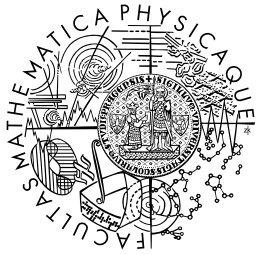

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER'S THESIS

Pavel Loub

Optimization of routing and scheduling for waste collection

Department of Probability and Mathematical Statistics

Supervisor of the master thesis:	Ing. Vít Procházka, Ph. D.
Study programme:	Mathematics
Study branch:	Probability, Mathematical Statistics and Econometrics

Prague 2023

I declare that I carried out this master's thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Hereby, I would like to thank my supervisor Ing. Vít Procházka, Ph. D., without whose practical advice, recommendations and, above all, his attitude this thesis would not have reached the quality it has now.

My thanks goes also to the team from Institute of Process Engineering at University of Technology, Brno, for their kindness and interest in my progress.

Finally, I want to show my gratitude to all of those who taught me and helped me to understand.

Title: Optimization of routing and scheduling for waste collection

Author: Pavel Loub

Institute: Department of Probability and Mathematical Statistics

Supervisor: Ing. Vít Procházka, Ph. D., Department of Probability and Mathematical Statistics

Abstract: This thesis proposes a mixed integer linear program for waste collection in South Moravian region. For deterministic models are difficult to solve by exact methods, it was necessary to find a way to obtain solutions in reasonable time. Hence, a brand new metaheuristic based on Simulated annealing is developed for solving the waste collection problem.

Keywords: Vehicle routing problem, simulated annealing, mixed integer linear programming

Contents

Notations and Abbreviations

To avoid any ambiguity it is useful to provide an overview of notations which will occur throughout the whole work. First of all let us start with sets. Sets will be denoted with capital letters of caligraphic font style, i. e. $\mathcal{B}$, $\mathcal{C}$, etc. Elements of of any set will be depicted using small letters of cursive font style, i. e. x, y, etc. In case of vectors, i. e. elements of any vector space, they will be represented as bold small letters of cursive font style, i. e. $\boldsymbol{x}, \boldsymbol{y}$, etc. Matrices will be always denoted as blackbord bold capital letters, i. e. $\mathbb{X}$, $\mathbb{M}$, etc. There is, on contrary, a stable blackboard bold notation of sets, e. g. natural numbers, or perhaps probability measure, but in this case we believe that there will be no misunderstanding if in the work will be any occurance of same denomination as in the list below describing set notations. Lastly, abbreviations will be always typeset with small caps font, i. e. *vehicle routing problem* will be abbreviated as VRP.

Set notations:

$\mathbb{N}$:	natural numbers
$\mathbb{Z}$:	integer numbers
$\mathbb{Q}$:	rational numbers
$\mathbb{R}$:	real numbers
$\emptyset$:	empty set
$\lvert\cdot\rvert$, $\lvert\lvert\cdot\rvert\rvert$ or $\mathrm{card}(\cdot)$:	cardinality of a set
$\mathcal{A}$:	σ-algebra
$\mathbb{P}$:	probability measure

Abbreviations:

IPE :	Institute of Process Engineering
LP :	linear program
MILP :	mixed integer linear program
SP :	stochastic programming
TSP :	Traveling Salesman Problem
SECs :	subtour elimination constraints
VRP :	Vehicle Routing Problem
MDVRPTW :	Multi-Depot Vehicle Routing Problem with Time Windows
MTPCARP :	Multi-Trip Periodic Capacitated Arc Routing Problem
NP :	non-deterministic polynomial time
HFCVRP :	Heterogeneous Fleet Capacitated Vehicle Routing Problem
LBPMDMFVRPTW :	Load Based Periodic Vehicle Routing Problem with Time Windows

Preface

Since George Dantzig and John Ramser in 1959 formulated a problem to optimize petrol deliveries, problem known under title *The Vehicle Routing Problem* (VRP), many researchers have been modifying VRP to get closer to the behavior of the world. VRP is a main puzzle of logistics upon which are humans dependent more and more – from food, goods or package delivery to bus or flight scheduling. On these pages we deal with another seizing of VRP, and that is optimization of waste collection in South Moravian region.

The problem has roots in *Institute of Process Engineering* (IPE) located at *Faculty of Mechanical Engineering* at *University of technology* in Brno. In South Moravia, few municipalities decided not to have waste collected by local services and equipped themselves with necessary instruments to take care of waste collection on their own. Since then there has been a development to obtain efficient schedulings by the members from IPE. In this work we want to offer the members an approach different from the one they propose. From the beginning we refused any specifics about their approach to not be affected by it. Any further consultations with the team from IPE were focused merely on the problem formulation and matters regarding the data we were provided.

On following pages we propose a mixed integer linear formulation for *load-based periodic multi-depot multi-fleet vehicle routing problem with time windows* (LBPMDMFVRPTW) and a metaheuristic algorithm based on simulated annealing, an algorithm inspired by a process of annealing in metallurgy, in which Kirkpatrick, Gellet and Vecchi with Černý saw an analogy thanks to which one is able obtain nearly or directly optimal solutions.

Both, the model and metaheuristic, are put in practice on real data afterwards and consequently is performed a comparative analysis of the solutions.

At the very end of the work we propose possible future extensions.

1. The Basics

Before we start on constructing a mathematical representation of South Moravian waste collection problem, it is necessary to lay down mathematical fundamentals to avoid any misunderstanding.

First concepts come from graph theory. For the beginning, let us start with the basics from graph theory, [1].

1.1 Graph theory

Definition 1. A *graph* is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of sets satisfying $\mathcal{E} \subseteq [V]^2$; thus the elements of $\mathcal{E}$ are 2-element subsets of $\mathcal{V}$. To avoid notational ambiguities, we shall always assume tacitly that $\mathcal{V} \cap \mathcal{E} = \emptyset$. The elements of $\mathcal{V}$ are the *vertices* (or *nodes*, or *points*) of the graph $\mathcal{G}$, the elements of $\mathcal{E}$ are its *edges* (or *lines*).

Definition 2. We say that a vertex $v \in \mathcal{V}$ is *incident* with an edge $e \in \mathcal{E}$ if $v \in e$. The two vertices incident with an edge are its *ends*. An edge $\{x, y\}$ is usually written as (x, y). Two vertices x, y of $\mathcal{G}$ are *adjacent* if (x, y) is an edge of $\mathcal{G}$, i. e. $(x, y) \in \mathcal{E}$. Two edges $e \neq f$ are *adjacent* if they have a vertex in common. If all the vertices of $\mathcal{G}$ are pairwise adjacent, then $\mathcal{G}$ is *complete*.

Remark. If a complete graph on n vertices is a K^n, then K^3 is a triangle.

Definition 3. The number of vertices of a graph $\mathcal{G}$ is its *order*, written as $|\mathcal{G}|$; its number of edges is denoted by $||\mathcal{G}||$. Graphs are *finite* or *infinite* according to their order; unless otherwise stated, the graphs we consider are all finite.

Definition 4. A *path* is a non-empty graph $P = (\mathcal{V}, \mathcal{E})$ of the form

$$\mathcal{V} = \{v_0, v_1, \dots, v_k\}, \qquad \mathcal{E} = \{(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)\},$$

where the v_i are all distinct. The vertices v_0 and v_k are linked by P and are called its *ends*; the vertices $v_1, \dots, v_{k-1}$ are the *inner* vertices of P. The number of edges in a path is its *length*, and the path of length k is denoted by P^k. Now that k is allowed to be zero; thus , $P^0 = K^1$.

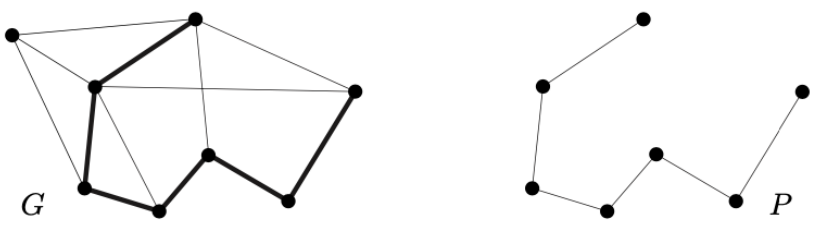

Figure 1.1: A path $P = P^6$ in $\mathcal{G}$.

Remark. We often refer to a path by the sequence of its vertices writing, e. g. $P = \{v_0, v_1, \dots, v_k\}$ and calling P a path from v_0 to v_k.

Definition 5. A *directed graph* (or *digraph*) is a pair $(\mathcal{V}, \mathcal{E})$, such that $\mathcal{V} \cap \mathcal{E} = \emptyset$, together with two maps $\mathsf{init}\colon \mathcal{E} \to \mathcal{V}$ and $\mathsf{ter}\colon \mathcal{E} \to \mathcal{V}$ assigning to every edge e an *initial vertex* $\mathsf{init}\,(e)$ and a *terminal vertex* $\mathsf{ter}\,(e)$. The edge e is said to be *directed from* $\mathsf{init}\,(e)$ *to* $\mathsf{ter}\,(e)$.

Remark. Note that a directed graph may have several edges between the same two vertices x, y. Such edges are called *multiple edges*. If $\mathsf{init}\,(e) = \mathsf{ter}\,(e)$, the edge e is called a *loop*.

In this thesis we also work with a term *clustering*. For it is not directly defined mathematically, let us provide a description of the term from [2]:

Clustering is division of data into groups of similar objects. Each group, called *a cluster*, consists of objects that are similar among themselves and dissimilar from other objects of other cluster. Proximity or distance measure is often used as the basis for clustering technique.

Remark. Consider set n-dimensional Euclidean space $\mathbb{R}^n$ and let $M = \{x_1, \dots, x_m\}$ be set of points where $x_i \in \mathbb{R}^n$, $i = 1, \dots, m$. Clustering is a process of partitioning the points into, let us say, k subsets according to some similarity of the points, e. g. minimal distance. Denote $C_1, \dots, C_k$ such that $C_j \subset \mathbb{R}^n$, $\forall\, j \in \{1, \dots, k\}$, then also holds that $C_j \cap C_\ell = \emptyset\ \forall j \neq \ell\ \wedge\ \bigcup_{j=1}^{k} C_j = M$.

1.2 Mathematical programming

In his section we introduce some of the fundamentals of mathematical programming. At first, we present basics from nonlinear programming based on Bazaraa et al., [3], as we want to stress out the idea behind stochastic programming (sp) and bazaraa provided succint but neat thought process from nonlinear program to stochastic one. And after that the subsequent subsection is devoted to special case of nonlinear programs where we assume linearity of all function, i. e. linear programming.

1.2.1 Nonlinear programming

Under the title *nonlinear program* we understand

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s. t.}\ \ g_j(\boldsymbol{x}) \leq 0 \qquad j = 1, ..., m \\ &\qquad h_k(\boldsymbol{x}) = 0 \qquad k = 1, ..., l \\ &\qquad\quad \boldsymbol{x} \in \mathcal{X}, \end{aligned} \tag{1.1}$$

where set $\mathcal{X} \subseteq \mathbb{R}^n$, functions f, g_j, $(j \in \{1, ..., m\})$, h_k, $(k \in \{1, ..., l\})$ are defined on $\mathbb{R}^n$, $n \in \mathbb{N}$, with values in $\mathbb{R}$. Function f is called *objective function*, each of the constraints $g_j(\boldsymbol{x}) \leq 0$ for every j is called an *inequality constraint*, and each of the constraints $h_k(\boldsymbol{x})$ for each k is called an *equality constraint*. Vector $\boldsymbol{x} \in \mathcal{X}$ fulfilling for every $j \in \{1, ..., m\}$ that $g_j(\boldsymbol{x}) \leq 0$ and for each $k \in \{1, ..., l\}$ that $h_k(\boldsymbol{x}) = 0$ is being declared as *feasible solution*. Set of such vectors $\boldsymbol{x}$ we call a *feasible region* and usually denote it by $\mathcal{M}$, i. e.

$$\mathcal{M} = \Big\{\boldsymbol{x} \in \mathcal{X} :\ g_j(\boldsymbol{x}) \leq 0,\ h_k(\boldsymbol{x}) = 0;\ j \in \{1, ..., m\},\ k \in \{1, ..., l\}\Big\}.$$

The nonlinear problem then is to find vector $\boldsymbol{x}^* \in \mathcal{M}$ such that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ for each feasible point $\boldsymbol{x}$. Such a point $\boldsymbol{x}^*$ is called an *optimal solution* to the problem. If there is more than one optimal solution, they are reffered to collectively as *alternative optimal solutions*.

We can notice that special case of (1.1) is a linear program in which functions f, g_j and h_k are linear and $\mathcal{X}$ is convex polyedric set. Such a program can be reformulated

$$\begin{aligned} &\min \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ &\text{s.t. } \mathbb{A}\boldsymbol{x} = \boldsymbol{b} \\ &\qquad \boldsymbol{x} \in \mathcal{X}, \end{aligned} \tag{1.2}$$

where $\boldsymbol{c} \in \mathbb{R}^n$, $\mathbb{A} \in \mathbb{R}^{(m+l)\times n}$ (compare dimension of $\mathbb{A}$ with dimensions of functions g_j and h_k) and $\boldsymbol{b} \in \mathbb{R}^{(m+l)}$. If we subtract $\boldsymbol{b}$ from both side of constraint $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$, we get $\mathbb{A}\boldsymbol{x} - \boldsymbol{b} = 0$. If $\mathbb{A}\boldsymbol{x} - \boldsymbol{b} = 0$ is a general system of $m + l$ equations and $\boldsymbol{x} \geq \boldsymbol{0}$, we obtain *linear program in standard form*. In a case when $\mathcal{X}$ is a subset of $\mathbb{R}^{n-k} \times \mathbb{Z}^k$, then we deal with *mixed integer linear program* or *mixed integer nonlinear program* depending on if functions f, g_j and h_k for each $j \in \{1, ..., m\}$ and for each $k \in \{1, ..., l\}$ are linear or not, respectively. More in the next subsection.

From (1.1) it is possible to obtain *parametric program* if functions f, g_j and h_k (for each j and k) are dependent also on some beforehand known parameter $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ is a subset of $\mathbb{R}^n$; the program can be formulated in a following way:

$$\begin{aligned} &\min f(\boldsymbol{x}, \boldsymbol{\lambda}) \\ &\text{s.t. } g_j(\boldsymbol{x}, \boldsymbol{\lambda}) \leq 0 \qquad j = 1, ..., m \\ &\qquad h_k(\boldsymbol{x}, \boldsymbol{\lambda}) = 0 \qquad k = 1, ..., l \\ &\qquad \boldsymbol{x} \in \mathcal{X}, \end{aligned} \tag{1.3}$$

where functions f, g_j, h_k are function defined on $\mathbb{R}^n \times \boldsymbol{\Lambda}$ with values in $\mathbb{R}$, for each $j \in \{1, ..., m\}$ and $k \in \{1, ..., l\}$.

Very useful extension of a *parametric program* is in the moment when the parameter $\boldsymbol{\lambda}$ is treated as a random vector. Let us therefore consider random vector $\boldsymbol{\xi} = (\xi_1, ..., \xi_p)^{\mathsf{T}}$ such that $\boldsymbol{\xi}(\omega) : \Omega \to \Theta$, where Ω is a *sample space* from probability space $(\Omega, \mathcal{F}, \mathbb{P})$ disposing with σ-algebra $\mathcal{F}$, and probability measure $\mathbb{P}$, and Θ is a subset of $\mathbb{R}^p$. Thus we get a program with a random parameter as follows:

$$\begin{aligned} &\min f(\boldsymbol{x}, \boldsymbol{\xi}) \\ &\text{s.t. } g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0 \qquad j = 1, ..., m \\ &\qquad h_k(\boldsymbol{x}, \boldsymbol{\xi}) = 0 \qquad k = 1, ..., \ell \\ &\qquad \boldsymbol{x} \in \mathcal{X}, \end{aligned} \tag{1.4}$$

where functions f, g_j, h_k are function defined on $\mathbb{R}^n \times \Theta$ with values in $\mathbb{R}$, for each $j \in \{1, ..., m\}$ and $k \in \{1, ..., l\}$. If we wanted to typify a set of feasible solutions for (1.4), then it would be obviously dependent – as analogously in the case

of the deterministic parameter program in (1.3) – on parameter $\boldsymbol{\xi}$ as it is formulated here below

$$\mathcal{M}(\boldsymbol{\xi}) = \Big\{\boldsymbol{x} \in \mathcal{X} :\ g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0,\ h_k(\boldsymbol{x}, \boldsymbol{\xi}) = 0;\ j \in \{1, ..., m\},\ k \in \{1, ..., l\}\Big\}.$$

A branch of mathematical programming dealing with problems involving uncertainty is called *stochastic programming*. There are many approaches to handle such problems from the simplest ones (replacing the uncertainty by its expected value) to, e. g. two-stage stochastic programming where, one in the first stage, makes so-called here-and-now decision, a decision based on the information known upto the present; the second stage describes our supposedly optimal behavior when the uncertainty is uncovered. Here we would like to describe a case when uncertainty is present only in constraints and not in the objective function.

Probability constraints

Let us work with a following program:

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s. t.}\ \ g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0 \qquad j = 1, ..., m \\ &\qquad\qquad \boldsymbol{x} \in \mathcal{X}. \end{aligned} \tag{1.4}$$

In this case we have few possible approaches to deal with the uncertainty in the constraints. As a solution can be considered a point which fulfills the constraints almost surely, i. e.

$$\mathcal{X}(\boldsymbol{\alpha}) = \Big\{\boldsymbol{x} \in \mathcal{X} : \mathbb{P}\Big(g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0;\ \forall\, j = \{1, ..., m\}\Big) = 1\Big\}. \tag{1.5}$$

Unfortunately, the problem here is that $\mathcal{X}(\boldsymbol{\alpha})$ is small or often times empty. Hence, there are also other approaches by which we can handle constraints with uncertainty.

The first one is called *joint probability constraints* formulated in Miller and Wagner, [4]. Consider a following set of feasible solutions:

$$\mathcal{X}_{\mathbf{J}}(\boldsymbol{\alpha}) = \Big\{\boldsymbol{x} \in \mathcal{X} : \mathbb{P}\Big(g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0;\ \forall\, j \in \{1, ..., m\}\Big) \geq \alpha\Big\}, \tag{1.6}$$

where $\alpha \in [0, 1]$. By this set we accept such solutions fulfilling for each $j \in \{1, ..., m\}$ the constraints with probability greater or equal to α, where α is chosen by the decision maker.

The other one is to view upon the probability constraints individually instead of jointly, i. e.

$$\mathcal{X}_{\mathbf{I}}(\boldsymbol{\alpha}) = \Big\{\boldsymbol{x} \in \mathcal{X} : \mathbb{P}\Big(g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0\Big) \geq \alpha_j;\ \forall\, j \in \{1, ..., m\}\Big\}, \tag{1.7}$$

where $\boldsymbol{\alpha} \in [0, 1]^m$. This approach is called *individual probability constraints* and it can be found in Charnes, Cooper and Symonds, [5]. A choice of parameter α, or α_j, $(j = 1, ..., m)$, is usually set to 0.99 or 0.95. Here, it is neccesary to remark that generally programs with permanently feasible constraints (1.5), with joint probability constraints (1.6) or with individual probability constraints (1.7) are not convex. To obtain wished convexity, we need to have additional assumptions (for further knowledge we encourage to see Prékopa, [6]).

1.2.2 Linear programming

Following on the Subsection 1.2.1, we want to dedicate this subection to fundamentals of linear programming, based on WOLSEY, [7], as a special case of non-linear programming.

Let us consider a linear program, in a manner of minimization, as a following formulation:

$$\begin{aligned} &\min \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ &\text{s.t. } \mathbb{A}\,\boldsymbol{x} \geq \boldsymbol{b} \\ &\qquad \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

where $\mathbb{A}$ is $m \times n$ matrix, $\boldsymbol{c}$ is n-dimensional vector, $\boldsymbol{b}$ is m-dimensional vector, and $\boldsymbol{x}$ is n-dimensional vector of variables or unknowns. Now let us add a restriction that specific variables must be integers. In a moment when only some of the variables have to be integer, we deal with *mixed integer program*, which can be written as

$$\begin{aligned} &\min \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{h}^{\mathsf{T}}\boldsymbol{y} \\ &\text{s.t. } \mathbb{A}\,\boldsymbol{x} + \mathbb{G}\boldsymbol{y} \geq \boldsymbol{b} \\ &\qquad \boldsymbol{x} \geq \boldsymbol{0} \\ &\qquad \boldsymbol{y} \in \mathbb{N}_0, \end{aligned}$$

where $\mathbb{A}$ is, again, $m \times n$ matrix, $\mathbb{G}$ is $m \times p$ matrix and $\boldsymbol{y}$ is p-dimensional vector of non-negative integer values. Assuming that all variables are integers we end up with *integer linear program* (ILP) written as

$$\begin{aligned} &\min \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ &\text{s.t. } \mathbb{A}\,\boldsymbol{x} \geq \boldsymbol{b} \\ &\qquad \boldsymbol{x} \in \mathbb{N}_0. \end{aligned}$$

One could also find a case when, on the other hand, all variables are binary. In this case we work with *binary integer program* (BIP) or *0-1 program* formulated as

$$\begin{aligned} &\min \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ &\text{s.t. } \mathbb{A}\,\boldsymbol{x} \geq \boldsymbol{b} \\ &\qquad \boldsymbol{x} \in \{0,1\}. \end{aligned}$$

To grasp any utilization, let us provide an example of a linear program entitled as *The Traveling Salesman Program* (TSP). As Wolsey writes in his book *Integer programming*: *"This is perhaps the most notorious problem in Operations Research because it is easy to explain, and so tempting to try and solve"* (page 7, [7]). The statement of the problem is following: a salesman must visit each of n cities exactly once; he starts from his home and it has to be also his ending spot.

The time to travel from city i to city j is denoted by t_{ij}. The assignment is to find the shortest path possible. Let us now denote by $\mathcal{C}$ the set of cities to visit and define a decision variable x_{ij} as follows:

$$x_{ij} = \begin{cases} 1: & \text{if the salesman travels directly from city } i \text{ to city } j;\ i,\ j \in \mathcal{C}, \\ 0: & \text{otherwise.} \end{cases}$$

For each $i \in \mathcal{C}$ we do not define x_{ii}. Let us continue with the definition of the constraints. To mathematically describe the fact that he leaves city i exactly once is by using binary variable x_{ij} formulated as

$$\sum_{j \in \mathcal{C}} x_{ij} = 1 \qquad \forall\, i \in \mathcal{C};\ i \neq j$$

and the reality that he also arrives to city j exactly once can be written by using x_{ij} as

$$\sum_{i \in \mathcal{C}} x_{ij} = 1 \qquad \forall\, j \in \mathcal{C};\ i \neq j.$$

Now, with these two constraints we could unfortunately obtain, although a feasible but not acceptable, solution of a form as displayed in Figure 1.2.

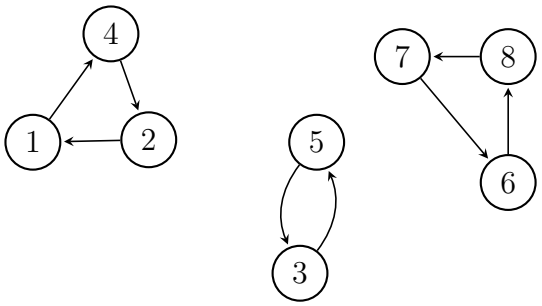

Figure 1.2: Subtours (a case when we consider $n = 8$).

To eliminate this type of solutions we need to extend the program by another set of constraints guaranteeing connectivity such that the salesman has to travel from one set of cities to another:

$$\sum_{i \in \mathcal{S}} \sum_{j \notin \mathcal{S}} x_{ij} \geq 1 \qquad \mathcal{S} \subset \mathcal{C};\ \mathcal{S} \neq \emptyset. \tag{1.1}$$

This type of constraints is called *cut-set* constraints. Nonetheless, constraints (1.1) can be replaced by so-called *subtour elimination constraints* (SECs) which are for TSP formulated in a following way:

$$\sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} x_{ij} \leq \mathrm{card}(\mathcal{S}) - 1 \qquad \mathcal{S} \subset \mathcal{C};\ 2 \leq \mathrm{card}(\mathcal{S}) \leq n - 1 \tag{1.2}$$

Consequently, the complete 0-1 LP formulation of TSP with SECs is

$$\begin{aligned}
&\min \sum_{i\in\mathcal{C}}\sum_{j\in\mathcal{C}} c_{ij}\, x_{ij} \\
&\text{s.t.} \sum_{j\in\mathcal{C}} x_{ij} = 1 && \forall\, i \in \mathcal{C};\ i \neq j \\
&\qquad \sum_{i\in\mathcal{C}} x_{ij} = 1 && \forall\, j \in \mathcal{C};\ i \neq j \\
&\qquad \sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{S}} x_{ij} \leq \operatorname{card}(\mathcal{S}) - 1 && \mathcal{S} \subset \mathcal{C};\ 2 \leq \operatorname{card}(\mathcal{S}) \leq n-1 \\
&\qquad\qquad x_{ij} \in \{0,1\} && \forall\, i,\, j \in \mathcal{C};\ i \neq j,
\end{aligned}$$

where in the objective function is the total travel time minimized.

We have proposed many formulations of nonlinear and linear programming but not any of possible solving approaches. Hence, let us introduce an algorithm often times used in optimization solvers, *simplex method*.

1.3 Primal simplex

In this section we explain the primal simplex algorithm based on *Linear Programming: Foundations and Extensions* by R. J. VANDERBEI (for further knowledge we refer to [8]).

Let us to start off by recalling a linear program, in terms of maximizing, i. e.

$$\begin{aligned}
&\max \sum_{j=1}^{n} c_j\, x_j \\
&\text{s.t.} \sum_{j} a_{ij}\, x_j \leq b_i \qquad i = 1, ..., m \\
&\qquad\qquad x_j \geq 0, \qquad j = 1, ..., n.
\end{aligned} \tag{1.3}$$

Further, we introduce so-called *slack variables* as follows:

$$x_{n+1} = b_i - \sum_{j=1}^{n} a_{ij}\, x_j \qquad i = 1, ..., m.$$

With these slack variables by reformulating (1.3) to a matrix form we obtain:

$$\begin{aligned}
&\max\ \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\
&\text{s.t.}\ \mathbb{A}\boldsymbol{x} = \boldsymbol{b} \\
&\qquad\ \boldsymbol{x} \geq \boldsymbol{0},
\end{aligned} \tag{1.4}$$

where

$$\mathbb{A} = \begin{pmatrix}
a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\
a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & & & \ddots & \\
a_{m1} & a_{m2} & \cdots & a_{mn} & 0 & 0 & \cdots & 1
\end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ x_{n+1} \\ \vdots \\ x_{n+m} \end{pmatrix}.$$

The simplex is an iterative procedure in which each iteration is characterized by specifying which m of the $n+m$ variables are basic. Therefore, denote by $\mathcal{B}$ the set of indices corresponding to the basic variables, and by $\mathcal{N}$ we denote a set of the nonbasic ones. In this manner, we can partition i-th component of $\mathbb{A}\boldsymbol{x}$ as

$$\sum_{j=1}^{n+m} a_{ij}\, x_j = \sum_{j \in \mathcal{B}} a_{ij}\, x_j + \sum_{j \in \mathcal{N}} a_{ij}\, x_j. \tag{1.5}$$

Now, we want to similarly express (1.5) in a matrix form, thus here we introduce by $\mathbb{B}$ a $m \times m$ matrix whose columns consist exactly of the m columns of matrix $\mathbb{A}$. Analogously, let denote by $\mathbb{N}$ a $m \times n$ matrix composed of the nonbasic columns of matrix $\mathbb{A}$. We notice that the denomination of matrix $\mathbb{N}$ and natural numbers is identical but we assure that in this section, i. e. Section 1.3, the denomination $\mathbb{N}$ will correspond only and only to the nonbasic part of matrix $\mathbb{A}$.

In this moment we can write matrix $\mathbb{A}$ in a partitioned-matrix form as follows:

$$\mathbb{A} = \begin{pmatrix} \mathbb{B} & \mathbb{N} \end{pmatrix}.$$

It is needed to highlight the fact that the matrix on the right side is not directly equal to matrix $\mathbb{A}$ for it is matrix $\mathbb{A}$ with its columns rearranged in a manner that all the columns associated with basic variables are listed first and followed by the nonbasic ones. Nevertheless, if we are consistent and rearrange the rows of $\boldsymbol{x}$ in the very same way, nothing can go wrong. As we partitioned matrix $\mathbb{A}$, we proceed similarly with $\boldsymbol{x}$ in a following manner:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_{\mathcal{B}} \\ \boldsymbol{x}_{\mathcal{N}} \end{pmatrix}.$$

Thus, we can formulate partitioning of $\mathbb{A}\boldsymbol{x}$ analogously as in (1.5) but with the recent denomination as a sum of basic and nonbasic parts written as

$$\mathbb{A}\boldsymbol{x} = \begin{pmatrix} \mathbb{B} & \mathbb{N} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_{\mathcal{B}} \\ \boldsymbol{x}_{\mathcal{N}} \end{pmatrix} = \mathbb{B}\,\boldsymbol{x}_{\mathcal{B}} + \mathbb{N}\,\boldsymbol{x}_{\mathcal{N}}.$$

Analogously, we can partition $\boldsymbol{c}$ as

$$\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} = \begin{pmatrix} \boldsymbol{c}_{\mathcal{B}} \\ \boldsymbol{c}_{\mathcal{N}} \end{pmatrix}^{\mathsf{T}} \begin{pmatrix} \boldsymbol{x}_{\mathcal{B}} \\ \boldsymbol{x}_{\mathcal{N}} \end{pmatrix} = \boldsymbol{c}_{\mathcal{B}}^{\mathsf{T}}\,\boldsymbol{x}_{\mathcal{B}} + \boldsymbol{c}_{\mathcal{N}}^{\mathsf{T}}\,\boldsymbol{x}_{\mathcal{N}}. \tag{1.6}$$

By this moment, when we are equipped sufficiently, we can proceed into the veins of the primal simplex method. The constraints $\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$ from (1.4) can be – by partitioning into basic and nonbasic parts – written as

$$\mathbb{B}\,\boldsymbol{x}_{\mathcal{B}} + \mathbb{N}\,\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{b}. \tag{1.7}$$

Since the variable $\boldsymbol{x}_{\mathcal{B}}$ can be formulated as a function of nonbasic variable $\boldsymbol{x}_{\mathcal{N}}$, it is the equivalent to the fact that matrix $\mathbb{B}$ is invertible. By *invertibility* we mean that the m columns of matrix $\mathbb{B}$ are linearly independent and thus form a basis for $\mathbb{R}^m$. If we multiply, from left, both sides in (1.7) by $\mathbb{B}^{-1}$ we get

$$\boldsymbol{x}_{\mathcal{B}} = \mathbb{B}^{-1}\boldsymbol{b} - \mathbb{B}^{-1}\mathbb{N}\,\boldsymbol{x}_{\mathcal{N}}. \tag{1.8}$$

In (1.8) we have obtained an expression for $\boldsymbol{x}_{\mathcal{B}}$. Now, by combining (1.6) and (1.8) we acquire

$$\begin{aligned}\zeta &= \boldsymbol{c}_{\mathcal{B}}^{\mathsf{T}}\,\boldsymbol{x}_{\mathcal{B}} + \boldsymbol{c}_{\mathcal{N}}^{\mathsf{T}}\,\boldsymbol{x}_{\mathcal{N}} \\ &= \boldsymbol{c}_{\mathcal{B}}^{\mathsf{T}}\Big(\mathbb{B}^{-1}\boldsymbol{b} - \mathbb{B}^{-1}\mathbb{N}\,\boldsymbol{x}_{\mathcal{N}}\Big) + \boldsymbol{c}_{\mathcal{N}}^{\mathsf{T}}\,\boldsymbol{x}_{\mathcal{N}} \\ &= \boldsymbol{c}_{\mathcal{B}}^{\mathsf{T}}\mathbb{B}^{-1}\boldsymbol{b} - \Big((\mathbb{B}^{-1}\mathbb{N})^{\mathsf{T}}\boldsymbol{c}_{\mathcal{B}} - \boldsymbol{c}_{\mathcal{N}}\Big)^{\mathsf{T}}\boldsymbol{x}_{\mathcal{N}}.\end{aligned} \tag{1.9}$$

Summarizing (1.9) and (1.8) we have

$$\begin{aligned}\zeta &= \boldsymbol{c}_{\mathcal{B}}^{\mathsf{T}}\mathbb{B}^{-1}\boldsymbol{b} - \Big((\mathbb{B}^{-1}\mathbb{N})^{\mathsf{T}}\boldsymbol{c}_{\mathcal{B}} - \boldsymbol{c}_{\mathcal{N}}\Big)^{\mathsf{T}}\boldsymbol{x}_{\mathcal{N}}, \\ \boldsymbol{x}_{\mathcal{B}} &= \mathbb{B}^{-1}\boldsymbol{b} - \mathbb{B}^{-1}\mathbb{N}\,\boldsymbol{x}_{\mathcal{N}}.\end{aligned} \tag{1.10}$$

For simplifying the expression let us propose following substitutions:

$$\zeta^* = \boldsymbol{c}_{\mathcal{B}}^{\mathsf{T}}\,\mathbb{B}^{-1}\boldsymbol{b}, \tag{1.11}$$

$$\boldsymbol{z}_{\mathcal{N}}^* = (\mathbb{B}^{-1}\mathbb{N})^{\mathsf{T}}\boldsymbol{c}_{\mathcal{B}} - \boldsymbol{c}_{\mathcal{N}}, \tag{1.12}$$

$$\boldsymbol{x}_{\mathcal{B}}^* = \mathbb{B}^{-1}\boldsymbol{b}. \tag{1.13}$$

Therefore we can write (1.10) succintly as

$$\begin{aligned}\zeta &= \zeta^* - \boldsymbol{z}_{\mathcal{N}}^{*\mathsf{T}}\boldsymbol{x}_{\mathcal{N}}, \\ \boldsymbol{x}_{\mathcal{B}} &= \boldsymbol{x}_{\mathcal{B}}^* - \mathbb{B}^{-1}\mathbb{N}\,\boldsymbol{x}_{\mathcal{N}}.\end{aligned} \tag{1.14}$$

The basic solution is obtained from (1.14) by setting $\boldsymbol{x}_{\mathcal{N}}$ to zero and therefore

$$\begin{aligned}\boldsymbol{x}_{\mathcal{N}}^* &= \mathbf{0} \\ \boldsymbol{x}_{\mathcal{B}}^* &= \mathbb{B}^{-1}\boldsymbol{b}.\end{aligned}$$

The primal simplex method is possible to describe as follows. Assume that we are given

1. a partition of the $n + m$ indices into a set of the basic indices denoted by $\mathcal{B}$ and a set $\mathcal{N}$ of the nonbasic indices such that the basis matrix $\mathbb{B}$ is invertible, and

2. an associated current primal solution $\boldsymbol{x}_{\mathcal{B}}^* \geq 0$ and $\boldsymbol{x}_{\mathcal{N}}^* = 0$.

The simplex method performs a sequence of steps to *adjacent* bases such that the current value ζ^* of the objective function ζ – in terms of a linear program in which we maximize the objective function – at each step increases while updating $\boldsymbol{x}_{\mathcal{B}}^*$ and $\boldsymbol{z}_{\mathcal{N}}^*$. *Adjacency* of two bases is whether they differ in one index only. Before we describe the simplex method in detail, it is necessary to provide conditions for feasibility of the solution as well as for its optimality. The feasibility condition is satisfied when $\boldsymbol{x}_{\mathcal{B}}^* \geq 0$ and optimality condition is fulfilled in case when $\boldsymbol{z}_{\mathcal{N}}^* \geq 0$ (in terms of maximizing objective value ζ; if we would be minimizing ζ, then the optimality condition would be $\boldsymbol{z}_{\mathcal{N}}^* \leq 0$).

STEP 1: If $\boldsymbol{z}^*_{\mathcal{N}} \geq \mathbf{0}$, then stop for the current solution is optimal. Otherwise proceed to STEP 2.

STEP 2: Pick a nonbasic index $j \in \mathcal{N}$ such that $z^*_j < 0$. To the index corresponding x_j is the *entering variable*.

STEP 3: Since we have selected the entering variable we want to increase its value from zero by letting

$$\boldsymbol{x}_{\mathcal{N}} = (0, ..., 0, t, 0, ..., 0)^{\mathsf{T}} = t\,\boldsymbol{e}_j,$$

where $\boldsymbol{e}_j$ represents the unit vector where any of its component is equal to zero but the j-th equals to one. From (1.14) we get

$$\boldsymbol{x}_{\mathcal{B}} = \boldsymbol{x}^*_{\mathcal{B}} - \mathbb{B}^{-1}\mathbb{N}\,t\boldsymbol{e}_j.$$

Therefore the step direction $\Delta\boldsymbol{x}_{\mathcal{B}}$ for the basic variables is given by

$$\Delta\boldsymbol{x}_{\mathcal{B}} = \mathbb{B}^{-1}\mathbb{N}\,\boldsymbol{e}_j.$$

STEP 4: Now we want to compute primal step length. We reach that by picking the largest $t \geq 0$ for every component of $\boldsymbol{x}_{\mathcal{B}}$ is nonnegative, i. e.

$$\boldsymbol{x}^*_{\mathcal{B}} \geq t\,\Delta\boldsymbol{x}_{\mathcal{B}}.$$

Since for every i holds that $x^*_i \geq 0$ and $t \geq 0$, let us divide both sides of the inequality. Here, it is necessary to highlight that the convention for $0/0$ is to set such ratios to zero. Hence, we get

$$\frac{1}{t} \geq \frac{\Delta x_i}{x^*_i} \qquad \text{for every } i \in \mathcal{B}.$$

If we want to get the largest t, in terms of its multiplication inverse we search for the smallest possible value of $1/t$ that fulfills all of the required inequalities, and that is

$$\frac{1}{t} = \max_{i\in\mathcal{B}} \frac{\Delta x_i}{x^*_i}.$$

The largest t for which all of the inequalities are satisfied is thus given by

$$t = \left(\max_{i\in\mathcal{B}} \frac{\Delta x_i}{x^*_i}\right)^{-1}.$$

If the maximum is less than or equal to zero, the problem is unbounded and we may stop here.

STEP 5: Select the *leaving variable* by choosing an arbitrary x_i, $i \in \mathcal{B}$, for which we obtain the largest t.

STEP 6: Update current solution by

$$x^*_j \leftarrow t$$
$$\boldsymbol{x}^*_{\mathcal{B}} \leftarrow \boldsymbol{x}^*_{\mathcal{B}} - t\,\Delta\boldsymbol{x}_{\mathcal{B}}.$$

STEP 7: Update the basis:

$$\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$$

In this section we have introduced the primal simplex method which is widely used in optimization solvers such as, e. g. GUROBI. Further, CPLEX is a software package for optimization which is based on simplex method (primal or dual) and can be also used on quadratic linear programs since it also uses so-called *interior-point method* (we refer to [8]).

Now, we are equipped with necessary theory to proceed further. Let us start with introducing, in logistics, well-known problem – *The Vehicle Routing Problem.*

2. Vehicle Routing Problem

2.1 Fundamentals

The *Vehicle Routing Problem* (VRP) can be described as a search of optimal paths (or routes) for provision of goods or services in distribution systems. This problem is a generalization of The *Traveling Salesman Problem* (TSP) as we introduced in the Section 1.2.2. VRP then is a problem of more than just one salesman. The problem is a main deal in logistics of distribution and can be found with many varieties according to the specifics of the real world situations, e. g. inclusion of time windows, meaning for example time intervals of the day during which a customer must be served (VRP with time windows; VRPTW), different type of vehicles where each vehicle has a different capacity or usage (Capacitated VRP with heterogeneous fleet; HFCVRP), [9].

Typically is road network described through a *graph* whose *vertices* characterize customers and depots, and *edges* (or *arcs*) depict links between the vertices. The graph can be *directed* or *undirected* according to the fact if one-way streets are taken into account or not, respectively. Each edge is associated with a *cost*, mostly represented as a *length*, and *travel time* [9].

2.2 Multi-depot Vehicle Routing Problem with Time Windows and Heterogeneous Fleet

For a beginning it is convenient to introduce the problem on a simple model just to grasp the idea from which we can proceed into more complex ones. Therefore let us consider *Multi-Depot Vehicle Routing Problem with Time Windows* (MDVRPTW) and heterogeneous fleet. The main idea is an item collection from given number of customers. We are provided with specific number of vehicles with different capacities (i. e. heterogeneous fleet) and each departing from a depot and ending in it as well. We also assume that there is more than one depot. Each customer is visited only once and has its own collection demand. Let us consider that each customer has also a time window in which wishes to be visited.

The mathematical formulation of this MDVRPTW with heterogeneous fleet is described as follows. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{1, ..., n+m\}$ denotes a set of vertices and $\mathcal{E} = \{(i, j) \,|\, i, j = 1, 2, ..., n+m;\, i \neq j\}$ depicts links between the vertices. Set $\{1, ..., n\}$ represents the set of customers and the remaining m nodes describe depots. Each customer has non-negative demand d_i. Let denote by $c_{ij} \geq 0$ a travel cost between node i and j in a graph $\mathcal{G}$. A set of vehicles is denoted by $\mathcal{K}$, capacity of vehicle k is denoted by Q_k. Let t_{ij} denote travel time between nodes i and j and service time at node i be denoted by T_i. Parameters l_i and u_i denote lower and upper bound for a visit of customer i. Furthermore, denote by x_{ijk} an indicator whether vehicle k travels from node i directly to node j, that is for any i, $j \in \mathcal{V}$ and $k \in \mathcal{K}$ holds.

$$x_{ijk} = \begin{cases} 1 & : \text{ node } j \text{ is visited right after node } i \text{ by vehicle } k, \\ 0 & : \text{ otherwise,} \end{cases}$$

then the formulation of MDVRPTW with heterogeneous fleet is following:

$$\min \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} \sum_{k \in \mathcal{K}} c_{ij} x_{ijk} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n+m} \sum_{k \in \mathcal{K}} x_{ijk} = 1 \qquad i \in \{1, ..., n\};\ i \neq j \tag{2.2}$$

$$\sum_{i=1}^{n+m} \sum_{k \in \mathcal{K}} x_{ijk} = 1 \qquad j \in \{1, ..., n\};\ i \neq j \tag{2.3}$$

$$\sum_{i=1}^{n+m} x_{ihk} - \sum_{j=1}^{n+m} x_{hjk} = 0 \qquad h \in \{1, ..., n+m\};$$

$$i \neq h, j \neq h, \forall k \in \mathcal{K} \tag{2.4}$$

$$\sum_{i=1}^{n+m} d_i \sum_{j=1}^{n+m} x_{ijk} \leq Q_k \qquad \forall k \in \mathcal{K} \tag{2.5}$$

$$s_{ik} + T_i + t_{ij} - s_{jk} \leq \mathbf{M}\,(1 - x_{ijk}) \qquad \forall i,\ j \in \{1, ..., n\},\ \forall k \in \mathcal{K} \tag{2.6}$$

$$l_i \leq s_{ik} \leq u_i \qquad \forall\, i \in \{1, ..., n+m\},\ \forall k \in \mathcal{K} \tag{2.7}$$

$$x_{ijkt} \in \{0, 1\} \qquad \forall\, i,\ j \in \{1, ..., n+m\},\ \forall k \in \mathcal{K}. \tag{2.8}$$

In the objective function (2.1) we minimize total cost. Constraints (2.2) and (2.3) assure that there is only one predecessor and only one successor for each customer. Flow conservation is secured by constraints (2.4). Capacity constraints can be found in (2.5). Time windows which works also as SECs are defined in constraints (2.6) and (2.7) – here we use big-$\mathbf{M}$ constant which activates the constraint whenever is x_{ijk} equal to 1. Constraints (2.8) define the domain of the binary variable.

2.3 Solution approaches for more complex problems

For VRP, where we assume only one depot, were developed solution approaches via exact algorithms, heuristic and metaheuristic algorithms. If we consider a situation where we have more than just one depot such that each depot possesses some positive number of vehicles, it is more challenging problem to solve. A single depot VRP is NP-hard problem, thus evidently MDVRP is more expensive in terms of time. Therefore are heuristics considered.
To solve VRP generally with heuristics, two phases are taken into account:

1. *clustering phase*: customers are assigned to the corresponding depot according to some metric;

2. *routing phase*: within each cluster routes of minimal cost are sought.

In GEETHA ET AL., [10], is stated that it is possible to solve the problem in both ways, i. e. first routing phase and then perform clustering, or clustering

first and after that the routing phase. Most often is the second approach where neighborhood customers are grouped based on the location of depots or its capacity and then an optimal route is sought within each cluster for each vehicle .

2.3.1 Clustering

MDVRP graph based on GEETHA ET AL., [10], is defined as follows: Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph where $\mathcal{V} = \{v_1, \dots, v_n\}$ is a set of vertices, $\mathcal{E} = \{(v_i, v_j) \,|\, i, j = 1, ..., n,\ i \neq j\}$ is a set of edges depicting links between vertices. $\mathcal{V}$ is partitioned into two subsets $\mathcal{V}^D = \{v_1, \dots, v_k\}$ and $\mathcal{V}^C = \{v_{k+1}, \dots, v_n\}$ representing a set of k depots and $n-k$ customers, respectively; $k < n \in \mathbb{N}$.

The $n-k$ customers are grouped to form k clusters based on some metric. In each cluster there is number of customers which we denote by n_j for $j = 1, ..., k$ with the condition that

$$\sum_{j=1}^{k} n_j = n - k.$$

In this problem, after clustering, for each depot there will be n_j customers from which, e. g. it is necessary to collect items. On top of that, customers are again split into ℓ_j groups, where each group is determined for a vehicle based in the corresponding depot. For the groups holds the following condition:

$$\sum_{q=1}^{\ell_j} n_j^q = n_j \quad j = 1, ..., m.$$

As was written in the previous section, the clusters are pairwise disjoint and their union gives the whole set of all vertices, i. e. $\mathcal{V}$. There are depot and customer vertices ($\mathcal{V}^D$ and $\mathcal{V}^C$, respectively), as mentioned above. The $n-k$ customers are clustered into k subsets based on their Euclidean distance to each depot. We therefore get a simplification of the problem in a form of k single-depot VRPs. One of common clustering algorithms is K-means++. As described in KUČERA, [11], K-means++ is an iterative method of clustering which splits a set into $k \geq 2$ clusters based on similarities of elements in a cluster and also their dissimilarities to elements from other clusters. The algorithm is explained below in Algorithm 1.

Algorithm 1 K-means algorithm

STEP 0: randomly select the first centroid;
STEP 1: calculate distances between the first centroid and other elements, and chose the furthest elements as another centroid;
STEP 2: create clusters such that remaining points are appended to the closest centroid
STEP 3: denote the farest point as another centroid;
STEP 4: repeat steps 2 and 3 until number of required centroids is found.

Centroid in a finite dimension space is defined according to DAVIS in [12] as follows: Let $A \subset \mathbb{R}^n$ be bounded with non-empty interior. *Centroid* is then defined: $\text{CENTROID}\,(A) = \frac{1}{\mu(A)} \int_A id\, d\mu$, where id is identity function on $\mathbb{R}^n$

and μ is Lebesgue measure. Since A is bounded and has non-empty inferior, then $\mu(A)$ is finite and positive.

There are other clustering algorithms such as, e. g., *Density-based spatial clustering of applications with noise* (abb. DBSCAN) or simple K-means algorithm. DBSCAN works with density of elements which in case of VRP should mirror the fact that it is not suitable for areas with different density of population. For more detailed description of abovementioned algorithms we refer to see KUČERA ([11], p. 9–16).

2.3.2 Routing

When clustering phase is done, the routing phase is initiated. As written at the beginning of this section, an exact algorithm for complex problems would be very expensive in a matter of time. In case of single-depot VRP have been developed many approaches to find exact solution as *branch-and-bound* in FISHER, [13], or *branch-and-cut* in LADANYI, RALPHS AND TROTTER, [14]. But finding an optimal solution of MDVRP is almost impossible even for small-sized problem instances. Therefore were developed other ways to find solutions in some reasonable time - *heuristics*. In 1964 CLARKE AND WRIGHT proposed a simple procedure *"but effective in producing a near-optimal solution"* for *"routing of a fleet of trucks of varying capacities used for delivery from a central depot to a large number of delivery points"* ([15], p. 568), a procedure called *saving criterion*. The algorithm starts with initial solution where each customer is assigned to a depot and each vehicle to each customer. The aim is to minimize distance traveled by each vehicle by reducing returns to the depot. Based on this procedure, TILLMAN, [16], introduced a heuristic algorithm to solve, as the name of the article indicates, *The Multiple Terminal Delivery Problem with Probabilistic Demands*. The modification takes into account several number of depots.

During the seventies was used so-called *sweep procedure* to solve multi-terminal, i. e. multi-depot, vehicle dispatch problem in a work of GILLET AND JOHNSON, [17], for which was a fulcrum an article by GILLET AND MILLER, [18], giving a heuristic using sweep algorithm for a single vehicle dispatch problem. Main pillar of the sweep algorithm is a reference axis having an origin in a depot; each customer/city is assigned to the closest depot and then for each customer/city is calculated polar angle between this customer/city and the depot using given reference axis. The algorithm alone was presented by WREN AND HOLLIDAY in [19]. From perspective of stochasticity, although unknown customer's demands were described e. g. in the abovementioned TILLMAN, in 1982 RAFT provided a paper in which he included probability of a customer requiring a visit, i. e. a possibility of postponing some of visits, and presented modular approach where the problem was partitioned into five subproblems: *the route assignement* in which customers were clustered into compact clusters with expectation of a small route length. In the next phase, *depot assignment*, each route is assigned to one of the depots and afterwards it is aimed to minimize the expected route length which is calculated by using λ-optimal (*shortly λ-opt*) algorithm proposed by LIN in [20] where $\lambda = 2$. Then follows *vehicle assignment* in which, for each depot are assigned routes for vehicles such that no vehicle violates its maximal route length. Lastly, phases *delivery period* and *detailed route*

design determines the period in which the delivery has to take a place and a construction of the route using Lin's 3-opt algorithm in the end of the phase.

Now, we are getting to a year in which the main algorithm of this work was introduced, hence let us detach it into a seperate subsection.

2.3.3 Simulated annealing

"There is a deep and useful connection between statistical mechanics (the behavior of systems with many degrees of freedom in thermal equilibrium at a finite temperature) and multivariate or combinatorial optimization."

Kirkpatrick et al.

In 1983 Kirkpatrick et al., [21], introduced a metaheuristic, nowadays known as *simulated annealing*, which was inspired by an algorithm proposed by Metropolis et al., [22], which is possible to use to obtain an efficient simulation of collection of atoms in equilibrium at some given temperature. Kirkpatrick describes the algorithm in a following way - an atom, in each step of the algorithm, is given a slight random displacement and afterwards is calculated resulting change ΔE in the energy of the system. If $\Delta E \leq 0$, then this displacement is accepted and the setting of the system with this displacement is used as a starting point of the next step. The complement condition, i. e. $\Delta E > 0$, is treated probabilistically. An acception of a configuration with the complement condition is $\mathbb{P}(\Delta E) := \exp\{-\frac{\Delta E}{k_B T}\}$. Let ε be a randomly chosen number from interval $(0, 1)$; if $\varepsilon < \mathbb{P}(\Delta E)$, then the new configuration is retained, otherwise the original configuration is used to start the new step. In statistical thermodynamics it is assumed that the system is in contact with a heat bath of a temperature T; and by repeating these steps we simulate the thermal motion of atoms which are in contact with the heat bath. The consequence of the choice $\mathbb{P}(\Delta E)$ is that the system evolves into a Boltzmann distribution.

Let us have a system of n independent particles that might exist in a number of different states with energies E_i, where $i \in \{0, ..., r-1\}$. The independency implies that energy of the whole system is sum of each particle's energy. Denote number of particles with energy E_i as n_i. Let p_i be probability of a system being in a state with energy E_i, i. e. finding a particle with energy E_i, is therefore equal to $p_i = n_i/n$. By using Lagrange multipliers and Maxwell's kinetic gas theory (we refer to [23]) can be shown that

$$p_i = \frac{n_i}{n} = \frac{\exp\left\{\frac{-E_i}{k_B T}\right\}}{\sum_{k=0}^{r-1} \exp\left\{\frac{-E_k}{k_B T}\right\}},$$

where $k_B = R/N_A$ is the Boltzmann constant which is a ratio of the universal gas constant R and the Avogadro constant N_A. The ratio of probabilites of two states

is known as *Boltzmann factor*:

$$\frac{p_i}{p_j} = \frac{\exp\left\{\frac{-E_i}{k_B T}\right\}}{\sum_{k=0}^{r-1} \exp\left\{\frac{-E_k}{k_B T}\right\}} \Bigg/ \frac{\exp\left\{\frac{-E_j}{k_B T}\right\}}{\sum_{k=0}^{r-1} \exp\left\{\frac{-E_k}{k_B T}\right\}} = \exp\{-(E_i - E_j)/k_B T\} =$$

$$= \exp\left\{ - \frac{\Delta E_{(i,j)}}{k_B T} \right\},$$

which gives us a probability of a change in a system between states i and j. Boltzmann distribution thus characterize a state of a system of particles with respect to temperature and energy (see [23]).

The algorithm utilizes the process of annealing as we heat up the particles up to temperature T and afterwards we let it cool down until the state of equilibrium is reached. In order to present a pseudocode of simulated annealing algorithm it is necessary to define so-called *evaluation function* $f(\cdot)$ which in terms of VRP behaves in the algorithm as a substitution of an objective function. In Algorithm 2 we provide pseudocode of simulated annealing algorithm.

Algorithm 2 Simulated Annealing

$T \leftarrow T_{max}$
ε: random number from interval $(0, 1)$
$x_{initial} \leftarrow$ INIT()
$x_{best} \leftarrow x_{initial}$
$f_{best} \leftarrow f(x_{initial})$
$f_{current} \leftarrow f(x_{initial})$
while $T > T_{min}$ **do**
  $x_{next} \leftarrow$ NEIGHBOR(x_{best})
  $f_{next} \leftarrow f(x_{next})$
  $\Delta f \leftarrow f_{next} - f_{current}$
  if $\Delta f \leq 0$ **then**
    $x_{current} \leftarrow x_{next}$
    $f_{current} \leftarrow f_{next}$
  else if $\varepsilon < \mathbb{P}(\Delta f) = \exp\{\frac{-\Delta f}{k_B T}\}$ **then**
    $x_{current} \leftarrow x_{next}$
    $f_{current} \leftarrow f_{next}$
  end if
  if $f_{current} < f_{best}$ **then**
    $x_{best} \leftarrow x_{current}$
    $f_{best} \leftarrow f_{current}$
  end if
  $T \leftarrow$ COOLING(T)
end while
return x_{best}

First of all, we need to provide to SA an initial solution $x_{initial}$ and also the starting temperature T. Until we reach the preset lowest temperature – during the process of cooling – at each temperature we modify the current solution x_{best} by using operations of a random displacement such as swap or reverse, etc.

To demonstrate, e. g. swap operation let us consider a sequence of four subsequent numbers, that is for example $\{1, 2, 3, 4\}$. Operation $swap(\cdot)$ can be defined as follows: for any given sequence randomly chooses its two components and swap them; in words of our example, randomly chosen components can be 2 and 3, then $swap(\{1, 2, 3, 4\}) = \{1, 3, 2, 4\}$. Such modified solutions can be elements of neighborhood of the feasible solution only if they are also feasible - here we refer to the function NEIGHBOR(): for temperature T we modify our currently best solution x_{best} by some chosen operation, such as $swap(\cdot)$, and if the modification does not violate the feasibility, it can be denoted as x_{next} and evaluated it by the evaluation function $f(\cdot)$. If $f(x_{next}) - f(x_{best}) \leq 0$, then we found a better solution than the initial (or same one as the initial); if not, we can still accept such a solution with probability $\varepsilon < \exp\{\frac{-\Delta f}{k_B T}\}$, or reject it with probability of $1 - \varepsilon$. After process of accepting/rejecting the current solution we cool down temperature T by preset and continue until we hit the minimal temperature T_{min}.

Metaheuristic SA is very often used to solve VRP and its modifications as can be read e. g. in OSMAN, [24], AYDEMIR AND KARAGÜL, [25] or WANG ET AL., [26]; let us also not forget about VLADIMÍR ČERNÝ, a member of *Faculty of mathematics, physics and informatics* in Comenius University in Bratislava, Slovakia, who in 1985 published (compare with KIRKPATRICK ET. AL) an article named *Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm*, in which he writes: *"The algorithm generates randomly the permutations of the stations of the traveling salesman trip, with probability depending on the length of the corresponding route. Reasoning by analogy with statistical thermodynamics, we use the probability given by the Boltzmann-Gibbs distribution. Surprisingly enough, using this simple algorithm, one can get very close to the optimal solution of the problem or even find the true optimum"* ([27], p. 1).

We have examined some of the important solving approaches of VRP problems including a metaheuristic nowadays called simulated annealing which will take place in upcoming chapters. But before that it is a duty to formulate the problem we keen on solving.

3. Waste collection problem

Few years ago several areas in South Moravian region decided to take charge of their waste collection and not to have it collected by local services. Since then there has been a development to obtain an efficient scheduling. Members from *Institute of Process Engineering* located at *Faculty of Mechanical Engineering* at *University of technology* in Brno were asked to provide such a solution. The development has got into a point where the team from *Institute of Process Engineering* (further IPE) evolves an application in which an user can configurate the problem instance itself, e. g. selection of cities, frequency of the waste collection for a specific city or allowed days of the collection, and afterwards provide the user an efficient schedule of the waste collection. Due to ongoing changes of the assignment, solving of this problem still remains. Also KUČERA addressed the issue in his master thesis, [11]. PROCHÁZKA developed an algorithm called POPELAR to solve *Multi-Trip Periodic Capacitated Arc Routing Problem* (MTPCARP) and NEVRLÝ improved the algorithm in terms of computing time. Nonetheless, neither of these approaches were shared to avoid any inspiration for this work, and thus have a possibility to offer the team a different approach – for a deeper insight of these works we refer to [28], [29]. In this chapter we propose a mathematical MILP formulation of this problem. But first, let us describe the problem.

3.1 Problem formulation

Objective of this thesis is to propose an efficient composition of paths and subsequent schedule for a specific number of garbage vehicles departing from given number of depots in South Moravian region and create waste collection schedule for a preset time horizon. Let us now specify assumptions of the problem thoroughly:

AREAS For each area we have a depot (or depots) containing given number of grabage vehicles and number of subareas where it is crucial to perform waste collection. A subarea can be a whole city or, in case of a large city, a smaller part of the city. Each area has different time duration of waste collection.

VEHICLES Every vehicle has its own depot which is its starting point at the beginning of the shift and also ending one at its end. Each vehicle has its own volume capacity and permitted route time length. Lastly, in the given depot there might be more than just one vehicle.

DEMANDS Each area has its own demand. During the waste collection capacity of a vehicle cannot be exceeded, thus if a vehicle is en route and after visiting several subareas has no capacity left to continue in waste collection, it has to travel to a disposal, unload cargo and proceed in the collection.

FREQUENCY A subarea can choose with what frequency, e. g. if twice per week or just once, and can specify also on which days the waste collection has

to be performed. Each subarea is visited only once on – for it allowed – day of the collection. For example subarea C requests to have waste collected once per two weeks on Tuesdays and another subarea, let us say subarea D, has no time requests at all and thus the waste can be performed on any day according to its frequency.

SHIFTS Time duration of a shift depends on working hours for each garbage vehicle. If time duration of waste collection is exceeded, it is treated as overtime. Also, we allow some tolerance if the length of a shift is exceeded, but it cannot happen that also this tolerance is violated.

As we have the assignment of the problem, let us proceed to its mathematial formulation.

3.2 Mathematical formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{v_1, ..., v_{n+m+r}\}$ is a set of nodes and $\mathcal{E} = \{(v_i, v_j) \in \mathcal{V}^2;\ i \neq j\}$ depicts a set of edges. Further, let $\mathcal{V}$ be splitted into three disjunct sets – set of depots $\mathcal{D}$, set of cities $\mathcal{C}$ and set of disposals $\mathcal{L}$ consisting of n, m and r nodes, respectively. Hereafter, let $\mathcal{K}$ describe a set of garbage vehicles, set of days on which the waste collection will be performed let be denoted by $\mathcal{T}$, where card($\mathcal{T}$) defines time horizon.

According to the definition of PVRP from CHRISTOFIDES AND BEASLEY, [30], a set of schedules $\mathcal{H}$ is introduced in a following way: each schedule consists of a set of days in which a city allows to perform waste collection. Each city will be served on every day of the schedule, i. e. city $i \in \mathcal{C}$ has schedule

$$\mathcal{H}_i = \{\, h \in \mathcal{H} : \sum_{t \in \mathcal{T}} a_{th} = f_i \,\},$$

where f_i denotes frequency of waste collection for city i and a_{th} is a binary parameter defined as

$$a_{th} = \begin{cases} 1: & \text{if day } t \text{ belongs to schedule } s \in \mathcal{H}, \\ 0: & \text{otherwise.} \end{cases}$$

Let us provide to an example of what a weekly schedule for city i with frequency $f_i = 1$ per one week could look like if allowed days for a visit were days from Monday to Thursday:

$$\mathcal{H}_i = \{(1,0,0,0,0,0,0), (0,1,0,0,0,0,0), (0,0,1,0,0,0,0), (0,0,0,1,0,0,0)\}.$$

With reference to known notations here we model *load-based periodic multi-depot multi-fleet vehicle routing problem with time windows* (further abbreaviated as LBPMDMFVRPTW).

Let us denote input data with which we are about to model the problem:

t_{ij} : travel time to get from node $i \in \mathcal{V}$ to $j \in \mathcal{V}$; holds $t_{ij} \neq t_{ji}$;
d_i : demand of node $i \in \mathcal{V}$;
T_i : service time at node $i \in \mathcal{V}$;
f_i : frequency of serving city $i \in \mathcal{C}$;
Q_k : capacity of vehicle $k \in \mathcal{K}$;
τ_k : a soft upper bound for vehicle k being in use; $k \in \mathcal{K}$;
$\mathcal{K}_d$: set of vehicles assigned to depot $d \in \mathcal{D}$; $\mathcal{K}_d \subseteq \mathcal{K}$ and $\cup_{d\in\mathcal{D}}\mathcal{K}_d = \mathcal{K}$;
γ : overtime tolerance (in hours), i. e. if $\gamma = 1$, then $\tau_k + \gamma$ is the maximum time for vehicle k to be in use; $k \in \mathcal{K}$.

For LBPMDMFVRPTW four sets of binary decision variables are introduced which identify arc traversing, vehicle assignments to each depot, cities that are visited at each time period and if cities are visited according to their schedule.

The definition of decision variables is following:

$$x_{ijkt} = \begin{cases} 1: & \text{if vehicle } k \in \mathcal{K} \text{ travers arc } (i,j) \text{ on day } t \in \mathcal{T};\ i,\ j \in \mathcal{V}, \\ 0: & \text{otherwise;} \end{cases}$$

$$y_{dk} = \begin{cases} 1: & \text{if vehicle } k \in \mathcal{K} \text{ is located in depot } d \in \mathcal{D}, \\ 0: & \text{otherwise;} \end{cases}$$

$$z_{ikt} = \begin{cases} 1: & \text{if city } i \in \mathcal{C} \text{ is visited by vehicle } k \in \mathcal{K} \text{ on day } t \in \mathcal{T}, \\ 0: & \text{otherwise;} \end{cases}$$

$$p_{ikh} = \begin{cases} 1: & \text{if city } i \in \mathcal{C} \text{ is visited by vehicle } k \in \mathcal{K} \text{ according to schedule} \\ & h \in \mathcal{H}_i, \\ 0: & \text{otherwise.} \end{cases}$$

Forth, let us introduce continuous variables used in the model:

w_{ikt} : quantity of waste on board of vehicle $k \in \mathcal{K}$ after visiting node $i \in \mathcal{V}$, on day $t \in \mathcal{T}$;
s_{jkt} : total time spent en route while entering node j (without servicing it), $j \in \mathcal{V}$, by vehicle $k \in \mathcal{K}$ on day $t \in \mathcal{T}$.

Here it is necessary to provide few complementary assumptions. For $i \in \mathcal{D} \cup \mathcal{L}$ holds that $d_i = 0$ and for every $d \in \mathcal{D}$ holds that $T_d = 0$. Also, any of the continuous variables are non-negative real numbers. According to the previous section, Section 3.1, if vehicle k exceeds its allowed time of performing waste collection, then it has to be penalized, and for that reason it is handy to introduce a penalty constant denoted by θ. The penalty evidently uses variable s_{jkt} and has to satisfy two following conditions:

1. if, for each day, the total time of vehicle $k \in \mathcal{K}$ spent en route is less than τ_k, the penalty term remains equal to zero;

2. once the total time of vehicle $k \in \mathcal{K}$ spent en route is greater than τ_k, the difference between the total time of vehicle k and τ_k is multiplied by the penalty constant θ, for each day.

To obtain a penalty function fulfilling these two conditions one has to utilize a function called *positive part of real-valued function* defined as

$$[f(x)]^+ = \max\{0, f(x)\}.$$

Now, we are ready to introduce mixed integer linear program to solve LBP-MDMFVRPTW.

The formulation is thus following:

$$\min \left(\sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{V}} \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}} (t_{ij} + T_i)\, x_{ijkt} + \theta \sum_{d\in\mathcal{D}} \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}} \Big[s_{dkt} - \tau_k \Big]^+ \right) \tag{3.1}$$

subject to

$$\sum_{k\in\mathcal{K}} \sum_{h\in\mathcal{H}_i} p_{ikh} = 1 \qquad \forall\, i \in \mathcal{C} \tag{3.2}$$

$$z_{ikt} = \sum_{h\in\mathcal{H}_i} p_{ikh}\, a_{th} \qquad \forall\, i \in \mathcal{C},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.3}$$

$$x_{ijkt} \le \frac{z_{ikt} + z_{jkt}}{2} \qquad \forall\, i,\, j \in \mathcal{C};\, i \ne j,\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.4}$$

$$\sum_{i\in\mathcal{V}} x_{ijkt} = z_{jkt} \qquad \forall\, j \in \mathcal{C},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.5}$$

$$x_{iikt} = 0 \qquad \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.6}$$

$$\sum_{j\in\mathcal{V},\, j\ne i} x_{ijkt} - \sum_{j\in\mathcal{V},\, j\ne i} x_{jikt} = 0 \qquad \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.7}$$

$$\sum_{d\in\mathcal{D}} \sum_{j\in\mathcal{C}} x_{djkt} - \sum_{j\in\mathcal{C}} \sum_{\ell\in\mathcal{L}} x_{j\ell kt} \le 0 \qquad \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.8}$$

$$\sum_{\ell\in\mathcal{L}} \sum_{j\in\mathcal{C}\cup\mathcal{D}} x_{\ell jkt} - \sum_{j\in\mathcal{C}} \sum_{\ell\in\mathcal{L}} x_{j\ell kt} = 0 \qquad \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.9}$$

$$\sum_{j\in\mathcal{C}} x_{djkt} - \sum_{\ell\in\mathcal{L}} x_{\ell dkt} = 0 \qquad \forall\, d \in \mathcal{D},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.10}$$

$$\sum_{d\in\mathcal{D}} \sum_{\ell\in\mathcal{L}} \sum_{k\in\mathcal{K}} x_{d\ell k} = 0 \qquad \forall\, t \in \mathcal{T} \tag{3.11}$$

$$\sum_{j\in\mathcal{C}} x_{djkt} \le y_{dk} \qquad \forall\, d \in \mathcal{D},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.12}$$

$$y_{dk} = 1 \qquad \forall\, d \in \mathcal{D},\, \forall\, k \in \mathcal{K}_d \tag{3.13}$$

$$y_{dk} = 0 \qquad \forall\, d \in \mathcal{D},\, \forall\, k \in \mathcal{K} \setminus \mathcal{K}_d \tag{3.14}$$

$$w_{ikt} = 0 \qquad \forall\, i \in \mathcal{D} \cup \mathcal{L},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.15}$$

$$w_{ikt} - w_{jkt} + d_j \le \mathbf{M}\,(1 - x_{ijkt}) \qquad \forall\, i \in \mathcal{V},\, \forall\, j \in \mathcal{D} \cup \mathcal{C};\, i \ne j,\, \forall\, k \in \mathcal{K}, \forall\, t \in \mathcal{T} \tag{3.16}$$

$$w_{ikt} \le Q_k \qquad \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T} \tag{3.17}$$

$$t_{dj} - s_{jkt} \leq \mathbf{M}\,(1 - x_{djkt}) \qquad \forall\, d \in \mathcal{D}, \forall\, j \in \mathcal{C}, \\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T} \quad (3.18)$$

$$s_{ikt} + T_i + t_{ij} - s_{jkt} \leq \mathbf{M}\,(1 - x_{ijkt}) \qquad \forall\, i \in \mathcal{C} \cup \mathcal{L},\ \forall\, j \in \mathcal{V};\ i \neq j, \\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T} \quad (3.19)$$

$$l_k \leq s_{ikt} \leq \tau_k + \gamma \qquad \forall\, i \in \mathcal{V},\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T} \quad (3.20)$$

$$x_{ijkt} \in \{0, 1\} \qquad \forall\, i,\ j \in \mathcal{V};\ i \neq j, \\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T} \quad (3.21)$$

$$y_{dk} \in \{0, 1\} \qquad \forall\, d \in \mathcal{D},\ \forall\, k \in \mathcal{K} \quad (3.22)$$

$$z_{ikt} \in \{0, 1\} \qquad \forall\, i \in \mathcal{V},\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T} \quad (3.23)$$

$$p_{ikh} \in \{0, 1\} \qquad \forall\, i \in \mathcal{V},\ \forall\, k \in \mathcal{K},\ \forall\, s \in \mathcal{H}_i. \quad (3.24)$$

The objective function (3.1) minimizes total travel time with penalty for any overtime. Constraints (3.2) establish that for each city there is only one schedule throughout the time horizon. To preserve that each city can be visited by given garbage vehicle only on that day if the day is a part of its schedule, it was needed to add constraints (3.3). Constraints (3.4) and (3.5) relate decision variables x_{ijkt} and z_{ikt}. By providing constraints (3.6) we restrict looping at the same node. Flow conservation constraints can be found in (3.7). Constraints (3.8) allow that if any given garbage vehicle on certain day leaves a depot, then during the day it visits a disposal at least once. Following contraints, (3.9), establish for any garbage vehicle and any given day departing disposal only in cases if the vehicle had arrived into the disposal; notice that the vehicle can travel from the disposal back to cities or depots. Constraints (3.10) assure for each day visiting a disposal by any garbage vehicle right before returning to any depot. In (3.11) we restrict, for any day, visiting a disposal right after leaving a depot at the beginning of the shift. Contraints (3.12)–(3.14) preserve that from any depot leaves a garbage vehicle only if the vehicle is assigned to this depot. In (3.15), (3.16) and (3.17) are defined capacity constraints; (3.15) impose that any garbage vehicle is empty after leaving arbitrary depot or disposal, (3.16) define the continuous variable w_{ikt} describing quantity of waste onboard of a garbage vehicle after leaving any node and (3.17) assure that quantity of waste onboard cannot exceed capacity of the vehicle. Constraints (3.18)–(3.20) are the time windows constraints which also work as subtour elimination constraints. Finally, constraints (3.21)–(3.24) define the domain of the variables.

In constraints (3.16), (3.18) and (3.19) we can notice an optimization method called *big* $\mathbf{M}$ *method* where the $\mathbf{M}$, as a large constant, activates or disactivates given constraint.

The penalty term

$$\theta \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} \Big[s_{dkt} - \tau_k \Big]^+$$

in terms of implementing has to be reformulated by introducing another continuous variable g_{dkt} and extend the model with following constraints

$$g_{dkt} \geq 0 \qquad \forall\, d \in \mathcal{D}, \forall\, k \in \mathcal{K}, \forall\, t \in \mathcal{T} \tag{3.25}$$

$$g_{dkt} \geq s_{dkt} - \tau_k \qquad \forall\, d \in \mathcal{D}, \forall\, k \in \mathcal{K}, \forall\, t \in \mathcal{T}, \tag{3.26}$$

and modify the term of the objective function into ensuing form

$$\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} (t_{ij} + T_i)\, x_{ijkt} + \theta \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} g_{dkt}.$$

One might ask why domain of the first sum is only over set of depots $\mathcal{D}$. The explanation lies in the definition of variable s_{jkt}. It is a cumulative sum of time spent en route right after we arrive in node j but without servicing it. And thus, if we want to obtain the total time spent en route, we just have to get value of s_{dkt} for $d \in \mathcal{D}$, and for each $k \in \mathcal{K}$, $t \in \mathcal{T}$, since service time in any depot is equal to zero.

4. Solving approaches

In this chapter we offer two approaches for solving the problem from previous chapter. At first, let us start with exact approach.

4.1 Exact approach

Since we have formulated LBPMDMFVRPTW and its mathematical formulation, it is possible to implement it into any optimization software or optimization package for given programming language designated for solving mixed integer linear programming problems. As mentioned earlier, it is very expensive in terms of time to use exact approaches for any big instance which unfortunately, on the other hand, mirrors the real-world problems.

In this work we decided to use GUROBI as our main solver from a company called *Gurobi Optimization Inc.* If one would search for any open source alternative to GUROBI which in free version is limited, we suggest trying COIN-OR (Computational Infrastructure for Operations Research). MEINDL AND TEMPL proposed in [31] an analysis of commercial solvers for linear optimization problems and GUROBI yielded, in the comparison, the best results of all. For a small insight, let us utter some of other solvers which were in the article for comparison: CPLEX from *IBM ILOG CPLEX Optimization Studio* or XPRESS from *Xpress Optimization Suite*, for example.

Let us start with notation of a problem instance. By notation 2/7/1/2/2 we understand – and as it is set here, it will be used in this way for the rest of the work – an instance composed from 2 depots, 7 cities, 1 disposal, 2 vehicles at each depot per 2-day time horizon, respectively. As reader can notice, such instance, even though is solvable in GUROBI, does not represent any real-world situation. Closer to some real-world problem would be, e. g. instance $3/50/2/*/14$, where according to the problem formulation each city has its own frequency of waste collection and allowed days as well. Symbol $*$ in the instance notation represents the fact that each depot can contain various number of garbage vehicles. The variety of vehicles is also taken into account in the model from previous chapter.

4.2 Metaheuristic approach

The aim of this work was to offer a solution for waste collection for specific number of areas in South Moravian region which is unfortunately expensive to obtain by the exact approach. Thus in this section we present an approach in which we construct an algorithm, based on *Simulated annealing* (SA), solving LBPMDMFVRPTW from Chapter 3. Remark that all notation from the model will be seized in this section as well.

SA is an improving metaheuristic which at its beginning works with an initial solution for LBPMDMFVRPTW, therefore, first we need to construct an algorithm yielding such a solution according to constraints from the model. This kind of algorithms belongs into *local search algorithm* class.

4.2.1 Construction of an initial solution

Let us describe the principal according to which the algorithm runs. At the beginning, for each city of the instance we find its closest depot and persuant to it assign the city to its closest depot. Thus, we end up with card($\mathcal{D}$) sets, let us denote them by $\mathcal{C}_d \subseteq \mathcal{C}$ where $\mathcal{C}_d$ describes subset of all cities $\mathcal{C}$ closest to depot $d \in \mathcal{D}$. For clarity reasons we provide an example in Figure 4.1 where we consider two depots denoted by d_1, d_2 and 10 cities denoted by $c_1, ..., c_{10}$. Thus according to abovementioned split of set of citites we obtain card($\mathcal{D}$) = 2 sets such that $\mathcal{C}_{d_1} = \{c_1, c_2, c_6, c_7\}$ and $\mathcal{C}_{d_2} = \{c_3, c_4, c_5, c_8, c_9, c_{10}\}$.

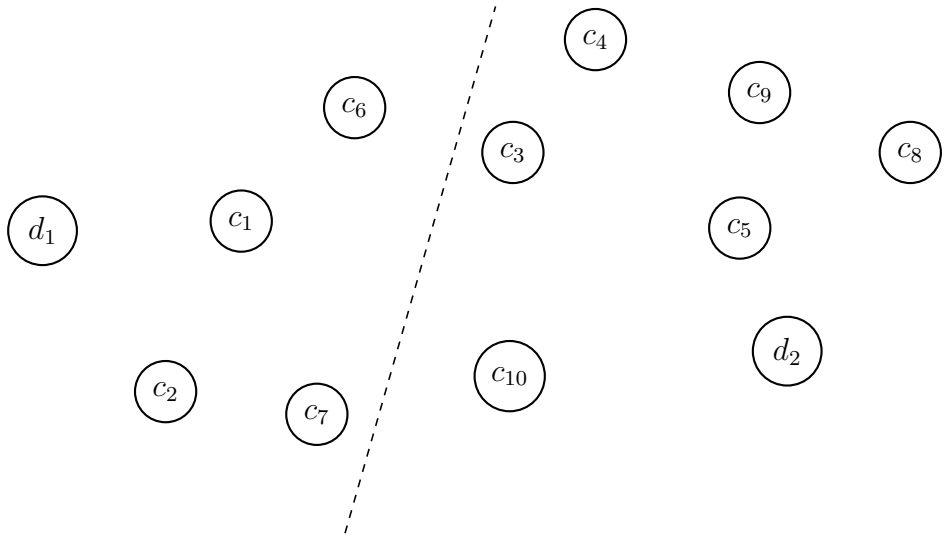

Figure 4.1: Example of city assigning to its closest depot.

Here, we have to take into account the fact that each city has its own schedule which is obtained from its frequency and allowed days of waste collection. Let us fix, for the sake of simplification, arbitrary $d \in \mathcal{D}$ and corresponding $\mathcal{C}_d$. For each city $i \in \mathcal{C}_d$ we obtain its frequency f_i and a set of the allowed days per preset time horizon card($\mathcal{T}$), which for card($\mathcal{T}$) = 3 and if allowed days for waste collection are, e. g. Monday and Wednesday, can be a set $\{1, 3\}$ where clearly 1 is a numeric representation of Monday and 3 a numeric representation of Wednesday. In case of greater time horizon, e. g. card($\mathcal{T}$) = 14, set of allowed days would be of form $\{1, 3, 8, 10\}$. From the set of allowed days we randomly choose exactly f_i distinct days and according to these distinct days we construct a feasible schedule for city i. This process we iterate over all days of time horizon card($\mathcal{T}$).

Now, we proceed to the main part of the algorithm for which we also provide to reader a simplified pseudocode. We have obtained a schedule for all cities per all days of the planning time horizon but note that this schedule is not feasible in terms of capacity constraints from the model. Moreover, disposals are not contained in the schedule at all and thus garbage vehicles are also necessary to be taken into account. This part of the algorithm works as described in the following sentences.

Again, for the matter of simplifying things, let us consider a fixed day of the schedule. On this day we have constructed routes. Let us highlight the fact

that here each route is defined by the depot and its closest cities such that time demands for city are fulfilled, as described in the second paragraph of this subsection. For each depot $d \in \mathcal{D}$ we get vehicles which are assigned to this depot, let the set of vehicles assigned to the depot d denote by $\mathcal{K}_d \subseteq \mathcal{K}$. Here, we split the route of depot d into $\text{card}(\mathcal{K}_d)$ subroutes based on the following principle (denote cardinality of the route of depot d by $\text{card}(\mathcal{R}_d)$):

1. if $\text{card}(\mathcal{R}_d)$, is even and $\text{card}(\mathcal{K}_d)$ is also even, then the route is split into same-sized subroutes of cardinality $\text{card}(\mathcal{R}_d)/\text{card}(\mathcal{K}_d)$;

2. if $\text{card}(\mathcal{R}_d)$ or $\text{card}(\mathcal{K}_d)$ or both are odd, then first we split the route into $\text{card}(\mathcal{K}_d)$ subroutes each containing η cities and the last ζ subroutes contains $\eta + 1$ cities where

$$\eta = \lfloor \text{card}(\mathcal{R}_d)/\text{card}(\mathcal{K}_d) \rfloor,$$
$$\zeta = \text{card}(\mathcal{R}_d) \mod (\text{card}(\mathcal{K}_d)),$$

   thus the route is splitted into $(\text{card}(\mathcal{K}_d) - \zeta)$ subroutes containing η cities and ζ subroutes containing $(\eta + 1)$ cities.

To continue, every vehicle $k \in \mathcal{K}_d$ has capacity Q_k. Now that each vehicle k has its own subroute to serve on day t (let us denote the subroute by $r^t_{d_k}$), let us cumulatively calculate demands of the cities in this subroute. If it happens that by visiting some city i in the subroute the quantity onboard would exceed Q_k, for predecessor of city i we find its closest disposal, insert the closest disposal before visiting city i, unload the cargo and then return to collect waste of city i. One more thing to take care of in terms of capacity constraints is that the vehicle has to return to its depot unloaded. Therefore, the process described above in this paragraph is iterated until we reach the last city of the subroute. If the vehicle reaches the last city, the algorithm finds its closest disposal, insert the disposal right after visiting the last city into the subroute, unloads cargo, returns to its depot and inserts the depot as the last element of the subroute.

For it may seem complicated, we provide also a simplified pseudocode on page 31 in Algorithm 3. Dear reader will certainly understand that for the sake of saving space we only provide a sample from the whole algorithm, i. e. for the case until we reach the last city of the subroute for in the complement case it is done analogously (in the pseudocode we denoted the analogous part of the algorithm by symbol $*$).

The resulting initial schedule is, in mathematical sense, of the following form:

$$\left(r^{t_\tau}_{d_{k_\kappa}}\right)^{K,T}_{\kappa,\tau=1} = \begin{pmatrix} r^{t_1}_{d_{k1}} & r^{t_2}_{d_{k1}} & r^{t_3}_{d_{k1}} & \dots & r^{t_T}_{d_{k1}} \\ r^{t_1}_{d_{k2}} & r^{t_2}_{d_{k2}} & r^{t_3}_{d_{k2}} & \dots & r^{t_T}_{d_{k2}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r^{t_1}_{d_{kK}} & r^{t_2}_{d_{kK}} & r^{t_3}_{d_{kK}} & \dots & r^{t_T}_{d_{kK}} \end{pmatrix}, \tag{4.1}$$

where $r^{t_\tau}_{d_{k_\kappa}}$ represents a subroute which starts in depot to which is assigned vehicle k_κ such that $\kappa \in \{1, ..., K\}$ – set $\{k_1, ..., k_K\} = \mathcal{K}$ represents whole fleet

Algorithm 3 Local Search Algorithm

localSearchSchedule = *dict*()
for day, routes **in** schedules **do**
  daySchedule ← list()
  for $d \in \mathcal{D}$ and $\mathcal{C}_d \subseteq \mathcal{C}$ **do**
    get $\mathcal{K}_d$ as a set of vehicles assigned to depot d
    route ← *list*()
    $r_d^k \in$ GETSUBROUTES($\mathcal{C}_d, \mathcal{K}_d$)
    for $k \in \mathcal{K}_d$ **get** Q_k and **do**
      if r_d^k **not** empty **then**
        lastNodeVisited ← d
        $w_k = 0$
        $w_{cumulative}$ ← list()
        vehiclePath = list(k, d)
        for city i in r_d^k **do**
          while city $i \neq$ LASTCITY(r_d^k) **do**
            if $w_k + d_i \leq Q_k$ **then** ▷ d_i: demand of city i
              append i **into** vehiclePath
              $w_k \leftarrow w_k + d_i$
              append w_k **into** $w_{cumulative}$
              lastNodeVisited ← i
            else if $w_k + d_i > Q_k$ **then**
              $\ell \leftarrow$ FINDCLOSESTDISPOSAL(lastNodeVisited)
              append ℓ **into** vehiclePath
              $w_k \leftarrow 0$
              append w_k **into** $w_{cumulative}$
              append i **into** vehiclePath
              $w_k \leftarrow w_k + d_i$
              append w_k **into** $w_{cumulative}$
              lastNodeVisited ← i
            end if
          end while
          *
        end for
        append d into subroute r_d^k
        append r_d^k into route
      end if
    end for
  end for
  map *daySchedule* into `dict` with vehicles as `keys` and subroutes as `items`
  update *localSearchSchedule* as `dict` with day as `key` and *daySchedule* as `item`
end for
return *localSearchSchedule*

of vehicles – and is performed on day $t_\tau \in \{t_1, ..., t_T\} = \mathcal{T}$. In case that subroute is not performed, we understand the route as an empty route which has no costs.

Now, that we have constructed an algorithm to yield an initial solution, we are ready to introduce to reader SA algorithm.

4.2.2 Modified simulated annealing algorithm

In this subsection we refer to Subsection 2.3.3, *Simulated annealing*, in which we introduced an algorithm inspired by annealing in metallurgy. The algorithm works with evaluation of energy of the heated system at specific temperature (analogous function to an objective function) and by controlled cooling its aim is to bring the system into a state with minimal energy of the system.

In the algorithm at each temperature of the cooling procedure we slightly perform small displacement in the system, which in words of our problem is a displacement of a city or cities in the whole schedule - this displacement is in the pseudocode denoted by function NEIGHBOR() which as an input takes the last best schedule. Now let us describe which displacements were taken into account in the algorithm.

The algorithm itself works with two types of displacement. The first one is change of position of one random city and the second one is swap of two random cities. Before we start decribing the displacement operations, let us remind the design of the schedule:

$$\left(r_{d_{k_\kappa}}^{t_\tau}\right)_{\kappa,\tau=1}^{K,T} = \begin{pmatrix} r_{d_{k1}}^{t_1} & r_{d_{k1}}^{t_2} & r_{d_{k1}}^{t_3} & \dots & r_{d_{k1}}^{t_T} \\ r_{d_{k2}}^{t_1} & r_{d_{k2}}^{t_2} & r_{d_{k2}}^{t_3} & \dots & r_{d_{k2}}^{t_T} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{d_{kK}}^{t_1} & r_{d_{kK}}^{t_2} & r_{d_{kK}}^{t_3} & \dots & r_{d_{kK}}^{t_T} \end{pmatrix}. \tag{4.1}$$

If we recall the description of elements in (4.1) from Subsection 4.2.1, for vehicle k_1 on day t_1 the route could be, e. g. written as a following sequence:

$$r_{d_{k_1}}^{t_1} = \{d_{k_1}, c_1, ..., c_v, \ell_v, d_{k_1}\},$$

where d_{k_1} is a denomination for a depot containing vehicle k_1; $c_1, ..., c_v$ are cities which according to the initial solution has to be served by vehicle k_1 on day t_1, and ℓ_v denotes the closest disposal to city c_v. Of course there is a real chance that between cities c_1 and c_v is at least one stop to unload the cargo at some closest disposal. This is only an illustrative example to help orientate before we dive into description of the modified simulated annealing metaheuristic.

Now, let us begin by describing the first one of the displacement operations – a change of position of the city (further *change* operation).

This idea is very intuitive and follows these steps:

STEP 1: Input a schedule.

STEP 2: Pick a city C randomly, find this city throughout the whole schedule, i. e. over all days and erase it.

STEP 3: For this city C get randomly its schedule; to orientate we provide a simple example: let us for simplicity fix time horizon, card$(\mathcal{T}) = 2$ (e. g. Monday and Tuesday), frequency $f_C = 1$ and Monday and Tuesday as allowed

days for waste collection, thus possible schedules could be of this form $\{(1,0),(0,1)\}$, where $(1,0)$ means schedule for performing the collection on Monday and not performing the collection on Tuesday, etc.

STEP 4: For city C also find its closest city and its closest depot.

STEP 5: For each day of the schedule of city C we proceed as follows:

(a) with probability 1/2 we insert city C into schedule right before visiting its closest city from STEP 3, but only in case if the closest city is served that day; if the closest city is not served on that day, we continue in the search of the closest city until we obtain the closest city which is also served on the same day; if such a city is not found, go to (b);

(b) with probability 1/2 we insert city C after its closest depot.

Note that *change* operation does not take into account if the randomly chosen schedule is not already scheduled because we might still obtain better solution in comparison with the initial one. Let us now proceed to the second displacement operation, *swap* operation.

With reference to Subsection 2.3.3, we already outlined the idea behind this operation on a simple example, but here we face quite complicated structure of the operation. Meanwhile in *change* operation we iterated through the feasible schedule of a given city and thus there was no need to check if the obtained schedule was feasible for the random city, now if we pick randomly two cities which can served, e. g. on different days, by swapping them we could fall into not a feasible schedule for one or either of them. Let us now thus reveal steps of *swap* operation to obtain feasible schedules:

STEP 1: Input a schedule.

STEP 2: Pick randomly two cities, e. g. cities C and D.

STEP 3: For each city get a set of days when they were served in the input schedule – denote them by $\mathcal{P}_C$ and $\mathcal{P}_D$.

STEP 4: Find $\mathcal{P}_C \cap \mathcal{P}_D$, $\mathcal{P}_C \cup \mathcal{P}_D$ and $\mathcal{P}_C \cup \mathcal{P}_D \setminus \mathcal{P}_C \cap \mathcal{P}_D$;

(a) if $\mathcal{P}_C \cap \mathcal{P}_D = \emptyset$, then we will not get a feasible solution;

(b) if $\mathcal{P}_C \cap \mathcal{P}_D \neq \emptyset$, then few cases are need to distinct:

i. if $\text{card}(\mathcal{P}_C \cup \mathcal{P}_D \setminus \mathcal{P}_C \cap \mathcal{P}_D) = 1$, then pick randomly one day from $\mathcal{P}_C \cap \mathcal{P}_D$ and perform the swap on this day;

ii. if $\text{card}(\mathcal{P}_C \cup \mathcal{P}_D \setminus \mathcal{P}_C \cap \mathcal{P}_D) > 1$, then with probability 1/2 do one of the two following options:

1) pick random day for C from set $(\mathcal{P}_C \cup \mathcal{P}_D \setminus \mathcal{P}_C \cap \mathcal{P}_D) \cap \mathcal{P}_C$ and pick random day for D from set $(\mathcal{P}_C \cup \mathcal{P}_D \setminus \mathcal{P}_C \cap \mathcal{P}_D) \cap \mathcal{P}_D$ only if both of the sets are not empty; otherwise perform 2);

2) pick the one day from $\mathcal{P}_C \cap \mathcal{P}_D$ and perform the swap on this day;

iii. if $\text{card}(\mathcal{P}_C \cup \mathcal{P}_D \setminus \mathcal{P}_C \cap \mathcal{P}_D) = 0$, then it is evident that both cities C and D were served on the same day, and therefore pick one day from $\mathcal{P}_C \cap \mathcal{P}_D$ and perform the swap on this day.

STEP 5: Swap cities C and D if the swap would end up into a feasible schedule; otherwise do not swap cities C and D.

The *swap* operation thus returns a new schedule which is obtained from the inputted one by swapping two distinct, randomly chosen, cities.

Here we described two displacement operations which will play a crucial role in SA algorithm. Before we provide the modified SA for solving LBPMDMFVRPTW, it is necessary to describe an algorithm which checks whether an inputted schedule follows also capacity constraints. It is based on following steps:

STEP 1: Input a schedule.

STEP 2: For each day we examine all subroutes performed on that day.

STEP 3: If we find a city which would cost exceeding capacity of the garbage vehicle performing the subroute (let us for simplicity label such cities as culprits), we find the closest disposal for the predecessor of the culprit.

STEP 4: Now we have to distinguish two situations:

(a) if successor of the culprit is a disposal and succesor of the disposal is not a depot, we swap the culprit and the disposal; in terms of mathematics, if we define $succ(\alpha_i) = \alpha_{i+1}$, i. e. a mapping yielding for an i-th element of a sequence $(i+1)$-th element of the sequence, therefore in terms of newly introduced mapping let the culprit denote by ζ, then if $succ(\zeta) \in \mathcal{L}$ and $succ(succ(\zeta)) \notin \mathcal{D}$, we swap positions of ζ and $succ(\zeta)$;

(b) in other cases we insert the closest disposal right before the culprit ζ.

STEP 5: We repeat this process until we have no culprits in every subroutes each day.

We have presented a simple algorithm by which we obtain feasible scheduling in terms of capacity constraints. What is left, though, is an approach of checking whether also time constraints were fulfilled as well, and therefore we provide solution to this problem in following steps:

STEP 1: Input a schedule.

STEP 2: For each day we examine all subroutes performed by each vehicle on that day.

STEP 3: If we find a node which would exceed the permitted length of usage of vehicle k, we examine of what type this node is, i. e. whether it is depot, city or disposal, and we extract only cities.

STEP 4: Now, few sequences designed for vehicle k are considered:

(a) {..., city, disposal, **city**, disposal, depot}: if bolded city is the one causing exceeding the permitted length of a shift for vehicle k, then we randomly pick a schedule for this city and change its position according to the schedule by using above defined *change* operation and also erase the disposal which was on schedule after serving the bolded city;

(b) {..., city, **city**, disposal, depot}: in this case if the bolded city is the one causing exceeding the permitted length of a shift, we only change its position according to its, randomly chosen, schedule by using *change* operation.

This algorithm will eliminate many of routes inducing violation of shift length for each vehicle. It is though necessary to keep in mind, that the penalization would be of no use if the constraint constricting time of vehicle k spent en route would not be soft.

After presenting these four important methods there is one more thing to be described, and that is the evaluation function $f(\cdot)$ which in SA works as an objective function in optimization models. For the modified SA the function has two parameters: solution $\boldsymbol{x}$ and penalization constant θ. If after visiting some node was time of collection for any garbage vehicle exceeded, we penalize this overlap of permitted shift length until the waste collection on that given day is not done; and not penalize otherwise.

In this moment we have introduced everything essential for introducing the modified SA algorithm. Let us denote *change* operation algorithm by CHANGEPOSITION(), *swap* operation by SWAP(), the one for checking whether a route contains a city causing violation of vehicle capacity as CAPACITYCULPRITSEARCH() and the one for checking if time of usage of any vehicle was not exceeded by TIMECULPRITSEARCH(). All of these algorithms were implemented in programming language Python as well as the upcoming modified SA algorithm.

The description of the algorithm is following. As an input we insert an initial solution $x_{initial}$, declare it as our best solution, and set the maximal temperarute T_{max}. Then evaluate function $f(x_{initial})$ and declare it as our current solution and so far the best one as well. In each temperature of the process of cooling we randomly choose one city and randomly one of its schedules according to which should be performed waste collection to change its position by *change* operation. Hereafter we check if the obtained schedule does not violate for arbitrary vehicle its capacity and permitted length of a shift. Afterwards we randomly choose two distinct cities and perform *swap* operation. Now follows another check if there, in the gained schedule, are no violations of capacity and allowed time of usage of a vehicle. Further it follows the same steps until we reach minimal temperature T_{min} as in the algorithm presented in Subsection 2.3.3.

Below we provide the pseudocode which due splitting the crucial procedures into four distinct functions – at the first glance – seems similar to the basic version of SA but reader knows the complexity in cogs of the algorithm (see Algorithm 4 on page 36).

In this subsection we have presented metaheuristic approach as a substitution to the exact approach. This method, although not yielding analytic solutions, according to many articles (we refer to OSMAN, [24], DUECK, [32], or to GOLDEN ET AL. in *Fleet Management and Logistics*, [33]) returns satisfying results close to the optimum.

So far, we have proposed a formulation of MILP of LBPMDMFVRPTW and constructed modified SA. In this very moment we shall put both into practice.

Algorithm 4 Modfied simulated annealing for LBPMDMFVRPTW

T_{max}: initial temperature
T_{min}: minimal temperature
$T \leftarrow T_{max}$
k_B: Boltzmann constant
ε: random number from interval $(0, 1)$
. . . . . .
$x_{initial} \leftarrow$ localSearchSchedule()
$x_{best} \leftarrow x_{initial}$
$f_{best} \leftarrow f(x_{initial})$
$f_{current} \leftarrow f(x_{initial})$
while $T > T_{min}$ **do**
    $i \leftarrow$ getRandomCity()
    $s_i \leftarrow$ getRandomScheduleOfCity(i)
    $x_{next} \leftarrow$ changePosition(i, s_i, x_{best})
    $x_{next} \leftarrow$ timeCulpritSearch(x_{next})
    $x_{next} \leftarrow$ capacityCulpritSearch(x_{next})
    $j, l \leftarrow$ getTwoRandomCities()
    $x_{next} \leftarrow$ swap(j, l, x_{next})
    $x_{next} \leftarrow$ timeCulpritSearch(x_{next})
    $x_{next} \leftarrow$ culpritSearch(x_{next})
    $f_{next} \leftarrow f(x_{next})$
    $\Delta f \leftarrow f_{next} - f_{current}$
    if $\Delta f \leq 0$ **then**
        $x_{current} \leftarrow x_{next}$
        $f_{current} \leftarrow f_{next}$
    else if $\varepsilon < \mathbb{P}(\Delta f) = \exp\{\frac{-\Delta f}{k_B T}\}$ **then**
        $x_{current} \leftarrow x_{next}$
        $f_{current} \leftarrow f_{next}$
    end if
    if $f_{current} < f_{best}$ **then**
        $x_{best} \leftarrow x_{current}$
        $f_{best} \leftarrow f_{current}$
    end if
    $T \leftarrow$ cooling(T)
end while
return x_{best}

5. Case study

Due to the assignment we have composed a MILP model for solving LBPMDM-FVRPTW according to the formulation of the problem, both in Chapter 3. For it is NP-hard problem and cannot be solved in some reasonable time, in Chapter 4 we constructed the modified SA metaheuristic which follows the mathematical model but in algorithmical sense. In this chapter we will occupy ourselves with data offering information about waste collection from South Moravian region. Data are provided thanks to the team from *Institute of Process Engineering* located at *Faculty of Mechanical Engineering* at *University of technology* in Brno. Let us – for the beginning – introduce the data.

5.1 Data

We are in possession of dataset containing 2 depots, 60 areas and 5 disposals. Here we would like to underline, as we uttered in Chapter 3, that if an area is of larger size, then the area is split into subares, thus we end up with exactly 98 subareas in which we have to perform waste collection according to schedule of each one, and requested frequency as well. In every depot there are 3 garbage vehicles, each with its specific capacity and upper time limit for a collection. From the dataset we can also extract latitude and longitude for an arbitrary spot. In Figure 5.1 we provide map of Czechia with marked areas demanding waste collection. Further, we were given a time matrix providing any information

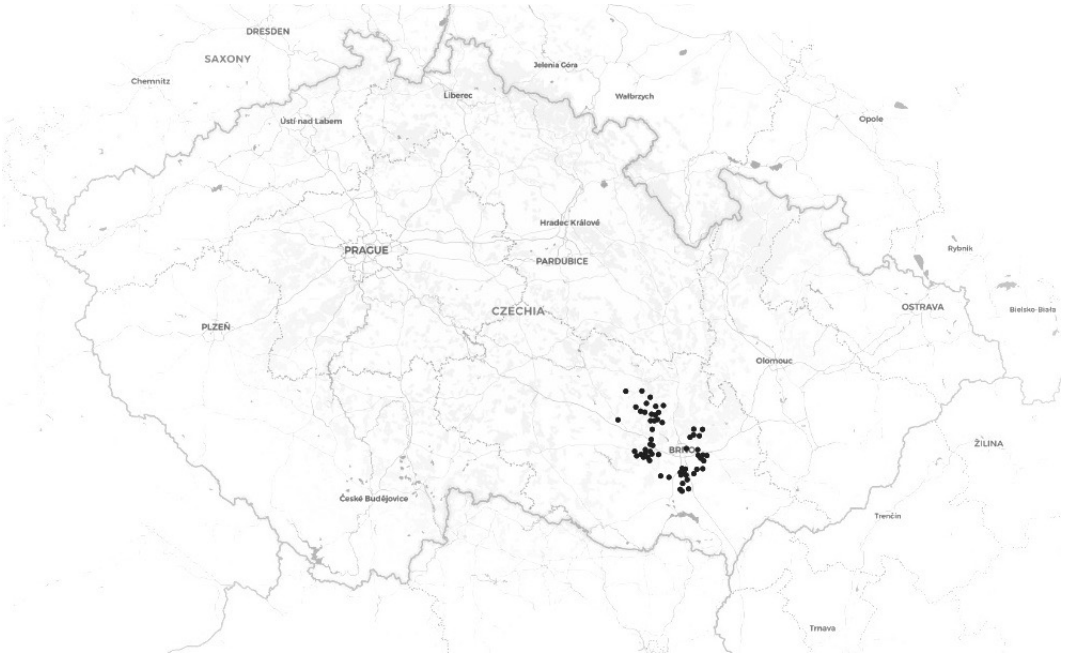

Figure 5.1: A map of the Czech Republic with the marked areas.

on the duration of transportation between cities, depots and disposals. The time matrix – with reference to the MILP model – represents cost in the objective function, $(t_{ij})_{i,j\in\mathcal{V}} = \mathbb{T}$; moreover holds $t_{ij} \neq t_{ji}$ for any $i,\ j \in \mathcal{V}; i \neq j$. Hereafter, the data contains expected value of the amount of waste to be collected for each

city and similarly it is the case with service time at any spot; the entries t_{ij} of matrix $\mathbb{T}$ as well.

Remark, even though we deal with subareas, further we stay faithful to the denomination we have been accustomed to on the previous pages, i. e. set $\mathcal{C}$ due the title *cities*.

5.2 Instances

One can certainly object that an instance of this dimension, which provides the dataset, cannot be solved analytically by using LBPMDMFVRPTW model, and that is true – it is enough if one calculates merely dimension of matrix $\mathbb{T}$ which equals to 105 and compares it with reasonable solving ability of the model around instances e. g. of type 2/7/1/2/2 (with reference to Section 4.1 in which we have established this denomination). Therefore, to be capable of a comparative analysis between LBPMDMFVRPTW model and the modified SA, we have extracted from the dataset reasonbale set of instances on which we can execute the model as well as the algorithm. The instances[1] are following:

	d / c / ℓ / k / $\lvert\mathcal{T}\rvert$
instance 1	1 / 3 / 1 / 1 / 1
instance 2	1 / 5 / 1 / 2 / 1
instance 3	1 / 6 / 1 / 2 / 1
instance 4	2 / 7 / 1 / 2 / 1
instance 5	2 / 8 / 1 / 2 / 1
instance 6	2 / 8 / 2 / 2 / 1
instance 7	2 / 9 / 2 / 2 / 1
instance 8	2 / 10 / 2 / 2 / 2
instance 9	2 / 11 / 2 / 2 / 2
instance 10	2 / 12 / 2 / 3 / 2
instance 11	2 / 15 / 3 / 3 / 3
instance 12	2 / 20 / 3 / 3 / 4
instance 13	2 / 25 / 4 / 3 / 4
instance 14	2 / 50 / 5 / 3 / 7
instance 15	2 / 98 / 5 / 3 / 7.

For each instance we will discuss its – if found – optimal value, i. e. total travel time for every garbage vehicle per whole time horizon, solving time until the optimal solution was found; in addition also evaluation which we obtain from the modified SA algorithm and also its solving time; the last thing we want to examine is to calculate so-called gap between an optimal solution and a solution yielded by the modified SA. For each instance, at the end of this chapter, we provide a table which contains all of the pieces of information here mentioned. Let us provide a comparison of the results, e. g. for instance 8.

[1] $d \in \mathcal{D}$, $c \in \mathcal{C}$, $\ell \in \mathcal{L}$, $k \in \mathcal{K}$, $|\mathcal{T}| = \mathrm{card}(\mathcal{T})$

Exact solution

In Figure 5.2 and 5.3 we provide illustrative graphs in which are marked optimal routes for instance 8. This instance is composed of 2 depots, 10 cities, 2 disposals, 2 vehicles in each depot and the time horizon is set to 2 days. The denomination in the graphs is same as we were accustomed to in previous chapters, i. e. $\mathcal{D} = \{d_1, d_2\}$, $\mathcal{C} = \{c_1, ..., c_{10}\}$ and $\mathcal{L} = \{\ell_1, \ell_2\}$. Let us denote set of garbage vehicles in depot d_i by $\mathcal{K}_{d_i}$ such that $\mathcal{K} = \cup_i \mathcal{K}_{d_i}$. According to the latest denomination, in instance 8 we get following sets: $\mathcal{K}_{d_1} = \{k_1, k_2\}$, $\mathcal{K}_{d_2} = \{k_3, k_4\}$. For better distinction, each vehicle is distinguished by using different arrow line styles, i. e. vehicle k_1 utilizes boldly dashed arrows, k_2 dashed arrows, k_3 solid arrows and k_4 dotted arrows. Each vehicle has same capacity and same permitted shift length which is equal to 9.5 hours. Each city has to be served only once and, in this case, none of these cities has any preference on which of these two days waste collection have to be performed.

Now, let us describe the optimal solution for instance 8. On the first day of the scheduling (see Figure 5.2) we can quickly notice that not all of the vehicles are used, just vehicles k_2, k_3 and k_4. Vehicle k_2 services cities c_1, c_2 and c_3 after leaving its depot and when the collection is done, it is forced to unload the cargo in the closest (in sense of time) disposal and afterwards return back to d_1.

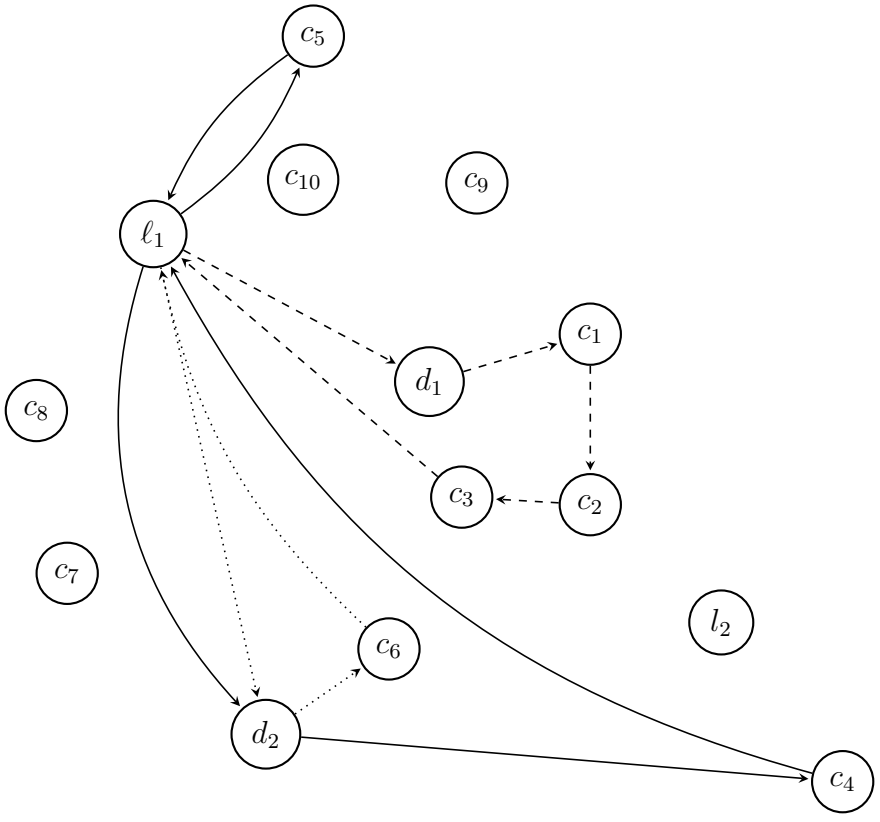

Figure 5.2: Optimal scheduling for instance 8 on for $t = 1$, $t \in \mathcal{T}$.

Vehicle k_3 starts its route from depot d_2 and serves city c_4, then empty itself at disposal ℓ_1 and continues to city c_5 which fills up the vehicle almost to its limit and hence it has to go to unload the waste again. Afterwards it travels back to depot d_2.

One could say that vehicle k_4 serves only city c_6, travels to disposal ℓ_1 and then back to its depot, i. e. d_2, but – according to the data – city c_6 requires whole capacity of k_4, service time in this city is almost 4 hours, further unloading the cargo in disposal ℓ_1 takes 30 minutes and if we add up travel time between the nodes, we get a shift of time length of more 6 hours.

One particular question comes to mind, and that is why, although in the instance we are allowed to use 2 disposals, all vehicles travels to disposal ℓ_1 even though in terms of distance is ℓ_2 closer. The reason is following: meanwhile ℓ_1 is a disposal based beyond municipality Tišnov (in which is based depot d_1), disposal ℓ_2 is based in Brno. According to the data it is way faster to travel from, e. g., node c_4 to disposal ℓ_1 than to drive through Brno. This could be explained by the fact that we were given only averaged values of travel time between nodes. Now, let us continue with the second day of the schedule.

On the second day (see Figure 5.3) we can observe that only two of the four vehicles are in use – vehicle k_1 from depot d_1 and vehicle k_3 from depot d_2.

Vehicle k_1 travels via cities c_9 and c_{10}, afterwards unloads the waste at disposal ℓ_1 and returns to d_1. Cities c_7 and c_8 are served by vehicle k_3, which then travels through disposal ℓ_1 back to its depot.

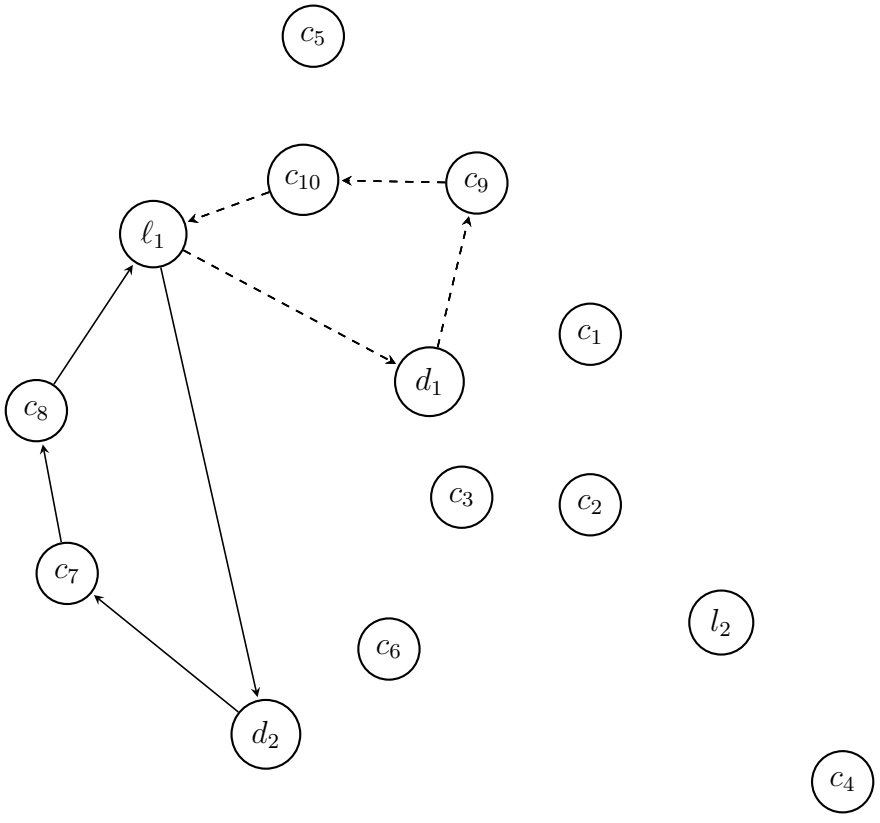

Figure 5.3: Optimal scheduling for instance 8 on for $t = 2$, $t \in \mathcal{T}$.

In total, objective function yielded for instance 8 value 33.83 hours. Decomposition into time duration of each shift is presented in Table 5.1.

Calculations were performed on a computer disposing by Intel® Core™ i5-5350U CPU @ 1.80 GHz and 8 GB RAM. GUROBI implemented in Python explored 2,115,292 nodes (53,020,017 simplex iterations) until the optimum was found.

Day	Vehicle	Shift duration (in hours)
1	k_2	7.41
1	k_3	6.24
1	k_4	7.74
2	k_1	6.01
2	k_3	6.43

Table 5.1: Duration of shift in instance 8 according to the optimal solution.

Time to get the optimum was 14,715.88 seconds which is more than 4 hours and 5 minutes. Now, let us compare this solution with the modified SA.

Heuristic solution

On the very same instance we wish to run the modified SA algorithm as described in Subsection 4.2.2, but first we have to obtain an initial solution which is acquired in the same manner as we proposed in Subsection 4.2.1. The initial solution for SA is depicted in Figures 5.4 and 5.5.

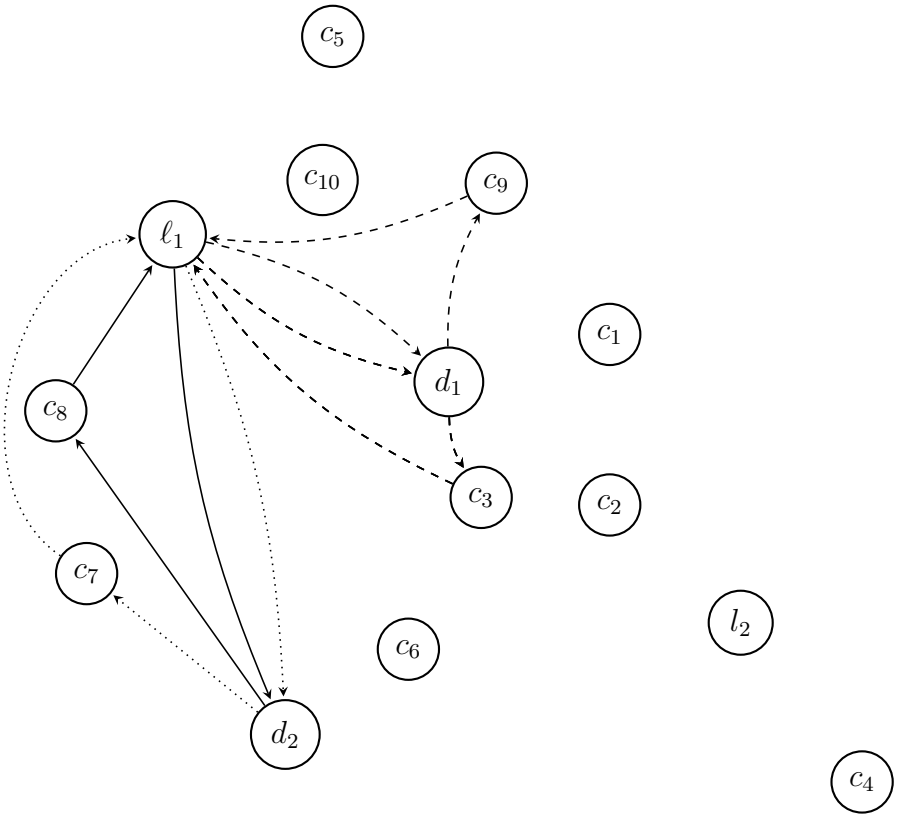

Figure 5.4: Initial solution of scheduling for instance 8 for $t = 1$, $t \in \mathcal{T}$.

From the figures we can utter that on both days of the scheduling are all of the four vehicles in use, and that – in fact – is natural according to the local search algorithm proposed in Subsection 4.2.1.

Let us take a look, e. g., at the second day of the scheduling (see Figure 5.5). From depot d_1 depart vehicles k_1 and k_2. Vehicle k_1, depicted by boldly dashed

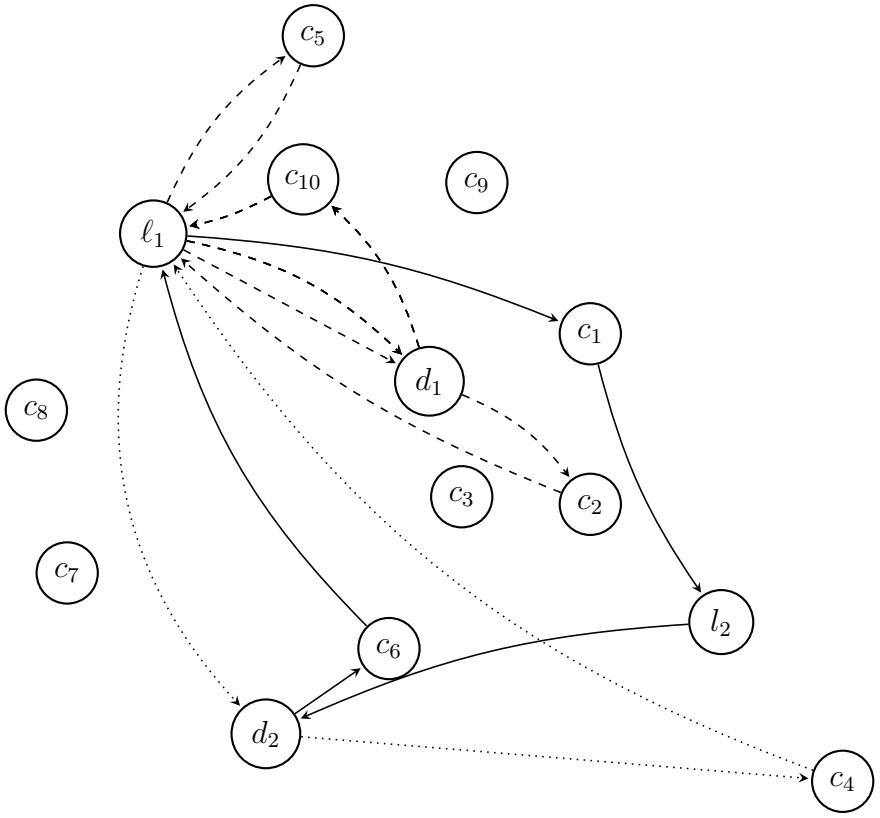

Figure 5.5: Initial solution of scheduling for instance 8 for $t = 2$, $t \in \mathcal{T}$.

arrow, travels to city c_{10}, performs waste collection and travels via disposal ℓ_1 to depot d_1; vehicle k_2 visits c_2 after which there is no space for another waste left, hence goes to disposal ℓ_1, then proceeds to city c_5 and before returning to depot d_1 unloads the waste at disposal d_1. From depot d_2 vehicle k_3 visits nodes c_6, ℓ_1, c_1 and at the end of its schedule unloads waste from c_1 at disposal ℓ_2. The remaining vehicle in depot d_2, vehicle k_4, serves city c_4 only and empties itself at disposal ℓ_1 in order to get back to the depot.

Evaluation of the initial solution is equal to 40.30 hours. Now let us seize this solution as a springboard for the modified SA.

After 12 seconds with 1087.76 iterations per second in average the evaluation function of the modified SA yielded a result equal to 34.47 hours. We have set in the modified SA parameters as follows:

$$T_{max} = 30\,000,\ \alpha = \{1/t\}_{t=1}^{n},\ n = 20\,000,$$

where cooling parameter is defined as a multiplicative inverse of number of the iteration we are at, i. e. at t-th iteration is temperature of the system equal to $(1/t) \cdot T_{max}$. These parameters were set according to behavior of the evaluation function when we were saving only the best solutions throughout the process of cooling. As we can see in Figure 5.6, sometimes we obtained better solutions also after $10,000^{\text{th}}$ iteration, scarcely after $15,000^{\text{th}}$ one, though, but for the sake of those cases, we extended the number of iterations to $20,000$.

The resulting schedules from the modified SA are depicted in Figures 5.7 and 5.8.

In the solution from the model are on the first day of scheduling served from

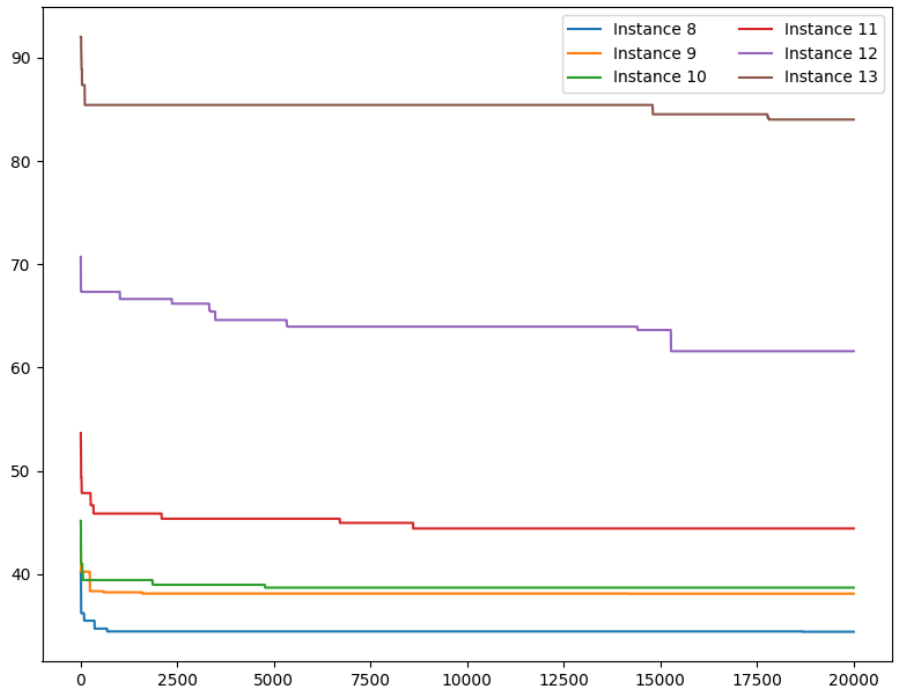

Figure 5.6: Best solution tracking from the evaluation function in the modified SA (shown only for instances 8–13).

depot d_1 cities c_1, c_2 and c_3, meanwhile in the solution obtained from the modified SA vehicle k_1 from depot d_1 collects waste also in city c_1, goes to disposal ℓ_1, then continues in cities c_2, c_9 and c_3 and aftewards travels to disposal ℓ_1; vehicle k_4 serves city c_4 as it was in the solution from the model on the same day and – in addition to that – takes care, after it unloads the cargo at disposal ℓ_1, of cities c_7 and c_8. On the second day are from depot d_1 by vehicle k_1 served cities c_{10} and c_5; from depot d_2 serves vehicle k_4 in only one city, city c_6.

As well as in the solution from the model, the modified SA also avoids the disposal ℓ_2 based in Brno due its possible traffic jams throughout the day. Lest we forget that we work with average travel time whose values can cause this avoidance. If we knew at which precise time of the day it would be less consuming to travel to disposal ℓ_2, the model, and also the heuristic, could yield better solutions.

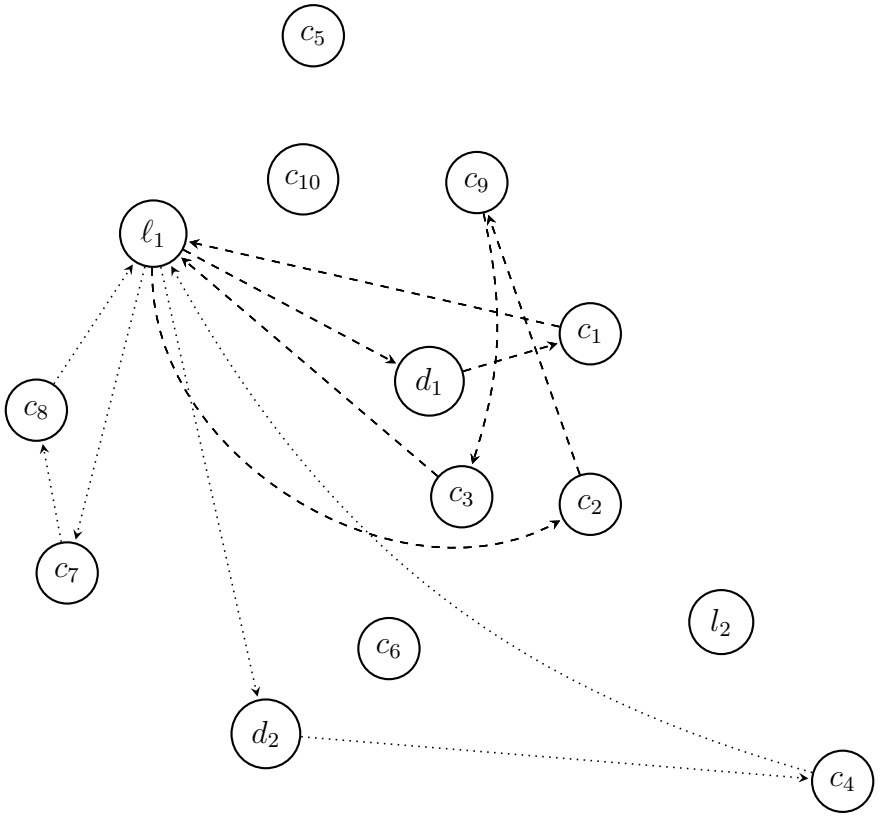

Figure 5.7: Scheduling for instance 8 according to modified SA, for $t = 1$, $t \in \mathcal{T}$.

The gap between the solution from the model and the one from the modified SA metaheuristic is 1.78 %. In Table 5.2 we provide, as same as it was it with the optimal solution, a table of shift durations in instance 8 according to the solution we obtained from the modified SA. Here, in comparison with Table 5.1, less vehicles were used but in exchange for the overtime caused by vehicle k_4 on the first day, as the permitted time of vehicle usage is 9.5 hours. Meanwhile the average shift length per vehicle in the optimal solution was 6.77 hours, the metaheuristic solution offers longer average shift length of a value of 8.53 hours.

Day	Vehicle	Shift duration (in hours)
1	k_1	8.91
1	k_4	6.24
2	k_1	9.17
2	k_4	9.78

Table 5.2: Duration of shift in instance 8 according to the solution yielded by the modified SA.

For more instances we provided table of results either for the model and the modified SA in Table 5.3. Let us briefly describe its values. The table is divided into three main columns - the first one is a column of instances and the two consecutive columns are devoted to the solutions from the model and from the modified SA, respectively.

In the column called *Model* we see four columns - *Optimal value, Incumbent,*

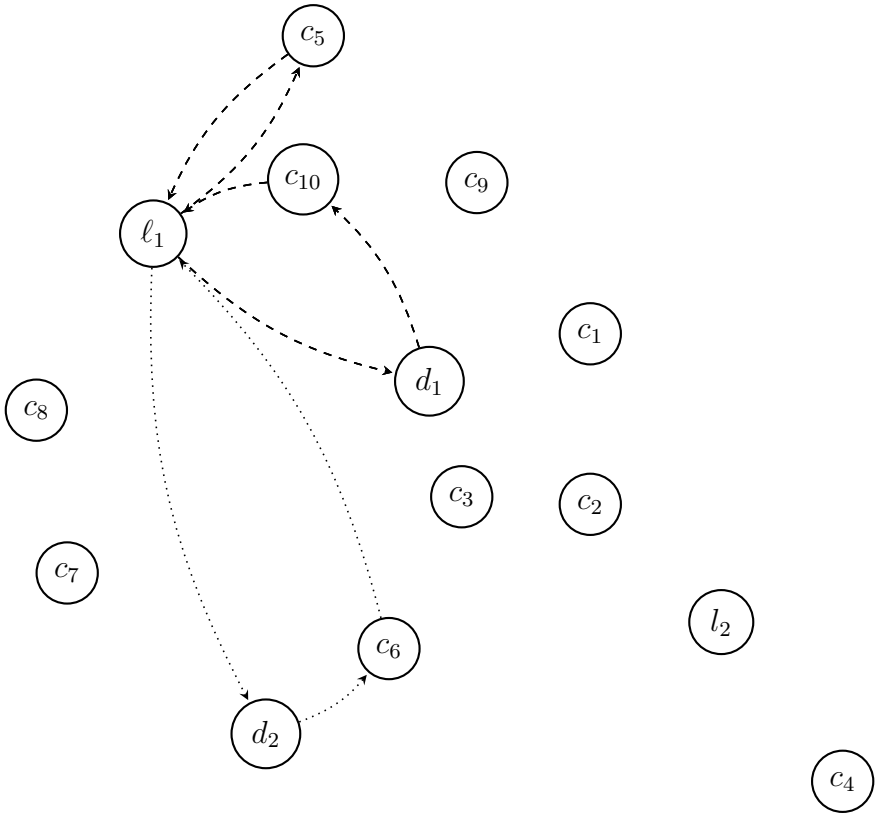

Figure 5.8: Scheduling for instance 8 according to modified SA, for $t = 2$, $t \in \mathcal{T}$.

Lower bound and *Time*. By *incumbent* is meant the best feasible solution so far at the specific time of solving. The difference between incumbent and lower bound is known as *gap*. If the gap is zero, the optimum is reached. Column *Time* contains information about solving time of each instance.

Column *Modified* SA reveals information about results yielded by the modified SA including also solving time of each instance; by column *Gap* we mean the gap between the optimal value from the model and the result obtained from the metaheuristic, if the optimal value was found; otherwise we calculate the gap w. r. t. the incumbent.

We also provide a table of improvements of the solutions obtained from the modified SA w. r. t. the local search algorithm proposed in Section 4.1 (see Table 5.4).

Instance	Model				Modified SA		
	Optimal value (hrs)	Incumbent (hrs)	Lower bound (hrs)	Time (s)	Evaluation (hrs)	Time (s)	Gap (%)
1 : 1/ 3/1/1/1	13.76	13.76	13.76	0.09	13.76	2.00	0.00
2 : 1/ 5/1/2/1	18.99	18.99	18.99	6.00	18.99	3.00	0.00
3 : 1/ 6/1/2/1	20.59	20.59	20.59	4.39	20.88	5.00	1.42
4 : 2/ 7/1/2/1	21.77	21.77	21.77	27.31	22.79	7.00	4.48
5 : 2/ 8/1/2/1	26.27	26.27	26.27	226.16	27.29	8.00	3.74
6 : 2/ 8/2/2/1	26.27	26.27	26.27	452.17	28.55	10.00	7.99
7 : 2/ 9/2/2/1	-	29.50	28.81	3600.00	30.69	12.00	3.88*
8 : 2/10/2/2/2	33.83	33.83	33.83	14 715.88	34.47	12.00	1.78
9 : 2/11/2/2/2	-	37.44	35.84	3600.00	38.01	12.00	1.51*
10 : 2/12/2/3/2	-	37.95	34.89	3600.00	38.68	23.00	1.85*
11 : 2/15/3/3/3	-	44.27	41.70	3600.00	44.34	29.00	0.17*
12 : 2/20/3/3/4	-	59.93	51.66	3600.00	61.59	66.00	2.69*
13 : 2/25/4/3/4	-	77.77	65.47	3600.00	77.88	72.00	0.68*
14 : 2/50/4/3/7	-	-	281.16	3600.00	193.52	625.00	-
15 : 2/98/5/3/7	-	-	529.42	3600.00	351.15	8955.00	-

Table 5.3: Comparison of results between the model and the modeified SA (*: gap calculated not with respect to the optimal value but w. r. t. incumbent value).

Instance	Local Search sol.	Modified SA sol.	Improvement (%)
1	15.37	13.76	10.48
2	18.99	18.99	0.00
3	23.51	20.88	11.22
4	24.23	22.79	1.46
5	27.82	27.29	1.91
6	29.08	28.55	1.82
7	32.55	30.69	5.72
8	40.30	34.44	14.54
9	43.39	38.01	12.39
10	45.18	38.68	14.40
11	53.68	44.34	17.38
12	70.72	61.59	12.91
13	92.02	77.88	15.36
14	211.68	193.52	8.58
15	458.67	351.15	23.44

Table 5.4: Improvement of the solution obtained from Local Search algorithm by the modified SA.

5.3 Results

Let us describe the results we have obtained in the previous section. As we can see, some of the instances were expensive in sense of time, and therefore we set the maximum solving time to 3,600 second at all but one case, and after this upper time limit we collected the results. We were examining 15 instances. In seven of the fifteen cases we reached an optimum and in the remaining ones we are left with the upper bound, i. e. incumbent (the far best solution), and the lower bound.

First, we take a glance at the instances in which we are in possession of an optimum which are instances 1, 2, 3, 4, 5, 6 and 8. The gap between an optimum and a metaheuristic solution is in average 2.77 %.

In cases, when we do not possess an optimum any rational relativity left was to calculate the gap with respect to the incumbent as it is the best solution at given time. Forget not that by metaheuristic solution getting close to the incumbent we do have any information about how exactly far we are from the optimum. If we analyze, for example, instance 13, the best solution found is 77.77 hours for four day schedule carried out by three vehicles – that yields a shift length of 6.48 hours (in case that all three vehicles are in use). The metaheuristic solution for instance 13 is not far from the incumbent, and that is 77.88 hours – only 0.68 % w. r. t. the incumbent, but that is the only information we can obtain from Table 5.3 about this instance. The lower bound at 3, 600 seconds is 65.47 hours, thus the gap between the incumbent and the lower bound equals to 15.8 %, but let us not to be misleaded that the metaheuristic solution is $15.8 + 0.68 = 16.48$ % far from the optimum. For this reason we had chosen calculating the gap w. r. t. to the incumbent, in cases of not having the optimum.

6. Future extensions

The objective of this work was to propose to the members from IPE a different approach to optimize waste collection of given municipalities in South Moravian region. We have proposed MILP formulation of this problem as well as the metaheuristic approach. In Section 5.3 we obtained average value of the gap between an optimum and a solution obtained from the modified SA, both run on the dataset provided by the members from IPE. The metaheuristic is working with two main displacement operations, *change* and *swap* operation (see Subsection 4.2.2). Other displacement operations could be taken into account, e. g. *reverse* operation which yields a mirrored order of the sequence, i. e. if we want reverse a sequence of elements $\{1, 2, 3, 4\}$, the reversed order would be evidently $\{4, 3, 2, 1\}$. In terms of VRP, the vehicle departs and returns to the same depot, thus we get that 1 and 4 represent the depot. One issue left would be to take care also of the disposal which happens to be represented by 3, the *reverse* operations hence should follow the feasibility and swap also 3 and 2 from the reversed sequence since the vehicle has to unload the cargo before returning to its depot. Other operation that may help to decrease the gap could be 3-swap, i. e. randomly choose three cities throughout the schedule and swap them.

With reference to the data, we were provided information about service time in any city or time length of unloading waste in any disposal; in addition to it, we also have an access to information about quantity of waste in each city. Nonetheless, we were given only slice of the real values in the form of its expected values. Let us recall one moment when we noticed the avoidance of one specific disposal in the optimal solution of instance 8. We believe that if one was working not with the average travel time but had an access to probability distribution of the travel time for each node, there could be an actual instant at which it would be more convenient to travel to the avoided disposal instead. These pieces of information can be handled by stochastic programming (we refer to short subsection in Chapter 1 devoted to nonlinear programming from which one can grasp easily grasp the idea behind the stochastic programming; see Subsection 1.2.1).

The travel time, as well as the service time, is not the only piece of information which can be modelled by stochastic programming. The speak is of demands of waste collection for each city. Although, we are not in possession of such data, let us propose a stochastic extension of LBPMDMFVRPTW with uncertain demand such that other parameters would be known in advance. For the purpose, here we remind, in (3.15), (3.16) and (3.17), the capacity constraints from the model:

$$w_{ikt} = 0 \qquad \forall\, i \in \mathcal{D} \cup \mathcal{L},\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T} \tag{3.15}$$

$$w_{ikt} - w_{jkt} + d_j \leq \mathbf{M}\,(1 - x_{ijkt}) \qquad \forall\, i \in \mathcal{V},\ \forall\, j \in \mathcal{D} \cup \mathcal{C},\ \forall\, k \in \mathcal{K}, \forall\, t \in \mathcal{T} \tag{3.16}$$

$$w_{ikt} \leq Q_k \qquad \forall\, i \in \mathcal{V},\ \forall\, k \in \mathcal{K},\ \forall\, t \in \mathcal{T}, \tag{3.17}$$

where w_{ikt} is defined as quantity of waste onboard of vehicle k right after servicing node i on day t such that the quantity cannot exceed capacity of vehicle k, i. e. Q_k. If we considered that demand of node i would be uncertain but its probability

distribution was known to us, the capacity constraints then would be reformulated by using joint probability constraints as

$$w_{ikt} \overset{a.s.}{=} 0 \qquad \forall\, i \in \mathcal{D} \cup \mathcal{L},\, \forall\, k \in \mathcal{K}, \; \forall\, t \in \mathcal{T} \tag{6.1}$$

$$w_{ikt}(\boldsymbol{\xi}) + d_j(\boldsymbol{\xi}) - w_{jkt}(\boldsymbol{\xi}) \overset{a.s.}{\leq} \mathbf{M}(1 - x_{ijkt}) \qquad \forall\, i \in \mathcal{V},\, \forall\, j \in \mathcal{D} \cup \mathcal{C}, \; \forall\, k \in \mathcal{K}, \forall\, t \in \mathcal{T} \tag{6.2}$$

$$\mathbb{P}\left(w_{ikt}(\boldsymbol{\xi}) \leq Q_k;\;\; \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K},\, \forall\, t \in \mathcal{T}\right) \geq \alpha. \tag{6.3}$$

For variable w_{ikt} is defined by demand $d_i(\boldsymbol{\xi})$ depending on random variable $\boldsymbol{\xi}$, therefore quantity of waste onboard of vehicle k after visiting node i on day t has to be also dependent on $\boldsymbol{\xi}$, and hence $w_{ikt}(\boldsymbol{\xi})$. Since we define variable $w_{ikt}(\boldsymbol{\xi})$ in (6.2), the inequality has to hold almost surely.

Let us further assume that $\boldsymbol{\xi}$ follows discrete probability distribution with finite support. Therefore $\boldsymbol{\xi}$ has S scenarios such that each scenario $\boldsymbol{\xi}^s$ will happen with probability $p^s \in [0, 1]$. We can reformulate constraints (6.1)–(6.3) by using another variable u^s such that

$$u^s = \begin{cases} 1: \text{ if } \boldsymbol{w} \text{ is feasible solution for sceanrio } s, \\ 0: \text{ otherwise.} \end{cases}$$

Hereafter, let us denote set of scenarios by $\mathcal{S} = \{1, ..., S\}$ Therefore the model LBPMDMFVRPTW, if we reformulated the capacity constraints, would look like as follows:

$$\min \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} (t_{ij} + T_i)\, x_{ijkt} + \theta \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} \sum_{t \in \mathcal{T}} g_{dkt} \tag{3.1}$$

$$\text{s. t. } (3.2) - (3.14)$$

$$w^s_{ikt} = 0 \qquad \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K}, \; \forall\, t \in \mathcal{T},\, \forall\, s \in \mathcal{S} \tag{6.4}$$

$$w^s_{ikt} + d^s_j - w^s_{jkt} - \mathbf{M}(1 - x_{ijkt}) \leq 0 \qquad \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K}, \; \forall\, t \in \mathcal{T},\, \forall\, s \in \mathcal{S} \tag{6.5}$$

$$w^s_{ikt} - Q_k - \mathbf{M}(1 - u^s) \leq 0 \qquad \forall\, i \in \mathcal{V},\, \forall\, k \in \mathcal{K}, \; \forall\, t \in \mathcal{T},\, \forall\, s \in \mathcal{S} \tag{6.6}$$

$$\sum_{s=1}^{S} p^s u^s \geq \alpha \tag{6.7}$$

$$u^s \in \{0, 1\} \qquad \forall\, s \in \mathcal{S} \tag{6.8}$$

$$(3.18) - (3.24),$$

where $\mathbf{M}$ has a role of activating the inequality whenever x_{ijkt} equals to one.

Here, we proposed some of the possible ways which can be undertaken to acquire better solutions for the South Moravian waste collection problem.

7. Epilog

In the work we – at first – presented some of the concepts of graph theory, nonlinear and also linear programming, as a special case of nonlinear programming, and – at last – we described primal simplex method to grasp an algorithmic approach of solving a linear program.

In Chapter 2 we introduced well-known problem in logistics, *The Vehicle Routing Problem* (VRP) and provided a mixed linear integer program for MDVRPTW with heterogeneous fleet as a springboard from which we could proceed to more complex formulations of VRP as we understood the idea on a simpler model. Later, we did a brief research on solving approaches to complicated VRPs and presented the main pillar of the work, simulated annealing algorihm.

We described the South Moravian waste collection problem in Chapter 3 and subsequently proposed its mathematical formulation in form of mixed linear integer program.

Since, generally, VRPs are very expensive – in terms of time – to solve, we proposed two solving approaches in Chapter 4 – an exact approach by solving the model from Chapter 3 and metaheuristic approach based on simulated annealing algorithm that was introduced in Chapter 2.

Chapter 5 was devoted to results obtained from the model and from the metaheuristic which both were performed on the data provided by the members from *Institute of Process Engineering*, located at *Faculty of Mechanical Engineering* at *University of technology* in Brno, and subsequently reported comparative analysis.

In the final chapter, Chapter 6, we stressed out some of the weaknesses of our approach and proposed possible remedies which could lead into better performances.

Bibliography

[1] Reinhard Diestel. *Graph Theory*. Springer-Verlag, New York, 2000. Electronic Edition.

[2] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[3] Sherali H. D. Bazaraa, M. S. and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, Ltd, United States of America, 2006.

[4] Bruce L. Miller and Harvey M. Wagner. Chance constrained programming with joint constraints. *Operations Research*, 13(6):930–945, 1965.

[5] A. Charnes, W. W. Cooper, and G. H. Symonds. Cost horizons and certainty equivalents: An approach to stochastic programming of heating oil. *Management Science*, 4(3):235–263, 1958.

[6] A. Prekopa. *On Probabilistic Constrained Programming*, pages 113–138. Princeton University Press, Princeton, 1971.

[7] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

[8] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2020.

[9] Paolo Toth and Danielle Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, 2002. ISBN: 0-89871-579-2.

[10] S. Geetha, P. T. Vanathi, and G. Poonthalir. Metaheuristic approach for the multi-depot vehicle routing problem. *Applied Artificial Intelligence*, 26(9):878–901, 2012.

[11] Jiří Kučera. *Modelování logistiky meziobecní přepravy odpadu (Modelling of logistics of inter-municipal waste transport)*. Brno, University of technology, 2022.

[12] Glenn Davis. A centroid for sections of a cube in a function space, with application to colorimetry, 2018.

[13] M. L. Fisher. Optimal solution of vehicle routing problems using minimum k-tree. *Operations Research*, 42:626–642, 1994.

[14] Ralphs T. K. Ladanyi, L. and L. E. Trotter Jr. Branch, cut, and price: Sequential and parallel, in computational combinatorial optimization. *Berlin: Springer*, 2001.

[15] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[16] Frank A. Tillman. The multiple terminal delivery problem with probabilistic demands. *Transportation Science*, 3(3):192–204, 1969.

[17] Billy E. Gillett and Jerry G. Johnson. Multi-terminal vehicle-dispatch algorithm. *Omega*, 4(6):711–718, 1976.

[18] Billy E. Gillett and Leland R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974.

[19] Anthony Wren and Alan Holliday. Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Research Quarterly (1970-1977)*, 23(3):333–344, 1972.

[20] Shen Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965.

[21] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–680, 06 1983.

[22] Metropolis NS, A.W. Rosenbluth, M.N. Rosenbluth, AH Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 01 1953.

[23] G. Wedler and H.-J. Freund. *Lehrbuch der Physikalischen Chemie*. 6th Ed. Weinheim: WILEY-VCH, 2012.

[24] Ibrahim Osman. Meta-strategy simulated annealing and tabu search algorithms for the vehicle routine problem. *Annals of Operations Research*, 41:421–451, 12 1993.

[25] Erdal Aydemir and Kenan Karagul. Solving a periodic capacitated vehicle routing problem using simulated annealing algorithm for a manufacturing company. *Brazilian Journal of Operations and Production Management*, 17, 02 2020.

[26] Chao Wang, Fu Zhao, Dong Mu, and John W. Sutherland. Simulated annealing for a vehicle routing problem with simultaneous pickup-delivery and time windows. AICT-415(Part II):170–177, September 2013. Part III: Sustainable Services.

[27] Vladimír Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[28] Vít Procházka. *Pokročilé optimalizační modely v odpadovém hospodářství (Advanced Optimization Models in Waste Management)*. Brno, University of technology, 2014.

[29] Vlastimír Nevrlý. *Modely a metody pro svozové úlohy (Models and methods for routing problems)*. Brno, University of technology, 2016.

[30] N. Christofides and J. E. Beasley. The period routing problem. *Networks*, 14(2):237–256, 1984.

[31] B Meindl and Matthias Templ. Analysis of commercial and free and open source solvers for linear optimization problems. 08 2013.

[32] Gunter Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993.

[33] Bruce L. Golden, Edward A. Wasil, James P. Kelly, and I-Ming Chao. *The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results*, pages 33–56. Springer US, Boston, MA, 1998.

List of Figures

List of Tables