap, which is the first forklift being ready to execute a route. We decide which route it will take and insert new *vehicle ready* event to the heap, with the key being the time it takes to execute the route plus current time. Now imagine we did not immediately update the state, the second forklift ready would see the same state as the first forklift, and it would try to execute the same route. This behavior is not desirable. When we immediately update the state, the second forklift sees the current state as if the first forklift already completed its route, which enables the second forklift to take a different, more desirable route.

```
input  : An instance of FSP and a Strategy to use
output: Set of routes and charging sessions executed in every period by
        every vehicle

create empty solution
initialize heap
add initial events to heap

while heap not empty do
    event ← extract minimum from heap
    result ← handle event with given strategy
    add result to solution
    if result signals a failure then
        set solution as infeasible
        break
end

return solution
```

Figure 3.1. Pseudocode of the main loop of the algorithm. Note that one of the inputs to the algorithm is a Strategy, we will elaborate on Strategy in further chapters.

We already described how the *end of period* and *vehicle ready* events are handled. The handling of *end of period* event is simple, so we will not describe it further, however handling of *vehicle ready* event is the most important aspect of the algorithm.

When a vehicle is available to deliver material to machines, we need to make some decisions:

- Which machines should be visited?
- Which path should we take to visit all the selected machines?
- How much material should we unload at each visited machine?

Most of the time, these decisions are done one by one, which means that the output of the first one is the input to the second, etc. There are multiple possibilities how to handle each decision. For example, when choosing machines to visit, we can only choose the one with the biggest demand or we could choose k machines with the biggest demand. We could also choose the set of machines to visit randomly. We call all the different possibilities on how to handle the mentioned decisions *strategies*.

However, there is one general skeleton of the *vehicle ready* handling algorithm regardless of chosen strategies. In the next chapter, we will describe the strategies for each decision in more detail. Now, we will present the general skeleton of the algorithm:

```
input  : A current state of instance of FSP and a Strategy to use
output: Route or charging session to be executed by the current vehicle

remaining time ← get remaining time to the end of the period
remaining battery ← get remaining battery capacity of the vehicle
selection pool ← strategy - get selection pool

if selection pool is empty then
    charge vehicle until the end of the current period
    add new vehicle ready event into the heap with time equal to the start of the
     next period
    return charging result
else
    strategy - initialize selection strategy
    while strategy - has next do
        machines ← strategy - get next set of machines
        tour ← strategy - get routing to selected machines
        time ← get time needed for execution of the tour
        battery ← get battery needed for execution of the tour
        if remaining time < time or remaining battery < battery then
            continue
        consume the battery needed for execution of the tour
        fill machines according to strategy
        add new vehicle ready event into the heap with time equal to the expected
         length of the tour
        return routing result
    end
    charge vehicle to its full battery capacity
    return charging result
end

return solution
```

Figure 3.2. Pseudocode of *vehicle ready* event handling method.

As we can see on Figure 3.2, there is a general skeleton of the algorithm which says what type of decision is made when, but the concrete way in which the decisions are made are tied to the selected *strategy*.

3.1 Strategy

The input to the handling method of *vehicle ready* event is a strategy. Strategy object consists of four distinct strategies, the types of strategies are:

- SelectionPoolStrategy - in green
- SelectionStrategy - in yellow
- RoutingStrategy - in orange

- FillingStrategy - in cyan

In Java, it is implemented with interfaces, every strategy from the list is an interface and we can plug-in any object that implements the interface. The main Strategy object is also created with the StepBuilder pattern. For example, if we come up with a new way of selecting machines for delivery, we just create a class that implements the SelectionStrategy interface, and we can easily construct a Strategy that uses our new Selection Strategy.

By designing the Strategy in the described way, it is very modular and easy to extend. However, not all strategies are compatible. For example, we would employ a selection strategy that selects multiple machines to deliver the material to, but we would set a direct routing strategy for the routing strategy. There is a direct conflict between those strategies; one selects many machines, and the other can only create routes to one machine. This problem has to addressed by the StepBuilder, to prevent the creation of invalid strategy combinations.

3.1.1 Selection Pool Strategy

Selection Pool Strategy chooses the machines that will enter the selection process. It can be thought of as a filter. We are not choosing any machines to deliver the material to; we are only creating the set from which the next strategy, the Selection Strategy, will choose the machines. There are two Selection Pool Strategies implemented.

Demand Sort Selection Pool sorts all the machines according to the amount they need to satisfy their demand in the current period. It discards all machines that already have enough material to satisfy the demands.

Random Sort Selection Pool also discards all machines that already have enough material; then it randomly shuffles the remaining machines.

3.1.2 Selection Strategy

Selection Strategy chooses machines that will be visited during the trip. If the chosen machines cannot be visited due to low battery or not enough time to execute the route, the selection strategy will try another set of machines. It can be seen on figure 3.2 that its use is very similar to an iterator, calling *has next* in a while loop and then getting the next set of machines with *get next*.

Largest Demand Selection selects machines one by one, in the order determined by Selection Pool Strategy.

Random Selection every time randomly selects a new subset of machines.

Subset Selection selects subsets of machines from the ones with the largest cardinality.

K-element Subset Selection selects only k-element subsets at maximum from the machines, the k is set to 2 initially.

ILP Selection chooses machines according to an *Integer Linear Programming* (ILP) model. It is the only strategy, which decides on more levels than one because it not only chooses the machines but also decides how much material will be delivered to them. The model tries to select as many machines provided that it can satisfy their demand for the current period. It penalizes the solutions that would deliver more material than necessary to a machine; it does so with the negative sum of y_m in objective function (1).

The model also includes a feedback mechanism that can add additional constraints if the machines chosen in the previous iteration could not be visited due to low battery or not enough time. When such thing occurs, the feedback ensures that the previously selected machines cannot be selected together again.

Objective function:

$$max \quad \sum_{m \in M} x_m - y_m \tag{1}$$

Subject to:

$$\sum_{m \in M} z_m = C \tag{2}$$

$$x_m \leq z_m \quad \forall m \in M \tag{3}$$

$$(BigM)\ x_m \geq z_m \quad \forall m \in M \tag{4}$$

$$d_m - c_m \leq z_m + (BigM)\ (1 - x_m) \quad \forall m \in M \tag{5}$$

$$c_m + z_m \leq W_m \quad \forall m \in M \tag{6}$$

$$c_m + z_m - d_m \leq (BigM)\ y_m \quad \forall m \in M \tag{7}$$

Variables:

- $z_m \in \mathbb{N}_0 \ldots$ amount of material to unload at machine m
- $x_m \in \{0, 1\} \ldots$ indicator variable equal to one if and only if z_m is enough to satisfy the demand of machine m in current period
- $y_m \in \{0, 1\} \ldots$ indicator variable equal to one if and only if z_m is strictly more than enough to satisfy the demand of machine m in current period

Parameters:

- $M \ldots$ set of machines
- $C \in \mathbb{N}_0 \ldots$ vehicle material capacity

- $d_m \in \mathbb{N}_0 \ldots$ the demand of machine m in current period
- $c_m \in \mathbb{N}_0 \ldots$ amount of material already present in the storage area of machine m
- $W_m \in \mathbb{N}_0 \ldots$ maximum storage capacity of machine m
- $BigM \in \mathbb{R} \ldots$ sufficiently large constant

The objective function (1) rewards the model for filling a machine just enough to satisfy the demand and penalizes the model for exceeding it. Constraint (2) ensures that the delivered amount is equal to the vehicle capacity. However, this constraint cannot always be satisfied, so if the model is infeasible, we solve a relaxed version that only ensures that the delivered amount is less or equal to the vehicle capacity. Constraints (3) and (4) tie the indicator variable x_m with z_m. Constraint (5) forces z_m to be at least the amount of material that is required to satisfy the demand of machine m. Constraint (6) prevents z_m from exceeding the maximum storage capacity W_m. Constraint (7) forces the indicator variable y_m to be equal to one if the demand of machine m is exceeded.

3.1.3 Routing Strategy

Routing Strategy chooses the path to take to the selected machines. The path needs to start and end at the forklift station, so it will always form a cycle on the underlying graph.

Direct Routing constructs the most simple kind of path - a direct path from the station to a machine and back.

TSP Routing constructs the shortest cycle that visits all the selected machines. It does so by solving a TSP ILP model with lazy constraints to eliminate sub tours.

Objective function:

$$min \quad \sum_{(i,j)\in E} d_{i,j}\, x_{i,j} \tag{8}$$

Subject to:

$$\sum_{(i,j)\in E} x_{i,j} = 1 \quad \forall j \in V \tag{9}$$

$$\sum_{(i,j)\in E} x_{i,j} = 1 \quad \forall i \in V \tag{10}$$

$$\sum_{(i,j)\in S : i\neq j} x_{i,j} \leq |S| - 1 \quad S \subset V, S \neq \emptyset \tag{11}$$

Variables:

- $x_{i,j} \in \{0,1\}$ ... is equal to one if and only if vertex i is in the cycle just before vertex j

Parameters:

- V ... set of vertices, consists of all machines and forklift station
- E ... set of edges
- $d_{i,j} \in \mathbb{R}^+$ ... the distance between vertices i and j

The objective function (8) minimizes the total distance of the cycle. Constraints (9) and (10) ensure there is only one incoming and one outgoing edge for every vertex. Constraints (11) require that every strict subset of vertices has at most $|S| - 1$ edges, which means that there are not enough edges to form a cycle in the set S. However, adding all constraints (11) is not practical since there are exponentially many subsets of V. Therefore they are added lazily into the model [14]. Whenever we get a solution from the solver, we can test the solution in a callback for any sub tours, if a sub tour is found, the constraint (11) is added, effectively preventing the sub tour from being included in the solution again.

3.1.4 Filling Strategy

Filling Strategy determines how much material will be unloaded at the selected machines. It needs to make sure that the maximum storage capacities of machines are not exceeded.

Even Filling fills the machines as evenly as possible. The leftover material is then given to the machines in the order determined by the Selection Pool Strategy

Random Filling fills the machines randomly.

Chapter 4

Column generation method

The second general approach for solving the FSP uses column generation. Column generation is a method to solve large linear programs efficiently. By large, we mean linear programs with an exponential number of variables with regard to the size of the input. Before we start to explain our application of column generation method, we will introduce the general notion of linear programs, their duality, dual prices and column generation.

Linear programs model problems of minimizing a linear objective function subject to a set of linear inequality constraints. Our description of a linear program is only one of the possible forms of a linear program. However, the critical point is that all the forms are equivalent, which means that they can be rewritten in any other form. The form we present is *inequality form* [15].

$$\begin{aligned} &\textit{minimize} && \mathbf{c}^T\mathbf{x} && (1)\\ &\textit{subject to} && \mathbf{A}\mathbf{x} \geq \mathbf{b} && (2)\\ & && \mathbf{x} \in \mathbb{R}^n \end{aligned}$$

Linear programming has broad application in modeling problems of routing, scheduling or planning. The primary industries that utilize linear programming models are manufacturing, transportation, and telecommunications [16]. LP is also used in operational research.

One of the essential theories of linear programming is duality theory [17]. Duality theory states that for every LP problem, called the primal, exists a dual problem that has a relationship with the primal. The theory gives essential insights and enables the use of numerous algorithms for solving LP.

For example, in case of minimization problems, the solution of the dual problem gives us a lower bound on the solution given by the primal. Another important thing is that each constraint in the primal problem corresponds to a variable in the dual and each variable in the primal corresponds to a constraint in the dual.

When we solve the primal problem, we can compute a dual price for each constraint; the dual price is the optimal value of the corresponding variable in the dual problem. The dual price tells us how much improvement can we get by relaxing the associated constraint by some small amount.

Finally, we introduce the method of column generation also known as delayed column generation; column generation helps with solving linear programs with a massive number of variables more efficiently. The idea behind column generation is similar to the idea of adding lazy constraints to an LP model 3.1. Lazy constraints help with the problem of exponentially many constraints, so they are added lazily into the problem, and column generation uses the same idea applied to variables. The FSP problem exactly suffers from the problems described. It can potentially have a massive number of variables because there are many routes to choose from, and for every route, we also have to decide how much material will be delivered to each machine.

The method works by solving the primal problem, also called the restricted master problem (RMP) with only a subset of variables present. After solving the RMP, we can get the dual prices and construct a pricing problem. In our case, the pricing problem is an ILP model. The purpose of pricing problem is to find variables (columns) with the negative reduced cost since only those can improve the objective value of the master problem.

We add the newly found column to the restricted master problem and resolve it, and then again construct the pricing problem. The loop continues until no variable with negative reduced cost is found. When there is no variable with negative reduced cost, we know that the solution to the master problem is optimal. The optimal solution can be found without even considering all the possible variables in the problem [18], and that is the reason why column generation can be much more efficient than the standard solving method. The pricing problem ensures that only the variables that have the potential to improve the objective value of the RMP can be added. The flowchart of the whole algorithm is in Figure 4.1

When combined with other methods such as branch-and-bound, column generation can become a powerful tool to solve mixed integer linear programming (MILP) problems, as M. Lübbecke states in his paper [18] *"...column generation is a real winner in the context of integer programming. This made the powerful method a must-have in the computational mixed integer programming bag of tricks."*. Another advantage of column generation is that it often gives much stronger bounds than the original LP relaxations [18].

The combined method is called branch-and-price. The general outline could be summarized as follows: In every node of the branch and bound tree, we perform column

generation and then continue branching until the solution is integral. As J. Desrosiers writes in his paper [19] *"When the linear relaxation in each node of a branch-and-bound tree is solved by column generation, one speaks of branch-and-price."*

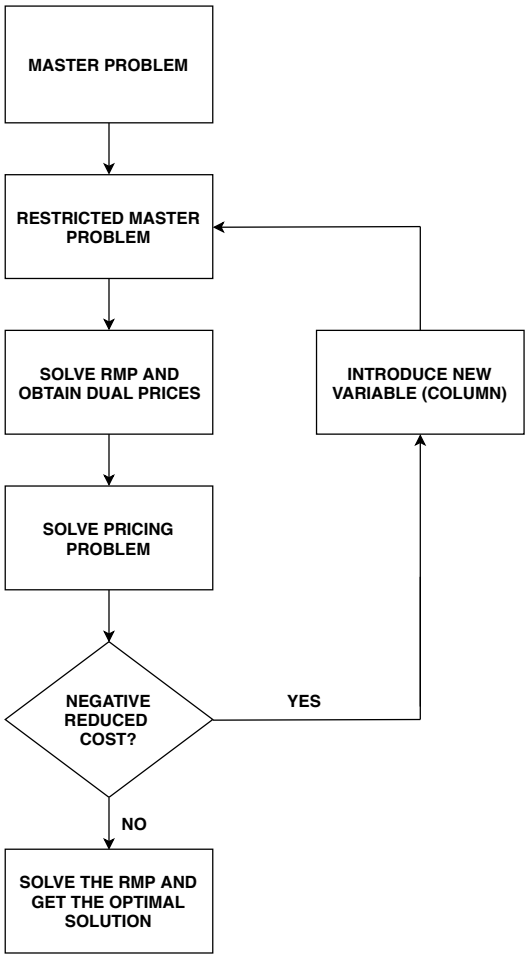

Figure 4.1. The column generation method flowchart.

4.1 Master problem

We formulated the master problem for one forklift with columns being the individual routes and with real-valued variables p_{k,τ_i} that decide how many times we use the route (column) k in period τ_i. A column is a vector of the form: $(t_k, u_{k,m_0}, u_{k,m_1}, u_{k,m_2}, \ldots, v_{k,m_1}, v_{k,m_2}, \ldots)$.

The t_k parameter is the time it takes to execute the route, the time is always the minimal one, and that is ensured by correctly generating the columns in the pricing problem. The parameters u_{k,m_i} say how much material the route delivers to machine

m_i and their indicators v_{k,m_i} indicate, that at least one unit of material was delivered by the column to machine m_i.

One important note about our formulation is the meaning of "machine" u_{k,m_0}, which represents the overall decrease in remaining battery capacity after executing the route (column) k. By formulating the battery decrease in this way, we can think about the battery as being a "virtual machine" that also needs to meet specific demands in every period. In particular, we need to ensure that the sum of all the battery contributions of all used routes for every period does not drop below zero or exceed the maximum battery capacity. Another implication of our formulation is that we can also use columns to represent charging of the forklift. We do it by setting all u_{k,m_i} to zero, then setting u_{k,m_0} to a negative value and since the values meaning is the decrease in the battery, we charge the forklift.

We need to be able to model the fact, that after every period, the amount of material the machine demanded for the period is consumed and any leftover material stays in the storage area of the machine. We considered two possibilities. The first is having a helper variable for every machine in every period equal to the sum of delivered material minus the demanded amount of material. If we choose this option, we can easily chain together the material states in every period for every machine, but the downside is that it complicates the formulation. The second possibility is to compute the cumulative demands of each machine in every period and modify the constraints accordingly. This formulation deals with the amount of leftover material implicitly, without the need of any additional variables. Our formulation uses the second option.

The master problem is formulated as follows:

Objective function:

$$min \quad \sum_{k \in K'} \sum_{\tau_i \in T} t_k \; p_{k,\tau_i} \tag{3}$$

Subject to:

$$\sum_{k \in K} \sum_{\tau_i \in \tau_i^P} u_{k,m_v} \; p_{k,\tau_i} \geq d_{\tau_i^P(-1),m_v} \quad \forall m_v \in M, \; \forall \tau_i^P \in T^P \tag{4}$$

$$\sum_{k \in K} \sum_{\tau_i \in \tau_i^P} v_{k,m_v} \; p_{k,\tau_i} \geq \left\lceil \frac{d_{\tau_i^P(-1),m_v}}{C} \right\rceil \quad \forall m_v \in M \setminus \{m_0\}, \; \forall \tau_i^P \in T^P \tag{5}$$

$$\sum_{k \in K} \sum_{\tau_i \in \tau_i^P} -u_{k,m_v} \; p_{k,\tau_i} \geq -(d_{\tau_i^P(-2),m_v} + W_{m_v}) \quad \forall m_v \in M, \; \forall \tau_i^P \in T^P \tag{6}$$

$$\sum_{k \in K} -t_k \; p_{k,\tau_i} \geq -3600 \quad \forall \tau_i \in T \tag{7}$$

Variables:

- $p_{k,\tau_i} \in \mathbb{R}_0^+ \ldots$ the number of times route k is executed in period τ_i

Parameters:

- $K \ldots$ set of routes
- $K' \ldots$ set of routes that satisfy $u_{k,m_0} > 0$
- $T \ldots$ set of periods
- $T^P \ldots$ set of sets τ_i^P
- $\tau_i^P \ldots$ set of periods from 1 to i, e.g. $\tau_3^P = \{\tau_1, \tau_2, \tau_3\}$
- $\tau_i \ldots$ period i
- $M \ldots$ set of machines $\{m_0, m_1, m_2 \ldots\}$, where m_0 represents the battery
- $u_{k,m_v} \in \mathbb{N}_0 \ldots$ amount of material unloaded at machine m_v in route k
- $v_{k,m_v} \in \{0, 1\} \ldots$ is equal to one if and only if the amount of material u_{k,m_v} is at least one
- $u_{k,m_0} \in \mathbb{R} \ldots$ amount of battery drained by route k
- $W_{m_v} \in \mathbb{N} \ldots$ storage capacity of machine m_v, for m_0 it is battery capacity B
- $d_{\tau_i,m_v} \in \mathbb{N}_0 \ldots$ cumulative demand of machine m_v in period τ_i, for m_0 it is always 0 (battery cannot get negative)
- $d_{\tau_i^P(-1),m_v} \in \mathbb{N}_0 \ldots$ cumulative demand of the last period in set τ_i^P
- $d_{\tau_i^P(-2),m_v} \in \mathbb{N}_0 \ldots$ cumulative demand of the second last period in set τ_i^P

The objective function minimizes the sum of traveling times of the forklift across all routes and all periods. If the solution does not use a route, its corresponding variable p_{k,τ_i} will be equal to zero. There is only a single forklift, but in the case of multiple forklifts the objective function would sum all the traveling times of all forklifts, that means having multiple forklifts does not have a significant impact on the objective value. We will discuss more on the effect of multiple forklifts on the solution in chapter 5.

The constraints (4) ensure that all cumulative demands of all machines are satisfied. We do it by defining the sets τ_i^P that include all periods up until the period i. We sum the material delivered to the machine m_v over all possible sets of τ_i^P, and we compare that to the cumulative demand of the last period in τ_i^P. The constraints (5) enforce the minimal number of visits to each machine that is needed to satisfy their demand. More discussion about the reasons behind adding the constraint to the model in Chapter 4.6. Constraints (6) are analogous to the first constraints, but instead of enforcing a minimum amount of material they make sure the material does not exceed the maximum storage capacity of the machine. The right-hand side of the inequality is equal to the amount of material that had to be delivered to the machine up until

the previous period $d_{\tau_i^P(-2),m_v}$ plus the maximum capacity W_{m_v}. The inequality is multiplied by minus one to make it a "greater than" inequality. Constraints (7) ensure that the sum of execution times of all routes in a period is less than one hour, the units of time are seconds, hence the -3600 on the right-hand side.

The master problem needs to be initialized with some set of routes (columns). The way we initialize our master problem is by solving the particular instance by the best achieving constructive heuristic and then converting the result to a set of columns. This set of columns is the initial input to the master problem. It is in our interest to initialize the column generation master problem with columns from the best solution we have since it can speed up the whole solution process.

4.1.1 Example

We present an example formulation for the master problem. The layout of the instance is on the Figure 4.2. The instance has three vertices, one of them being the station. The machine m_0 represents the "virtual machine" that is the battery. The number of periods is two. Vehicle capacity $C = 200$ and battery capacity $B = 100$.

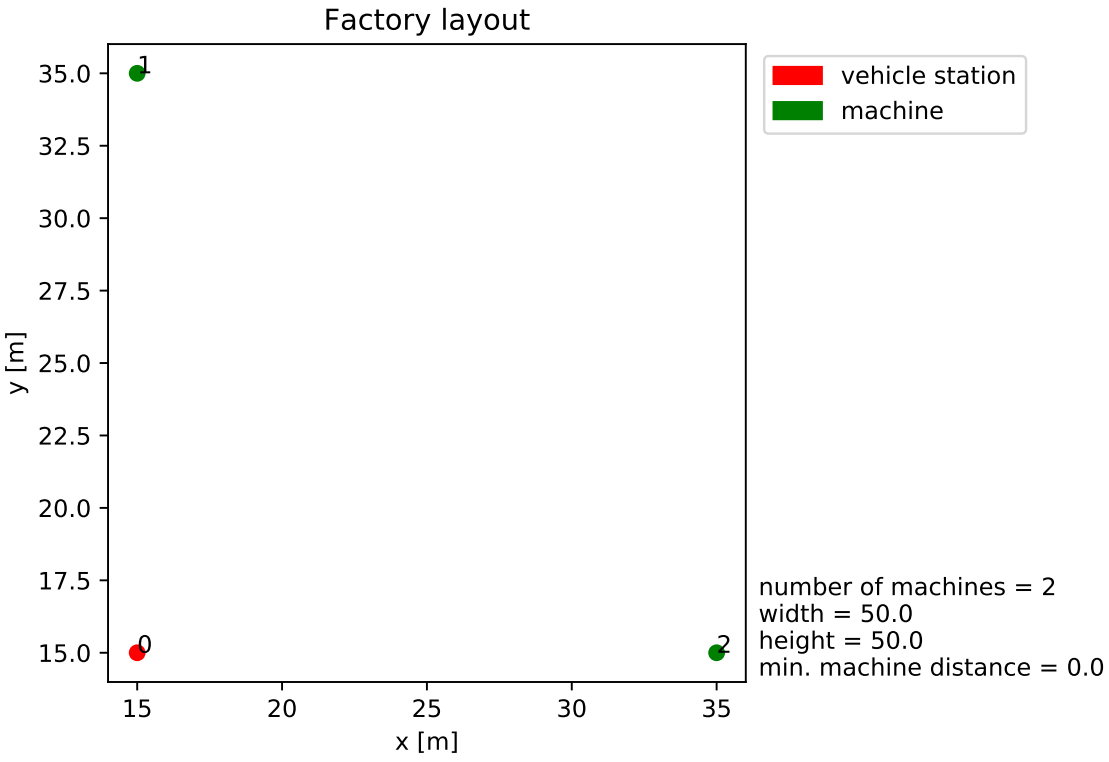

Figure 4.2. Layout of example instance.

The demands of the machines and battery m_0 are:

- $d_{\tau_1,m_0} = 0$, $d_{\tau_2,m_0} = 0$
- $d_{\tau_1,m_1} = 30$, $d_{\tau_2,m_1} = 60$
- $d_{\tau_1,m_2} = 70$, $d_{\tau_2,m_2} = 25$

The cumulative demands of the machines and battery m_0 are therefore:

- $d_{\tau_1,m_0} = 0$, $d_{\tau_2,m_0} = 0$
- $d_{\tau_1,m_1} = 30$, $d_{\tau_2,m_1} = 90$
- $d_{\tau_1,m_2} = 70$, $d_{\tau_2,m_2} = 95$

The machine capacity limits are:

- $W_{m_0} = B = 100$
- $W_{m_1} = 100$
- $W_{m_2} = 100$

The initial set of columns, including the charging column and an example form of the column, are:

- $(t_k, u_{k,m_0}, u_{k,m_1}, u_{k,m_2}, v_{k,m_1}, v_{k,m_2})$
- $(40, 4.44, 50, 0, 1, 0)$
- $(40, 4.44, 0, 50, 0, 1)$
- $(80, -8.88, 0, 0, 0, 0)$

The first two columns are direct deliveries to machines m_1 and m_2, because they are 20 meters away and the speed of the forklift is one meter per second, the time it takes to execute each of the columns is 40. The third column is the charging column, that is why the amount of battery drained is negative.

Objective function:

The objective function (3) looks as follows in our example instance. Notice, that only the routing columns are present in the objective function.

$$min \quad 4.44 \cdot p_{1,\tau_1} + 4.44 \cdot p_{1,\tau_2} + 4.44 \cdot p_{2,\tau_1} + 4.44 \cdot p_{2,\tau_2}$$

Subject to:

The constraints (4) from the master problem, the first two rows are for the machine m_0 (the battery), and the next four rows are for machines m_1 and m_2. The right-hand sides of the inequalities are the cumulative demands.

$$
\begin{aligned}
4.44 \cdot p_{1,\tau_1} + & & 4.44 \cdot p_{2,\tau_1} + & & -8.88 \cdot p_{3,\tau_1} & & & \geq 0 \\
4.44 \cdot p_{1,\tau_1} + & 4.44 \cdot p_{1,\tau_2} + & 4.44 \cdot p_{2,\tau_1} + & 4.44 \cdot p_{2,\tau_2} - & 8.88 \cdot p_{3,\tau_1} - & 8.88 \cdot p_{3,\tau_2} & & \geq 0 \\
50 \cdot p_{1,\tau_1} + & & 0 \cdot p_{2,\tau_1} + & & + 0 \cdot p_{3,\tau_1} & & & \geq 30 \\
50 \cdot p_{1,\tau_1} + & 50 \cdot p_{1,\tau_2} & + 0 \cdot p_{2,\tau_1} + & 0 \cdot p_{2,\tau_2} & + 0 \cdot p_{3,\tau_1} + & 0 \cdot p_{3,\tau_2} & & \geq 90 \\
0 \cdot p_{1,\tau_1} + & & 50 \cdot p_{2,\tau_1} + & & + 0 \cdot p_{3,\tau_1} & & & \geq 70 \\
0 \cdot p_{1,\tau_1} + & 0 \cdot p_{1,\tau_2} & + 50 \cdot p_{2,\tau_1} + & 50 \cdot p_{2,\tau_2} & + 0 \cdot p_{3,\tau_1} + & 0 \cdot p_{3,\tau_2} & & \geq 95
\end{aligned}
$$

The constraints (5) for ensuring a certain number of visits to every machine. We do not include m_0 since it does not make sense for the battery.

$$
\begin{aligned}
1 \cdot p_{1,\tau_1} + & & 0 \cdot p_{2,\tau_1} + & & +0 \cdot p_{3,\tau_1} & & & \geq 1 = \lceil \tfrac{30}{200} \rceil \\
1 \cdot p_{1,\tau_1} + & 1 \cdot p_{1,\tau_2} + & 0 \cdot p_{2,\tau_1} + & 0 \cdot p_{2,\tau_2} & + 0 \cdot p_{3,\tau_1} + & 0 \cdot p_{3,\tau_2} & & \geq 1 = \lceil \tfrac{90}{200} \rceil \\
0 \cdot p_{1,\tau_1} + & & 1 \cdot p_{2,\tau_1} + & & +0 \cdot p_{3,\tau_1} & & & \geq 1 = \lceil \tfrac{70}{200} \rceil \\
0 \cdot p_{1,\tau_1} + & 0 \cdot p_{1,\tau_2} + & 1 \cdot p_{2,\tau_1} + & 1 \cdot p_{2,\tau_2} & + 0 \cdot p_{3,\tau_1} + & 0 \cdot p_{3,\tau_2} & & \geq 1 = \lceil \tfrac{95}{200} \rceil
\end{aligned}
$$

The constraints (6) that ensure that the maximum storage capacity for every machine is not exceeded. Logically, the "maximum storage capacity" for the battery is equal to the battery capacity.

$$
\begin{aligned}
4.44 \cdot p_{1,\tau_1} + & & 4.44 \cdot p_{2,\tau_1} + & & -8.88 \cdot p_{3,\tau_1} & & & \leq 100 \\
4.44 \cdot p_{1,\tau_1} + & 4.44 \cdot p_{1,\tau_2} + & 4.44 \cdot p_{2,\tau_1} + & 4.44 \cdot p_{2,\tau_2} - & 8.88 \cdot p_{3,\tau_1} - & 8.88 \cdot p_{3,\tau_2} & & \leq 100 \\
50 \cdot p_{1,\tau_1} + & & 0 \cdot p_{2,\tau_1} + & & + 0 \cdot p_{3,\tau_1} & & & \leq 100 \\
50 \cdot p_{1,\tau_1} + & 50 \cdot p_{1,\tau_2} & + 0 \cdot p_{2,\tau_1} + & 0 \cdot p_{2,\tau_2} & + 0 \cdot p_{3,\tau_1} + & 0 \cdot p_{3,\tau_2} & & \leq 130 \\
0 \cdot p_{1,\tau_1} + & & 50 \cdot p_{2,\tau_1} + & & + 0 \cdot p_{3,\tau_1} & & & \leq 100 \\
0 \cdot p_{1,\tau_1} + & 0 \cdot p_{1,\tau_2} & + 50 \cdot p_{2,\tau_1} + & 50 \cdot p_{2,\tau_2} & + 0 \cdot p_{3,\tau_1} + & 0 \cdot p_{3,\tau_2} & & \leq 170
\end{aligned}
$$

The following are constraints (7) for our instance. The sum of all columns multiplied by the number of times they are used cannot exceed the time of one hour.

$$
\begin{aligned}
40 \cdot p_{1,\tau_1} + 40 \cdot p_{2,\tau_1} + 80 \cdot p_{3,\tau_1} \leq 3600 \\
40 \cdot p_{1,\tau_2} + 40 \cdot p_{2,\tau_2} + 80 \cdot p_{3,\tau_2} \leq 3600
\end{aligned}
$$

4.2 Dual problem

The next step is the formulation of the dual problem from our master problem. We need to get the dual master problem to formulate the pricing problem. After solving the master problem, we can obtain the dual prices of the constraints which correspond to the variables ψ, π, ϕ and γ in the dual. The meaning of these variables is described below. We will not describe the dual formulation further since we directly derived it from the master problem.

Objective function:

$$\begin{aligned} max \quad & \sum_{\tau_i^P \in T^P} \sum_{m_v \in M} d_{\tau_i^P(-1),m_v} \; \psi_{\tau_i^P,m_v} \\ & + \sum_{\tau_i^P \in T^P} \sum_{m_v \in M} -(d_{\tau_i^P(-2),m_v} + W_{m_v}) \; \pi_{\tau_i^P,m_v} \\ & + \sum_{\tau_i^P \in T^P} \sum_{m_v \in M} \left\lceil \frac{d_{\tau_i^P(-1),m_v}}{C} \right\rceil \phi_{\tau_i^P,m_v} \\ & -3600 \sum_{\tau_i \in T} \gamma_{\tau_i} \end{aligned} \tag{8}$$

Subject to:

$$\sum_{\tau_i^P \in T^P : \tau_i \in \tau_i^P} \sum_{m_v \in M} (u_{k,m_v} \; (\psi_{\tau_i^P,m_v} - \pi_{\tau_i^P,m_v}) + v_{k,m_v} \; \phi_{\tau_i^P,m_v}) - t_k \; \gamma_{\tau_i} \leq t_k \quad \forall k \in K, \; \forall \tau_i \in T \tag{9}$$

Variables:

- $\psi_{\tau_i^P,m_v}$ ... dual prices of constraints (4) from the Master problem
- $\pi_{\tau_i^P,m_v}$ ... dual prices of constraints (6) from the Master problem
- $\phi_{\tau_i^P,m_v}$ ... dual prices of constraints (5) from the Master problem
- γ_{τ_i} ... dual prices of constraints (7) from the Master problem

Parameters:

- $u_{k,m_v} \in \mathbb{N}_0$ ... amount of material unloaded at machine m_v in route k
- $u_{k,m_0} \in \mathbb{R}$ ... amount of battery drained by route k
- $t_k \in \mathbb{R}^+$ ... time needed to execute route k

ψ, π, ϕ and γ are variables in the dual, but when we solve the master problem and obtain the dual prices of each constraint, we get the optimal values of these dual variables. So when we introduce them to the pricing problem, they are not variables but parameters. We distinguish the variables from dual from parameters in the pricing problem with a hat above the letter, e.g., $\psi \rightarrow \hat{\psi}$.

4.3 Pricing problem - routing

The pricing problem is derived from the dual problem by taking the constraint (9) and formulating a maximization problem on the constraint. We know that in the master problem u_{k,m_v} were parameters, in the pricing problem, however, they are variables. The goal of pricing problem is to find the column that has the best objective value because that column is the one which can improve the solution from the master problem.

The pricing problem essentially solves a problem of choosing which machines to visit and how much material to deliver to them. It remotely resembles the traveling salesman problem. However another problem exists that is much closer to the model of our pricing problem, and that is the Orienteering Problem [20], we explain the similarities and differences from Orienteering Problem in Chapter 4.5.

Because of the structure of the master problem, we need to choose a period τ_i for which we want to find a new column. The existential quantification on k in the objective function (10) points out the fact we are finding the route k with the maximum objective value.

Objective function:

$$max \quad \sum_{m_v \in \widehat{M} \setminus \{m_s\}} (u_{k,m_v}\ \hat{\lambda}_{m_v} + v_{k,m_v}\ \hat{\phi}_{\tau_i^P,m_v}) - t_k(1+\hat{\gamma}_{\tau_i}) \quad \exists k \in K, for\ chosen\ \tau_i \in T \tag{10}$$

Subject to:

$$y^{in}_{m_s} = 1 \tag{11}$$

$$y^{out}_{m_s} = 1 \tag{12}$$

$$y^{in}_{m_i} = y^{out}_{m_i} \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{13}$$

$$y^{in}_{m_i} \leq 1 \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{14}$$

$$s_i - s_j + t_{i,j} \leq (BigM)(1 - x_{i,j}) \quad \forall m_i \in \widehat{M}, \forall m_j \in \widehat{M} \setminus \{m_s\} \tag{15}$$

$$s_s = 0 \tag{16}$$

$$s_i \geq 0 \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{17}$$

$$u_{k,m_i} \leq (BigM)\ y^{out}_{m_i} \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{18}$$

$$u_{k,m_i} \geq y^{out}_{m_i} \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{19}$$

$$v_{k,m_i} \leq (BigM)\ u_{k,m_i} \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{20}$$

$$(BigM)\ v_{k,m_i} \geq u_{k,m_i} \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{21}$$

$$u_{k,m_0} = \beta\ t_k \tag{22}$$

$$u_{k,m_0} \leq B \tag{23}$$

$$u_{k,m_i} \leq W_{m_v} \quad \forall m_i \in \widehat{M} \setminus \{m_s\} \tag{24}$$

$$\sum_{m_i \in \widehat{M} \setminus \{m_s\}} u_{k,m_i} \leq C \tag{25}$$

Variables:

- $x_{i,j} \in \{0,1\} \ldots$ indicates if machine m_i is visited just before machine m_j
- $s_i \in \mathbb{R}^+_0 \ldots$ the imaginary time of visit of machine m_i
- $u_{k,m_i} \in \mathbb{N}_0 \ldots$ amount of material delivered to m_i
- $u_{k,m_0} \in \mathbb{R} \ldots$ amount of battery drained by route k

Helper variables:

- $y^{in}_{m_i} = \sum_{m_j \in \widehat{M}} x_{j,i} \ldots$ sum of edges coming in machine m_i
- $y^{out}_{m_i} = \sum_{m_j \in \widehat{M}} x_{i,j} \ldots$ sum of edges coming out of machine m_i
- $t_k = \sum_{m_i, m_j \in \widehat{M}} x_{i,j}\ t_{i,j} \ldots$ time needed to execute the route

Parameters:

- $M \ldots$ set of machines $\{m_s, m_0, m_1, m_2 \ldots\}$, where m_0 represents battery and m_s is the forklift station
- $\widehat{M} = M \setminus \{m_0\}$
- $t_{i,j} \in \mathbb{R}^+ \ldots$ the time it takes to get from machine m_i to machine m_j
- $\hat{\lambda}_{m_v} = \sum_{\tau^P_i \in T^P : \tau_i \in \tau^P_i} (\hat{\psi}_{\tau^P_i, m_v} - \hat{\pi}_{\tau^P_i, m_v}) \ldots$ the score of machine m_v in given period τ_i
- $\hat{\phi}_{\tau^P_i, m_v} \ldots$ dual prices of constraints (5) from the Master problem
- $\hat{\gamma}_{\tau_i} \in \mathbb{R} \ldots$ dual prices of constraints (7) from the Master problem
- $C \in \mathbb{N} \ldots$ the capacity of forklift
- $B \in \mathbb{R}^+ \ldots$ the battery capacity of forklift
- $\beta \in \mathbb{R}^+ \ldots$ the discharge rate factor
- $BigM \ldots$ high valued constant used for turning off constraints and tying variables with their corresponding indicator variables

There are three helper variables that make the formulation more comprehensible. First, variables $y^{in}_{m_i}$ and $y^{out}_{m_i}$ that sum all of the incoming and outgoing edges from a

vertex (machine). Second, the variable t_k that sums all the weights of the edges along the path chosen by indicators $x_{i,j}$.

Constraints (11) and (12) force the route to start and end in the station m_s. Constraints (13) and (14) ensure that when a route enters a given machine, it will also leave the machine and that visiting all machines is not mandatory. Constraints (15), (16) and (17) are sub-tour elimination constraints. Constraints (18) and (19) ensure that if we visit a machine, at least one unit of material will be delivered to it and if we do not visit that machine, no amount of material can be added to that machine. A bit redundant constraints (20) and (21) that tie the indicator variables v_{k,m_i} to u_{k,m_i}. Both of those variables are present in the column returned to the master problem. Constraint (22) sets the "virtual machine" u_{k,m_0} that represents the battery drain to the correct value based on how long the selected tour takes. Constraint (23) puts an upper bound on the amount of battery that can be drained. Constraints (24) ensure that the column cannot deliver more material than the maximum storage capacity of the machine. Finally, constraint (25) ensures all the delivered material can fit into the forklift.

4.3.1 Example

We will show an example of columns generated by the pricing problem when solving the instance from Chapter 4.1.1. Only the routing columns are shown, the charging column is the same in every iteration of the master problem, and the pricing problem does not produce charging columns anyway, more on that fact in Chapter 4.4. We will show the result of the master followed by the newly generated column from the pricing problem. The column used in the master problem will have their corresponding value of variable p_{k,τ_i}. The columns are shown with their variable names to make the notation more comprehensible. By u_k vector we mean a vector of $(u_{k,m_0}, u_{k,m_1}, u_{k,m_2})$ and by v_k we mean (v_{k,m_1}, v_{k,m_2}).

Iteration 0 - master solution with initial columns that correspond to direct deliveries:

- period $= 0$, $p_{k,\tau_i} = 1.0$, $t_k = 40.0$, vector $u_k = (4.44, 0.0, 100.0)$, vector $v_k = (0, 1)$
- period $= 0$, $p_{k,\tau_i} = 1.0$, $t_k = 40.0$, vector $u_k = (4.44, 100.0, 0.0)$, vector $v_k = (1, 0)$

New column found by Pricing problem:

- $t_k = 60.0$, vector $u_k = (6.66, 1.0, 1.0)$, vector $v_k = (1, 1)$

Iteration 1 - master solution:

- period $= 0$, $p_{k,\tau_i} = 0.9494$, $t_k = 40.0$, vector $u_k = (4.44, 0.0, 100.0)$, vector $v_k = (0, 1)$
- period $= 0$, $p_{k,\tau_i} = 0.9494$, $t_k = 40.0$, vector $u_k = (4.44, 100.0, 0.0)$, vector $v_k = (1, 0)$

- period $= 0$, $p_{k,\tau_i} = 0.0505$, $t_k = 60.0$, vector $u_k = (6.66, 1.0, 1.0)$, vector $v_k = (1, 1)$

New column found by Pricing problem:

- $t_k = 60.0$, vector $u_k = (6.66, 1.0, 100.0)$, vector $v_k = (1, 1)$

Iteration 2 - master solution:

- period $= 0$, $p_{k,\tau_i} = 0.8989$, $t_k = 40.0$, vector $u_k = (4.44, 0.0, 100.0)$, vector $v_k = (0, 1)$
- period $= 0$, $p_{k,\tau_i} = 0.8989$, $t_k = 40.0$, vector $u_k = (4.44, 100.0, 0.0)$, vector $v_k = (1, 0)$
- period $= 0$, $p_{k,\tau_i} = 0.1010$, $t_k = 60.0$, vector $u_k = (6.66, 1.0, 100.0)$, vector $v_k = (1, 1)$

New column found by Pricing problem:

- $t_k = 60.0$, vector $u_k = (6.66, 100.0, 1.0)$, vector $v_k = (1, 1)$

Iteration 3 - master solution:

- period $= 0$, $p_{k,\tau_i} = 0.8484$, $t_k = 40.0$, vector $u_k = (4.44, 0.0, 100.0)$, vector $v_k = (0, 1)$
- period $= 0$, $p_{k,\tau_i} = 0.8484$, $t_k = 40.0$, vector $u_k = (4.44, 100.0, 0.0)$, vector $v_k = (1, 0)$
- period $= 0$, $p_{k,\tau_i} = 0.1010$, $t_k = 60.0$, vector $u_k = (6.66, 1.0, 100.0)$, vector $v_k = (1, 1)$
- period $= 0$, $p_{k,\tau_i} = 0.0505$, $t_k = 60.0$, vector $u_k = (6.66, 100.0, 1.0)$, vector $v_k = (1, 1)$

New column found by Pricing problem:

- $t_k = 60.0$, vector $u_k = (6.66, 100.0, 100.0)$, vector $v_k = (1, 1)$

Iteration 4 - master solution:

- period $= 0$, $p_{k,\tau_i} = 1.0$, $t_k = 60.0$, vector $u_k = (6.66, 100.0, 100.0)$, vector $v_k = (1, 1)$

No new columns found by the Pricing problem.

The fact that the final solution to the master problem is an integer one is a coincidence. The solution to the master problem does not have to be integer-valued. The master problem solutions in iteration 1, 2 and 3 show how a common result of the master problem looks like.

4.4 Pricing problem - charging

At first, it seems like there needs to be a separate problem for generating the charging columns, but after careful thought process we end up realizing it is not needed. The end goal is to have an integer-valued solution of p_{k,τ_i}. It does not make sense for the number of executions of a route to assume a fractional value. However, when the ”route” is the charging column, it makes sense, because the master problem can tune the length of

the charging session by changing its corresponding p_{k,τ_i} variable. To conclude, there are two types of columns, routing columns that drain the battery and deliver material to machines, and charging columns that only charge the battery. For a solution to have a physical interpretation, the first type of columns needs to be used an "integer number of times" but the second can be used "a real number of times".

4.5 Orienteering Problem

This section introduces the Orienteering Problem (OP), we study OP because of the similarity to our Pricing problem. The Orienteering Problem is a maximization problem on a graph. The objective is to maximize the score collected from the traversed vertices subject to a time limit, in which the route needs to be executed. We do not have to find a route that visits all the vertices in the graph. Consequently, the OP can be seen as a combination of the traveling salesman problem and knapsack problem [20]. The Orienteering Problem is also known as the maximum collection problem, selective traveling salesperson problem and the bank robber problem [20]. Other varieties such as the team orienteering problem and version with time windows are also studied.

We present a modified formulation of the OP from a paper *The orienteering problem: A survey* [20] by P. Vansteenwegen et al.

Objective function:

$$max \quad \sum_{i \in N} \sum_{j \in N} S_i \, x_{i,j} \tag{26}$$

Subject to:

$$\sum_{i \in N} \sum_{j \in N} t_{i,j} x_{i,j} \leq T_{max} \tag{27}$$

$$s_i - s_j + t_{i,j} \leq (BigM)(1 - x_{i,j}) \quad \forall i \in N, \forall j \in N \setminus 1 \tag{28}$$

$$y_1^{in} = 1 \tag{29}$$

$$y_1^{out} = 1 \tag{30}$$

$$y_i^{in} = y_i^{out} \quad \forall i \in N \tag{31}$$

$$y_i^{in} \leq 1 \quad \forall i \in N \tag{32}$$

Variables:

- $x_{i,j} \in \{0, 1\} \ldots$ indicates if vertex i is visited just before vertex j
- $y_i^{in} = \sum_{j \in N} x_{j,i} \ldots$ sum of edges coming in vertex i
- $y_i^{out} = \sum_{j \in N} x_{i,j} \ldots$ sum of edges coming out of vertex i

- $s_i \in \mathbb{R}^+ \ldots$ the imaginary time of visit of vertex i

Parameters:

- $N \ldots$ number of vertices
- $t_{i,j} \ldots$ the time it takes to travel between vertex i and j
- $T_{max} \ldots$ maximum time a route can take
- $S_i \ldots$ the score for traversing vertex i

As we have already stated, objective function (26) maximizes the total "score" of the selected route by summing S_i over the visited vertices. Constraint (27) is the limit on the route length. Constraints (28) eliminate sub-tours. Constraints (29) and (30) ensure the route starts and ends in the vertex number one. Constraints (31) ensure the continuity of the route, and finally, constraints (32) make the traversing of all vertices not mandatory.

The reason why we present the Orienteering Problem is the similarity to our pricing problem. If we omit $u_{k,m_v}\ \hat{\lambda}_{m_v}$ from the objective function (10) of the pricing problem, we end up with almost identical objective function as in OP. The only difference is that the constraint (27) is a hard constraint in the OP but only a soft constraint in the pricing problem in the form of $t_k(1 + \hat{\gamma}_{\tau_i})$ which is equivalent to $\sum_{m_i,m_j \in \widehat{M}} x_{i,j}\ t_{i,j}\ (1 + \hat{\gamma}_{\tau_i})$. Also, the only problem with the term $u_{k,m_v}\ \hat{\lambda}_{m_v}$ in the objective function of the pricing problem is that u_{k,m_v} is not a binary variable.

However, if we wanted the pricing problem to be as similar as it can be to the OP, we would have to omit the constraints (4) and (6) from the master problem. That would change the whole problem because the only enforced constraint would be on the minimal number of visits to a machine and the aspect of the amount of needed material would be entirely dismissed, which would make it very different from the initial problem.

4.6 Modifications and summary of the model

In this subsection, we will describe the pitfalls in developing the master problem formulation and the solutions to overcome them. It is critical to realize, that by modifying the master problem, we effectively change the dual and pricing problem as well because they are directly related and derived from each other.

One of the first master problem formulations did not have constraints (5), and pricing problem also did not have constraints (24). Consequently, the pricing problem did not have the term $v_{k,m_v}\ \hat{\phi}_{\tau_i^P,m_v}$ in the objective function. As a result, the pricing problem generated columns that delivered the maximum possible amount of material to one machine. The maximum possible amount was equal to the forklift capacity. After some

examination on why this happens, the reason is apparent. By excluding the constraints (24) from the pricing problem, there is only one upper limit on the amount of material that can be delivered to a machine, and that is the vehicle capacity C. The pricing model then simply chooses the machine with the highest value of $\hat{\lambda}_{m_v}$ and multiplies it by the highest value, which is C. So the pricing problem generated columns in the form:

$$(t_k, u_{k,m_0}, C,\ 0,\ 0, \ldots)$$
$$(t_k, u_{k,m_0}, 0,\ C,\ 0, \ldots)$$
$$(t_k, u_{k,m_0}, 0,\ 0,\ C, \ldots)$$

When such columns enter the master problem, they enable the solution to minimize the real-valued variables p_{k,τ_i} which in turn minimize the overall objective value. For example, if machine one needs 50 units of material, and the forklift capacity is 200, the pricing problem generates a column in the form $(t_k, u_{k,m_0},\ 200,\ 0,\ 0, \ldots)$ and then sets the p_{k,τ_i} to 0.25. In that way, this solution still satisfies constraints (4) because $0.25 \cdot 200 = 50$ and at the same time can minimize the corresponding term t_k 0.25 in the objective function.

The first modification to prevent this kind of undesirable behavior of the model is to introduce the constraints (24) into the pricing problem. That forces the pricing problem to generate more realistic columns. However, even with this change, the model still mostly generated columns that would deliver material to only one machine.

The second idea to force the model to generate more sensible columns that would deliver material to multiple machines is to change the constraints (4) to constraints that would enforce a certain integer-valued number of visits to each machine. The minimum number of visits to a machine that has a cumulative demand of $d_{\tau_i^P(-1),m_v}$ by a forklift with capacity C is $\left\lceil \frac{d_{\tau_i^P(-1),m_v}}{C} \right\rceil$. These constraints are exactly the constraints (5) from the master problem. However, if we only include those constraints and not constraints (4) and (6) we essentially end up with a type of *PVRP*. The solutions of this modified version are oblivious to the concrete demands of the machines, and as a consequence, it does not necessarily produce feasible solutions.

The final step is to merge the ideas and produce a master problem with both constraints (4), (6) and (5). Plus pricing problem with constraints (24). The result of that is the formulations presented in this chapter.

Unfortunately, even the final formulation does not give an integer result. It does not suffer from the described problems, but it still produces columns that are not very useful in an integer solution. Sometimes, it gives a set of columns in which we can create

a linear combination of two columns that will result in a column that is executable in the master problem. On the other hand, it gives us some form of recommendation on which machines to choose when executing a route. Also, the columns are more spread out and the problem of putting all the available forklift capacity into one machine is gone. Another valuable information we can get is the lower bound on the quality of any solution with one forklift. However, thanks to the objective function (3), it is also a lower bound on solutions with multiple forklifts.

Instead of trying to apply branch-and-price on the result of our column generation approach, we decided to take the resulting schedule and apply yet another model to acquire an integer solution. We call the model *Assignment model*, since it assigns the amounts of material that will be delivered to machines on already predetermined routes from the column generation method.

4.7 Assignment model

The assignment model operates on the output of column generation method. The column generation method gives us a lower bound on the quality of solutions on a particular instance but does not give us an executable schedule, since it does not give us an integer result. The idea behind the assignment model is to take the non-integer schedule produced by the column generation method and try to find a schedule that is executable.

We take all the routes scheduled by the column generation algorithm and fix them in time. That means we already have a predefined maximum number of routes in each period. Also, each route has already predefined set of vertices it has to visit, their permutation, and consequently, the battery drained and the time it takes to execute the route is already known too. The only thing that the model needs to decide is which routes will it use and if it uses a route, how much material will it deliver to each predefined machine it has to visit.

At first sight, the model is similar to the master problem, but there are major differences, such as u_{k,m_v} is a variable and not a parameter. Also notice, that u_{k,m_0} is a parameter because the routes are already determined from the input. In this model, the p_k variables of routing columns are binary (we either execute the route or not) but the p_k variables of charging columns are real-valued.

Finally, thanks to this model, we can improve the solution given by our constructive heuristic approach in Chapter 3. The improvement is measured and quantified in Chapter 5.

Objective function:

$$min \quad \sum_{k \in K'} t_k \, p_k \tag{33}$$

Subject to:

$$\sum_{k \in K_{\tau_i^P}} u_{k,m_v} \geq d_{\tau_i^P(-1),m_v} \quad \forall m_v \in M, \, \forall \tau_i^P \in T^P \tag{34}$$

$$\sum_{k \in K_{\tau_i^P}} u_{k,m_0} \, p_k \geq 0 \quad \forall \tau_i^P \in T^P \tag{35}$$

$$\sum_{k \in K_{\tau_i^P}} -u_{k,m_v} \geq -(d_{\tau_i^P(-2),m_v} + W_{m_v}) \quad \forall m_v \in M, \, \forall \tau_i^P \in T^P \tag{36}$$

$$\sum_{k \in K_{\tau_i^P}} -u_{k,m_0} \, p_k \geq -B \quad \forall \tau_i^P \in T^P \tag{37}$$

$$\sum_{k \in K_{\tau_i}} -t_k \, p_k \geq -3600 \quad \forall \tau_i \in T \tag{38}$$

$$\sum_{m_v \in M} u_{k,m_v} \leq C \quad \forall k \in K \tag{39}$$

$$u_{k,m_v} \leq (BigM) \, p_k \quad \forall k \in K, \, \forall m_v \in M \tag{40}$$

$$u_{k,m_v} \leq (BigM) \, v_{k,m_v} \quad \forall k \in K, \, \forall m_v \in M \tag{41}$$

$$(BigM) \, (1 - p_k) + u_{k,m_v} \geq v_{k,m_v} \quad \forall k \in K, \, \forall m_v \in M \tag{42}$$

Variables:

- $p_k \in \{0, 1\}, k \in K' \ldots$ indicates if route k is executed, only the routing columns have binary p_k
- $p_k \in \mathbb{R}^+, k \in K \setminus K' \ldots$ sets the length of charging of charging column p_k
- $u_{k,m_v} \in \mathbb{N}_0 \ldots$ amount of material unloaded at machine m_v in route k

Parameters:

- $K \ldots$ set of routes
- $K' \ldots$ set of routes that satisfy $u_{k,m_0} > 0$
- $K_{\tau_i} \ldots$ set of routes executed in period τ_i
- $K_{\tau_i^P} \ldots$ set of routes executed in periods $\{\tau_1, \tau_2, \ldots, \tau_i\}$
- $M \ldots$ set of machines $\{m_1, m_2 \ldots\}$, notice that both m_0 and m_s are excluded from the set, this is a change from the notation in the pricing problem
- $v_{k,m_v} \in \{0, 1\} \ldots$ is an indicator that forces the route k to visit machine m_v if set to one

- $u_{k,m_0} \in \mathbb{R}$ ... amount of battery drained by route k
- $t_k \in \mathbb{R}^+$ ... time needed to execute the route k

The objective function (33) minimizes the total time spent by the forklift traveling. Constraints (34) and (36) are very similar to the constraints (4) and (6) in the master problem. The only difference is the missing p_{k,τ_i} on the left-hand side of the inequalities. That is because variables p_k control the amount of material u_{k,m_v} through constraints (40). Constraints (35) and (37) are preventing the battery from dropping below zero or being above the allowed capacity B. As opposed to the master problem, they are written separately from the constraints on the machine demands (34) and (36). Constraints (38) serve the same purpose as in the master problem. Constraints (39) ensure the amount of material can fit into the forklift. Constraints (41) and (42) tie the indicator parameters v_{k,m_v} to the variables u_{k,m_v} and force the amounts of material u_{k,m_v} to be at least one if the route k is used.

To reiterate, the assignment model takes the result of the column generation method and solves an ILP model on it. The ILP model takes the routes, and by routes, we only mean the permutation of machines it visits, not including the material amounts. The goal of assignment model is to take those routes and decide how much material will be delivered in each of the routes to the machines it visits. The assignment also has the option to not use certain routes, provided that it can still reach a feasible solution even without using the routes. The result we get by solving the assignment model will always have a physical interpretation.

4.7.1 Example

To further clarify the procedure of the Assignment model, we present an example instance. The instance has four machines, one period and one forklift. The forklift capacity C is 100 units of material; the maximum machine capacities are also equal to 100. The layout of the factory is on Figure 4.3.

The demands of machines are as follows:

- $d_{\tau_1,m_1} = 60$
- $d_{\tau_1,m_2} = 40$
- $d_{\tau_1,m_3} = 70$
- $d_{\tau_1,m_4} = 30$

The column generation method will give us the following columns as a result. The p_{k,τ_i} variables are rounded to increase readability. We can see an interesting thing in the result. If we combine the first and fourth column, we can get another column that could have the p_{k,τ_i} set to one and deliver 60 units of material to the first machine and

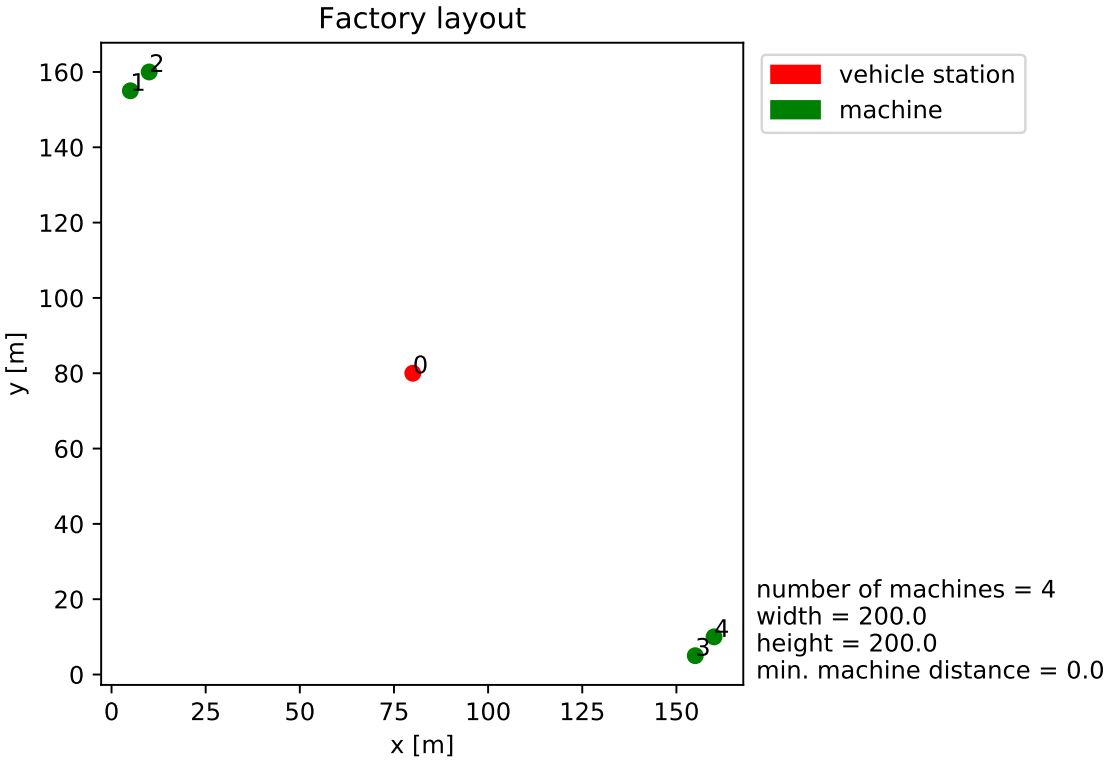

Figure 4.3. Layout of example instance for the Assignment model.

40 to the second machine. The same could be done for the second and third column. So the next step is to input these columns to the Assignment model.

- $p_{k,\tau_i} = 0.60$, $t_k = 219.43$, vector $u_k = (24.38\ , 99.0, 1.0, 0.0, 0.0)$, vector $v_k = (1, 1, 0, 0)$
- $p_{k,\tau_i} = 0.70$, $t_k = 219.43$, vector $u_k = (24.38\ , 0.0, 0.0, 99.0, 1.0)$, vector $v_k = (0, 0, 1, 1)$
- $p_{k,\tau_i} = 0.30$, $t_k = 219.43$, vector $u_k = (24.38\ , 0.0, 0.0, 1.0, 99.0)$, vector $v_k = (0, 0, 1, 1)$
- $p_{k,\tau_i} = 0.40$, $t_k = 219.43$, vector $u_k = (24.38\ , 1.0, 99.0, 0.0, 0.0)$, vector $v_k = (1, 1, 0, 0)$

When we solve the Assignment model with the previous columns as an input, we get the following set of columns as a solution. The solution has integer-valued variables p_{k,τ_i}. It correctly used only the first two columns and also set the material deliveries correctly to the demanded amount.

- $p_{k,\tau_i} = 1.0$, $t_k = 219.43$, vector $u_k = (24.38\ , 60.0, 40.0, 0.0, 0.0)$, vector $v_k = (1, 1, 0, 0)$
- $p_{k,\tau_i} = 1.0$, $t_k = 219.43$, vector $u_k = (24.38\ , 0.0, 0.0, 70.0, 30.0)$, vector $v_k = (0, 0, 1, 1)$
- $p_{k,\tau_i} = 0.0$, $t_k = 219.43$, vector $u_k = (24.38\ , 0.0, 0.0, 0.0, 0.0)$, vector $v_k = (0, 0, 1, 1)$
- $p_{k,\tau_i} = 0.0$, $t_k = 219.43$, vector $u_k = (24.38\ , 0.0, 0.0, 0.0, 0.0)$, vector $v_k = (1, 1, 0, 0)$

Chapter 5
Results

To analyze the quality of the developed strategies and column generation method, we created three separate sets of instances. The main reason for doing so is the computational time of the LP solver used for solving the column generation model. The column generation model requires solving a potentially massive number of linear programs in its cycle. Given the fact that we tested 20 different combinations of strategies from the constructive heuristics approach on a large number of instances with varying parameters, it is not practical to test the column generation approach on the same instances.

Instead, we measured the quality of solutions of the constructive heuristics approach, took the strategy that was the best on average, and a basic direct routing strategy and created another two sets of instances, which we solved with the two strategies and also with column generation, and assignment model. That gives us the lower bound, and potentially better objective value given by the assignment model. All time measurements were taken on a laptop with Intel Core i5-6200U 2.30 GHz processor, and 8 GB of RAM.

Every set of instances has a set of values for each chosen parameter. Then, we generate all possible combinations from these values. For every combination, we create ten different instances. When we evaluate the results, we take the mean objective value of the ten solutions and standard deviation.

The tables 5.1, 5.2 and 5.3 show the sets of parameters from which we generate every combination and create 10 random instances with those parameters. The coordinates of the machines and the machine station are chosen at random; the only given value is the total number of vertices (one more than the number of machines). Also, there is a defined interval in which the machines have their demand. This interval is between the *machine capacity upper limit* and *machine capacity lower limit* parameters. Other parameters are self-explanatory. Some of the parameters are not shown because they do not vary from instance to instance, those parameters are battery capacity, the charge rate of vehicles, the discharge rate of vehicles and the length of one period.

parameter	values			
vertex count	3	6	9	12
machine capacity upper limit	50			
vehicle capacity	50	100	150	
vehicle count	1	3	5	
machine capacity lower limit	5	50		
periods	1	5	10	

Table 5.1. Parameters of set 1.

parameter	values		
vertex count	6	12	
machine capacity upper limit	50		
vehicle capacity	150	300	
vehicle count	1		
machine capacity lower limit	5	50	
periods	1	5	10

Table 5.2. Parameters of set 2.

parameter	values		
vertex count	5	10	
machine capacity upper limit	50	100	
vehicle capacity	200	350	
vehicle count	1		
machine capacity lower limit	5	50	
periods	2	4	6

Table 5.3. Parameters of set 3.

5.1 Example demonstration

To demonstrate an interesting phenomenon, affecting the quality of solutions given by different strategies, we will show two simple problem instances. Both instances are almost identical; they have one period, only one forklift, four machines with demands $d_{\tau_1} = (100)$. The layout is in Figure 5.1. Please note, that the layout can be quite confusing because manhattan distance is used and not euclidean. The red vertex is the station, all the distances between the station and other machines are 10 meters, and distances between neighboring machines are 5 meters.

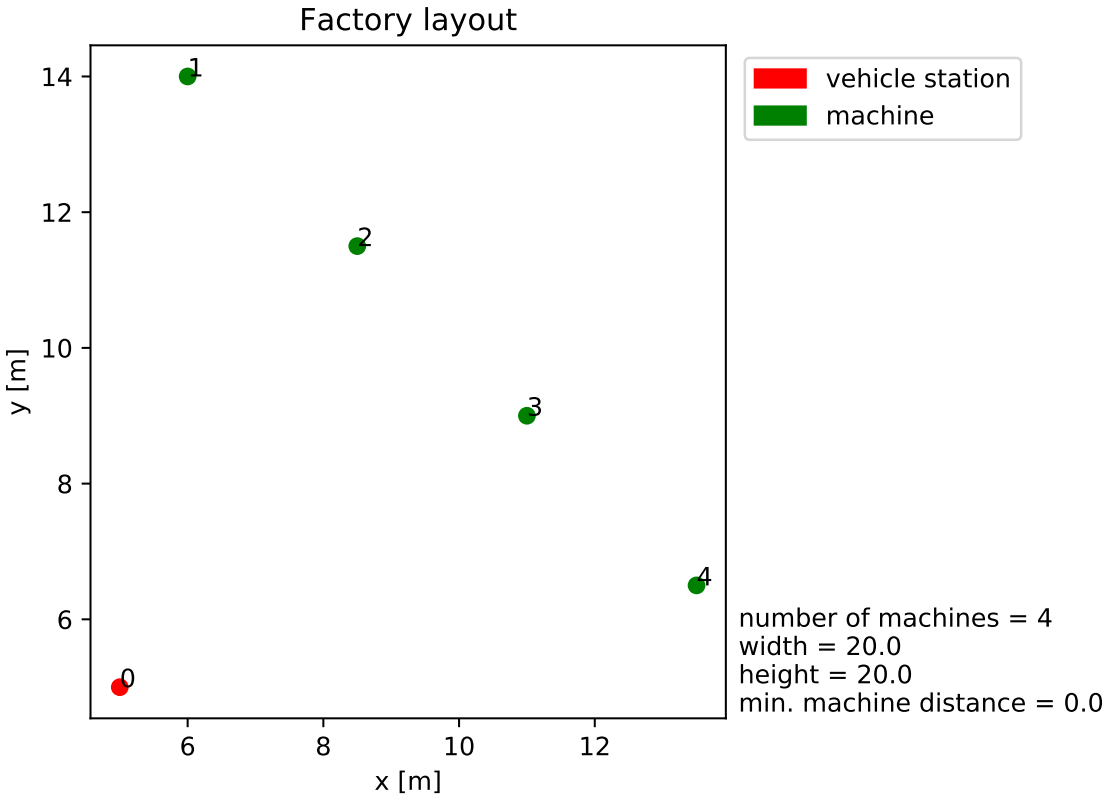

Figure 5.1. Layout of example instance.

Now consider two strategies, one can only perform direct deliveries and no tours, for example, the **Largest Demand Selection** strategy forces such behavior. The other strategy can only perform tours, and no direct deliveries, one strategy that would heavily favor tours is **Subset Selection** strategy, that prioritizes the largest subsets of machines available.

Lets set vehicle capacity $C = 100$ and see how the two strategies behave. The direct routing strategy will execute 4 routes, one to each machine, with each trip being 20 meters long. Each trip will deliver 100 units of material to the target machine. So the objective value of the direct routing strategy is 80.

The tour strategy, however, will be forced to execute 4 complete tours through all machines, each being 35 meters long. Each tour will drop 25 units of material to every machine. The objective value will be 140.

If we set the vehicle capacity $C = 400$, the direct routing strategy will still achieve the objective value of 80, because it will do the same thing as in the first instance. On the other hand, the tour strategy will execute only one trip, that will be 35 meters long, delivering all 100 units of material to every machine in that one trip. The objective value will, therefore, be only 35.

The point of this simple example is to demonstrate how the choice of parameters can dramatically impact the relative quality of solutions given by different strategies. The only thing that we did is change the vehicle capacity from 100 to 400; everything else is fixed. With only that change, suddenly one strategy gives much better results than the other. The effect of different machine capacity to vehicle capacity ratios on different strategies is shown in the following chapter.

5.2 Set 1

As we show in Section 5.1, the quality of solutions of a strategy can vary based on the parameters of the instance. Despite that fact, we want to asses the quality of each strategy as a whole on the entire set of different instances. We cannot directly compare the actual objective values those strategies achieve since the larger instances would skew the whole score in their favor.

To asses the overall score of all strategies used in a given set, we go through the following procedure. For every combination of input parameters of the instance set, we compute the mean objective value from the ten instances generated for the given combination. Then we sort the strategies according to the mean and get the one with minimal mean. Then, we compute by how much percent is each strategy worse than the strategy with minimal mean. Finally, the overall score of a strategy is the mean of the computed percentages. The resulting number can be interpreted as the mean percentual distance from the best-achieving strategy.

We employ the described technique for all three sets of instances. The tables 5.4, 5.13 and 5.18 show the statistic for each set.

On the first instance set, we only evaluated the constructive heuristic approach. We tested 20 different strategy combinations; those strategies are described in chapter 3.1. The overall score of all the combinations is in Table 5.4. The computational time is not included in the tables, because the running time of constructive heuristics is relatively small, generally within 1 second. Immediately, we can draw some conclusions looking at the table.

The best overall selection strategy is the ILP Selection strategy. We can see that the first two entries have the same exact score, that is because the ILP Selection strategy overrides the Filling strategy. As we have explained in chapter 3.1, ILP Selection is the only strategy that makes decisions on more than one level, and that is on the selection level and the filling level. The third and fourth place is also very close regarding the score, the strategies are effectively identical, but the randomness of the sorting strategy caused a little difference in the score.

The second observation is that when the Filling strategy has an effect (that is when it is not paired with ILP selection), the Even filling is overall always better than the Random filling. The third observation is that the Selection Pool Strategy does not have a big effect on the overall quality of the strategies. Lastly, the standard deviation of strategies is steadily increasing among the overall score, which solidifies the relative quality of the best strategies even more.

Selection Pool Strat.	Selection Strat.	Filling Strat.	mean [%]	stdev [%]
Demand Sort	ILP	Even	6.87	33.7
Demand Sort	ILP	Random	6.87	33.7
Random Sort	ILP	Random	6.90	33.5
Random Sort	ILP	Even	6.92	33.48
Demand Sort	Largest Demand	Even	43.03	47.88
Random Sort	Largest Demand	Even	43.03	47.88
Random Sort	K-Element Subset	Even	43.62	45.69
Demand Sort	K-Element Subset	Even	50.10	48.85
Random Sort	Random	Even	55.66	62.33
Demand Sort	Random	Even	55.87	63.35
Demand Sort	Subset	Even	63.50	78.18
Random Sort	Subset	Even	64.71	77.07
Demand Sort	Subset	Random	186.10	98.48
Random Sort	Subset	Random	187.71	96.37
Random Sort	Random	Random	212.67	104.27
Demand Sort	Random	Random	215.86	106.86
Demand Sort	K-Element Subset	Random	245.67	143.34
Random Sort	K-Element Subset	Random	254.20	147.80
Demand Sort	Largest Demand	Random	299.95	170.93
Random Sort	Largest Demand	Random	300.82	172.00

Table 5.4. Overall results on set one.

For the rest of this section, we will take the two best strategies and analyze the effect of different instance parameters on their solution quality. The selected strategies are **Demand Sort Selection Pool + ILP Selection + Even Filling** and **Demand Sort Selection Pool + Largest Demand Selection + Even Filling**, referred to as **ILP** and **LRG**. In every subsection, we present two different parameter configurations and show tables with the average objective value and standard deviation drawn from ten measurements. The units of the objective values are seconds.

5.2.1 Impact of vehicle count

The tables 5.5 and 5.6 show the mean objective values for different vehicles counts.

The parameters for instances in Table 5.5 are:

- vertex count = 12
- machine capacity upper limit = 50
- machine capacity lower limit: 50
- vehicle capacity = 150
- number of periods = 10

The parameters for instances in Table 5.6 are:

- vertex count = 6
- machine capacity upper limit = 50
- machine capacity lower limit: 50
- vehicle capacity = 150
- number of periods = 5

We can see that due to the objective function, having more vehicles does not necessarily improve the result. In fact, we can see that sometimes it can make the objective value even worse. This trend can be seen in Table 5.5 for the ILP strategy. On the other hand, the standard deviation of the objective value is very high, so it is hard to draw any definitive conclusions.

vehicle count	ILP mean	ILP stdev	LRG mean	LRG stdev
1	4254.85	880.12	6703.07	1688.03
3	4537.09	533.15	7319.77	1304.54
5	4717.20	442.40	7189.88	809.42

Table 5.5. Mean objective values for different vehicle counts with 10 periods and 12 vertices.

vehicle count	ILP mean	ILP stdev	LRG mean	LRG stdev
1	1194.69	156.86	1830.33	367.40
3	1014.88	224.52	1599.60	370.73
5	1187.47	208.16	1770.11	398.35

Table 5.6. Mean objective values for different vehicle counts with 5 periods and 6 vertices.

5.2.2 Impact of the ratio between vehicle capacity and machine capacity

The tables 5.7 and 5.8 show the mean objective values for different ratios of vehicle and machine capacity.

The parameters for instances in Table 5.7 are:

- vertex count = 12
- machine capacity upper limit = 50
- machine capacity lower limit: 50
- number of periods = 10
- vehicle count = 5

The parameters for instances in Table 5.8 are:

- vertex count = 6
- machine capacity upper limit = 50
- machine capacity lower limit: 50
- number of periods = 1
- vehicle count = 1

We can see the effect described in Section 5.1. So, when the vehicle capacity is the same as machine capacity, being able to visit multiple machines with one forklift in one trip does not help to improve the objective function. We can see that in both tables in the first row, ILP values are equal to LRG. However, as we see increase the forklift capacity, being able to deliver material to multiple machines can improve the objective value of ILP strategy. As opposed to the ILP strategy, the LRG strategy does not see a dramatic improvement when the vehicle capacity rises.

vehicle capacity	ILP mean	ILP stdev	LRG mean	LRG stdev
50	7494.88	1110.06	7494.88	1110.06
100	5663.03	996.30	7429.87	1660.92
150	4717.20	442.40	7189.88	809.42

Table 5.7. Mean objective values for different ratios of vehicle and machine capacity with 10 periods and 12 vertices.

5.2.3 Impact of the machine capacity range

The tables 5.9 and 5.10 show the mean objective values for different machine capacity ranges.

vehicle capacity	ILP mean	ILP stdev	LRG mean	LRG stdev
50	336.52	80.11	336.52	80.11
100	271.24	56.71	340.06	71.47
150	217.58	40.41	310.71	64.80

Table 5.8. Mean objective values for different ratios of vehicle and machine capacity with 1 period and 6 vertices.

The parameters for instances in Table 5.9 are:

- vertex count = 12
- machine capacity upper limit = 50
- vehicle capacity = 150
- number of periods = 1
- vehicle count = 5

The parameters for instances in Table 5.10 are:

- vertex count = 6
- machine capacity upper limit = 50
- vehicle capacity = 150
- number of periods = 10
- vehicle count = 5

Not surprisingly, if we allow the machines to have random demand in the range of 5 to 50 unit, the objective value is smaller than if we strictly set the demands to 50. In Table 5.9, the difference is not so huge, because the problem instances have only one period, but in Table 5.10 with ten periods, the difference is quite significant.

Also, in Table 5.9, the LRG strategy performs even better with a strict demand of 50 units. That is because in relatively small instances there is no difference between a machine with 25 units of demand and 50 units of demand when we can only execute direct deliveries. In both cases, the contribution to the objective value will be the same.

machine capacity lower limit	ILP mean	ILP stdev	LRG mean	LRG stdev
5	339.67	62.56	776.23	153.30
50	465.30	56.73	734.02	156.31

Table 5.9. Mean objective values for different machine capacity ranges.

machine capacity lower bound	ILP mean	ILP stdev	LRG mean	LRG stdev
5	1341.08	198.64	2407.07	631.84
50	2267.13	381.65	3418.28	700.97

Table 5.10. Mean objective values for different machine capacity ranges.

5.2.4 Impact of the number of periods

The tables 5.11 and 5.12 show the mean objective values for different number of periods.

The parameters for instances in Table 5.11 are:

- vertex count = 12
- machine capacity upper limit = 50
- machine capacity lower limit = 50
- vehicle capacity = 100
- vehicle count = 3

The parameters for instances in Table 5.12 are:

- vertex count = 9
- machine capacity upper limit = 50
- machine capacity lower limit = 50
- vehicle capacity = 150
- vehicle count = 1

The impact of the number of periods is quite obvious. The average objective value increases with the increase in periods. Both ILP and LRG strategies seem to drop in quality at a comparable rate.

number of periods	ILP mean	ILP stdev	LRG mean	LRG stdev
1	577.15	85.38	749.03	135.07
5	2944.26	365.04	3984.18	703.97
10	5438.51	981.48	6827.16	1426.93

Table 5.11. Mean objective values for different number of periods.

5.3 Set 2

The overall score of the column generation method, assignment model, and two selected constructive heuristic strategies for set 2 is shown in Table 5.13. The selected heuristic

number of periods	ILP mean	ILP stdev	LRG mean	LRG stdev
1	378.71	58.05	613.69	102.30
5	1757.07	313.51	2879.14	542.91
10	3620.06	348.45	5737.92	759.40

Table 5.12. Mean objective values for different number of periods.

strategies are the **Demand Sort Selection Pool + ILP Selection + Even Filling** and **Demand Sort Selection Pool + Largest Demand Selection + Even Filling** we will refer to them the same as in previous chapter, **ILP** and **LRG**.

Since the column generation approach serves as a lower bound and not an actual method of getting a feasible solution it is no surprise that it achieves the best objective values overall. We also see that the assignment model is approximately 5% better relative to the ILP strategy. However, it has bigger standard deviation which means that it does not have consistent results, which is undesirable. The last strategy is the LRG strategy, having objective value approximately 150% higher than the lower bound on average.

method	mean [%]	stdev [%]
column generation	0	0
assignment model	24.73	27.85
ILP Selection	29.46	18.64
Largest Demand Selection	166.19	61.45

Table 5.13. Overall results on set two.

In Table 5.14 we can see the percentage of times the Assignment model achieves better solutions than the ILP strategy for the set 2. We can also see the percentage in which the Assignment model gives the optimal solution, notice that its the same percentage the ILP strategy achieves the optimum and after examination, we discovered that the methods are optimal on the same instances.

Unfortunately, we have not been able to find a pattern in the instances of both sets 2 and 3 that would indicate when the Assignment model performs better than the ILP strategy. The results do not show clearly where the Assignment is better or worse.

5.3.1 Impact of the ratio between vehicle capacity and machine capacity

The parameters for instances in Table 5.15 are:

case	percentage of instances [%]
Assignment better than ILP	54.16
Assignment equal to ILP	16.66
Assignment worse than ILP	29.16
Assigment optimal	16.66
ILP optimal	16.66
LRG optimal	0.0

Table 5.14. Assignment model statistics for set 2.

- vertex count = 12
- machine capacity upper limit = 50
- machine capacity lower limit: 5
- number of periods = 5
- vehicle count = 1

In Table 5.15 we see that as vehicle capacity rises, the objective value of both Assignment model and ILP strategy get lower. In both cases their objective values are comparable. The abbreviation CG means column generation, and it is the computed lower bound of the instance. Another thing to point out is the comparison to the computed lower bound gets worse when the vehicle capacity rises.

vehicle cap.	Assig. mean	time [s]	ILP mean	time [s]	CG mean	time [s]
150	1492.65 ±219.19	0.06	1561.72 ±126.26	0.08	967.08 ±107.32	23.60
300	1156.34 ±188.01	0.08	955.11 ±90.04	0.05	646.67 ±63.12	75.59

Table 5.15. Mean objective values for different ratios of vehicle and machine capacity.

5.3.2 Impact of the machine capacity range

The parameters for instances in Table 5.16 are:

- vertex count = 12
- machine capacity upper limit = 50
- number of periods = 10
- vehicle count = 1
- vehicle capacity = 150

An interesting result can be seen in Table 5.16. When the lower limit of machine demands is 5, the objective values of Assignment and ILP are almost identical, and they are almost two times worse than the lower bound. However, when we increase

min. mach. cap.	Assig. mean	time [s]	ILP mean	time [s]	CG mean	time [s]
5	3014.83 ±515.99	0.17	3014.99 ±235.03	0.18	1885.61 ±341.19	40.22
50	3936.96 ±457.22	0.05	4964.63 ±400.07	0.20	3692.55 ±358.09	11.98

Table 5.16. Mean objective values for different machine capacity ranges.

the lower limit to 50, suddenly, the ILP strategy gets much worse, but overall, both methods are much closer to the lower bound.

5.3.3 Impact of the number of periods

The parameters for instances in Table 5.17 are:

- vertex count = 6
- machine capacity upper limit = 50
- machine capacity lower limit: 50
- vehicle count = 1
- vehicle capacity = 150

The Table 5.17 shows expected behavior. As the number of periods increases, both Assignment and ILP strategy increase in objective value. Both methods are in all cases relatively close to the lower bound, and the distance from the lower bound does not change much.

periods	Assig. mean	time [s]	ILP mean	time [s]	CG mean	time [s]
1	173.34 ±32.47	0.004	211.07 ±39.69	0.01	166.59 ±28.41	0.46
5	912.32 ±122.90	0.01	993.71 ±124.98	0.06	856.07 ±91.58	0.59
10	1884.34 ±553.31	0.02	2173.38 ±508.46	0.10	1761.21 ±449.20	0.84

Table 5.17. Mean objective values for different number of periods.

5.4 Set 3

The overall score of the solution methods for set 3 is in Table 5.18. Again, as in set 2, we can see the assignment model is 5% better relative to the ILP Selection strategy. The difference from the set 2 lies in the standard deviation of the assignment model score; it is almost the same as ILP Selection. The last strategy is again the LRG strategy, also with a very similar score of 150% higher than the lower bound on average.

Again, in Table 5.19 we can see the percentages for the set 3. The Assignment model is better than the ILP strategy in a lower percentage of instances than in the instances

method	mean [%]	stdev [%]
column generation	0	0
assignment model	34.54	19.96
ILP Selection	37.44	18.65
Largest Demand Selection	148.09	52.52

Table 5.18. Overall results on set three.

case	percentage of instances [%]
Assignment better than ILP	43.75
Assignment equal to ILP	18.75
Assignment worse than ILP	37.5
Assigment optimal	12.5
ILP optimal	12.5
LRG optimal	0.0

Table 5.19. Assignment model statistics for set 3.

of set 2. It is also the case for set 3, that when Assignment model achieves the optimum so does the ILP strategy.

5.4.1 Impact of the ratio between vehicle capacity and machine capacity

The parameters for instances in Table 5.20 are:

- vertex count = 10
- machine capacity lower limit = 50
- number of periods = 4
- vehicle count = 1
- vehicle capacity = 200

The Table 5.20 shows expected behavior. As the number of periods increases, both Assignment and ILP strategy increase in objective value. The lower bound does not change much with the increase of the maximum machine capacity.

max. mach. cap.	Assig. mean	time [s]	ILP mean	time [s]	CG mean	time [s]
50	1144.81 ±119.27	0.02	1436.53 ±205.74	0.07	1033.48 ±63.82	2.79
100	1562.34 ±309.18	0.02	1554.52 ±154.28	0.11	1083.34 ±151.96	1.68

Table 5.20. Mean objective values for different ratios of vehicle and machine capacity.

5.4.2 Impact of the machine capacity range

The parameters for instances in table 5.21 are:

- vertex count = 10
- machine capacity upper limit = 100
- number of periods = 4
- vehicle count = 1
- vehicle capacity = 200

We can see in Table 5.21 that the objective values of both methods are nearly the same for both lower limits of the machine capacity.

min. mach. cap.	Assig. mean	time [s]	ILP mean	time [s]	CG mean	time [s]
5	1286.68 ±186.16	0.02	1248.22 ±246.73	0.08	830.52 ±140.87	3.50
50	1562.34 ±309.18	0.03	1554.52 ±154.28	0.13	1083.34 ±151.96	1.85

Table 5.21. Mean objective values for different machine capacity ranges.

5.4.3 Impact of the number of periods

The parameters for instances in Table 5.22 are:

- vertex count = 5
- machine capacity upper limit = 50
- machine capacity lower limit: 5
- vehicle count = 1
- vehicle capacity = 200

Again as in the previous section, the results in Table 5.22 show little difference between the objective values of both methods. Obviously, the objective values increase as the number of periods increases. However, they are closer to the lower bound when the number of periods is lower.

periods	Assig. mean	time [s]	ILP mean	time [s]	CG mean	time [s]
2	254.40 ±63.22	0.007	254.40 ±63.22	0.02	184.56 ±28.35	0.11
4	507.21 ±146.32	0.02	502.02 ±148.07	0.04	346.74 ±95.10	0.44
6	730.29 ±147.95	0.02	717.71 ±157.75	0.07	505.80 ±105.39	0.65

Table 5.22. Mean objective values for different number of periods.

5.5 Discussion

The key findings of the measurements show, that when dealing with an instance of the problem where the capacity of the forklift is equal to all of the individual demands of the machines, the most basic solution strategy, the LRG, can suffice. However, when the vehicle capacity is, for example, double the size of the machine demands and above, the ILP strategy is the clear winner. Furthermore, we can also try to initialize the column generation algorithm with the output from the ILP strategy and then solve the Assignment problem on it to get an even better result in approximately 50% of all measured cases.

Chapter 6
Conclusion

In this thesis, we introduced new problem called Forklift Scheduling Problem (FSP). We studied similar problems found in literature, namely the Production Routing Problem, Inventory Routing Problem and Periodic Vehicle Routing Problem and pointed out key differences between them and the Forklift Scheduling Problem. We also proved that the FSP is $\mathcal{NP}$-hard.

We implemented a generator of instances for the FSP and proposed two algorithms to solve it. The first algorithm is based on constructive heuristic approach. The design of the algorithm is modular and allows us to change different parts of the algorithm with relative ease. The second algorithm is based on column generation method. The algorithm is able to find a lower bound on the objective value of a given instance and to improve the solutions given by the constructive heuristic algorithm.

Lastly, we evaluated all the developed algorithms on three sets of instances with different input parameters and found out the effects of the input parameters on the quality of solutions. The results show that the best overall quality of solutions from the constructive heuristic strategies is given by the strategy that uses an ILP model to select the machines. The result can be further improved by applying the column generation method along with another ILP model called the Assignment model. As a potential future work, we suggest testing different formulations of the problem including different speed modes of forklifts. Another possible future work is to apply the Branch and Price method.

Appendix A

Glossary

- AGV — Automated Guided Vehicle
- ALNS — Adaptive Large Neighborhood Search
- FSP — Forklift Scheduling Problem
- ILP — Integer Linear Programming
- IRP — Inventory Routing Problem
- LSP — Lot-Sizing Problem
- NP — Nondeterministic Polynomial Time complexity class of problems
- OP — Orienteering Problem
- P — Polynomial Time complexity class of problems
- PRP — Production Routing Problem
- PVRP — Periodic Vehicle Routing Problem
- SCM — Supply Chain Management
- TSP — Traveling Salesman Problem
- VMI — Vendor-managed Iinventory
- VRP — Vehicle Routing Problem

Appendix B

Contents of the attached CD

The attached CD contains the following directory structure:

```
* /attachments
   |-- /forklift_java
   |-- /forklift_python
   |-- /instances
   |       |---/set 1
   |       |---/set 2
   |       |---/set 3
   |-- /pdf
   |-- /time
```

The attachment includes projects implemented in java and python, the PDF file of this thesis, and the three sets of problem instances with their time measurements.

References

[1] Coelho, Leandro C., Jean-Francois Cordeau, and Gilbert Laporte. Thirty Years of Inventory Routing. *Transportation Science*. 2014, Vol. 48, No. 1, pp. 1-19. ISSN 0041-1655. Available from DOI 10.1287/trsc.2013.0472.
http://pubsonline.informs.org/doi/abs/10.1287/trsc.2013.0472.

[2] Karp, Richard M. Reducibility among Combinatorial Problems. *Complexity of Computer Computations*. Boston, MA: Springer US, 1972, pp. 85-103. Available from DOI 10.1007/978-1-4684-2001-2_9.
http://link.springer.com/10.1007/978-1-4684-2001-2_9.

[3] Mostafa, Noha A., and Amr B. Eltawil. The production-inventory-distribution-routing problem. *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*. IEEE, 2015, pp. 1-10. Available from DOI 10.1109/IEOM.2015.7093751.
http://ieeexplore.ieee.org/document/7093751/.

[4] Adulyasak, Yossiri, Jean-Francois Cordeau, and Raf Jans. The production routing problem. *Computers & Operations Research*. 2015, Vol. 55, pp. 141-152. ISSN 03050548. Available from DOI 10.1016/j.cor.2014.01.011.
http://linkinghub.elsevier.com/retrieve/pii/S0305054814000240.

[5] Jung, Sungwon, Tai-Woo Chang, Eoksu Sim, and Jinwoo Park. Vendor Managed Inventory and Its Effect in the Supply Chain. *Systems Modeling and Simulation: Theory and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 545-552. Available from DOI 10.1007/978-3-540-30585-9_61.
http://link.springer.com/10.1007/978-3-540-30585-9_61.

[6] Francis, Peter M., Karen R. Smilowitz, and Michal Tzur. The Period Vehicle Routing Problem and its Extensions. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Boston, MA: Springer US, 2008, pp. 73-102. Available from DOI 10.1007/978-0-387-77778-8_4.
http://link.springer.com/10.1007/978-0-387-77778-8_4.

[7] Adulyasak, Yossiri, Jean-Francois Cordeau, and Raf Jans. Optimization-Based Adaptive Large Neighborhood Search for the Production Routing Problem. Avail-

able from DOI 10.1287/trsc.1120.0443.
http://pubsonline.informs.org/doi/abs/10.1287/trsc.1120.0443.

[8] Glock, Christoph H., Eric H. Grosse, and Jörg M. Ries. The lot sizing problem. *International Journal of Production Economics*. 2014, Vol. 155, pp. 39-51. ISSN 09255273. Available from DOI 10.1016/j.ijpe.2013.12.009.
http://linkinghub.elsevier.com/retrieve/pii/S0925527313005689.

[9] Cordeau, Jean-Francois, Gilbert Laporte, Martin W.P. Savelsbergh, and Daniele Vigo. Chapter 6 Vehicle Routing. *Transportation*. Elsevier, 2007, pp. 367-428. Available from DOI 10.1016/S0927-0507(06)14006-2.
http://linkinghub.elsevier.com/retrieve/pii/S0927050706140062.

[10] Bell, Walter J., Louis M. Dalberto, Marshall L. Fisher, Arnold J. Greenfield, R. Jaikumar, Pradeep Kedia, Robert G. Mack, and Paul J. Prutzman. Improving the Distribution of Industrial Gases with an On-Line Computerized Routing and Scheduling Optimizer. *Interfaces*. 1983, Vol. 13, No. 6, pp. 4-23. ISSN 0092-2102. Available from DOI 10.1287/inte.13.6.4.
http://pubsonline.informs.org/doi/abs/10.1287/inte.13.6.4.

[11] Campbell, Ann Melissa, and Jill Hardin Wilson. Forty years of periodic vehicle routing. Available from DOI 10.1002/net.21527.
http://doi.wiley.com/10.1002/net.21527.

[12] Christofides, N., and J. E. Beasley. The period routing problem. Available from DOI 10.1002/net.3230140205.
http://doi.wiley.com/10.1002/net.3230140205.

[13] Giglio, Davide. Task scheduling for multiple forklift AGVs in distribution warehouses. *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1-6. Available from DOI 10.1109/ETFA.2014.7005360.
http://ieeexplore.ieee.org/document/7005360/.

[14] *The Traveling Salesman Problem with integer programming and Gurobi*.
http://examples.gurobi.com/traveling-salesman-problem/.

[15] Boyd, Stephen P., and Lieven. Vandenberghe. *Convex optimization*. New York: Cambridge University Press, 2004. ISBN 05-218-3378-7.

[16] *Linear programming*.
https://en.wikipedia.org/wiki/Linear_programming.

[17] Bertsimas, Dimitris., and John N. Tsitsiklis. *Introduction to linear optimization*. Belmont, Mass.: Athena Scientific, c1997. ISBN 18-865-2919-1.

[18] Lübbecke, Marco E. Column Generation. *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ, USA, 2010-06-15. Available from DOI 10.1002/9780470400531.eorms0158.
http://doi.wiley.com/10.1002/9780470400531.eorms0158.

[19] Desrosiers, Jacques, and Marco E. Lübbecke. Branch-Price-and-Cut Algorithms. *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ, USA, 2010-06-15. Available from DOI 10.1002/9780470400531.eorms0118.
http://doi.wiley.com/10.1002/9780470400531.eorms0118.

[20] Vansteenwegen, Pieter, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem. *European Journal of Operational Research*. 2011, Vol. 209, No. 1, pp. 1-10. ISSN 03772217. Available from DOI 10.1016/j.ejor.2010.03.045.
http://linkinghub.elsevier.com/retrieve/pii/S0377221710002973.