TECHNISCHE UNIVERSITÄT BERLIN
FAKULTÄT IV ELEKTROTECHNIK UND INFORMATIK
BACHELORSTUDIENGANG INFORMATIK

FERNANDO BOMBARDELLI DA SILVA

Solving The Dial-a-Ride Problem With The Firefly Metaheuristic

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Dr.-Ing. Axel Heßler
Reviewer: Prof. Dr. Dr. h.c. Sahin Albayrak
Reviewer: Prof. Dr. habil. Odej Kao

Berlin
February 2016

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den

.....
Unterschrift

ABSTRACT

The Dial-a-Ride problem (DARP) is an $\mathcal{NP}$-hard combinatorial optimization problem with practical applications in user-oriented public transportation. The DARP is a vehicle routing problem whose input consists of a set of vehicles and a set of pick-up and drop-off requests from passengers. The goal is to assign the requests to the vehicles and to calculate the routes of each one of them, minimizing the operation costs and ensuring that all the constraints, such as vehicle capacity, departure and arrival time windows, and maximal user ride time are fulfilled. This work presents a mathematical formulation of the problem and seeks to solve it through the firefly algorithm (FA). The FA is a novel nature-based metaheuristic inspired by the behavior of fireflies, which applies the concept of swarm intelligence in order to optimize mathematical functions by looking for near-optimal solutions. With the aid of this technique we aim to model and implement a solver to the DARP which explores the huge combinatorial search space, generated by the problem instances, in an efficient way. Finally, a performance comparison between the proposed method and the algorithms found in the scientific literature will point out to what extent our implementation of the FA outperforms other approaches.

Keywords: Dial-a-ride problem. Firefly algorithm. Metaheuristic. Integer programming. Vehicle routing.

Lösung des Dial-a-Ride-Problems mit der Firefly-Metaheuristik

ZUSAMMENFASSUNG

Das Dial-a-Ride-Problem ist ein $\mathcal{NP}$-schweres kombinatorisches Optimierungsproblem, das auf benutzerorientierten öffentlichen Personenverkehr Anwendung findet. Das DARP ist ein Tourenplanungsproblem, deren Eingabe aus einer Menge von Fahrzeugen und einer Menge von aus Fahrgästen eingetragenen Anträgen zum Abholen und zum Absetzen besteht. Das Ziel ist, den Fahrzeugen die Anträge zuzuweisen und die Routen jeden Wagen zu berechnen, sodass die Betriebskosten minimiert werden und jede Beschränkung, so wie Fahrzeugnutzlast, Zeitfester für Ein- und Ausstieg, maximale Fahrdauer, erfüllt wird. Diese Arbeit stellt eine mathematische Formulierung des Problems vor und versucht, es durch den Einsatz des Firefly-Algorithmus (FA) zu lösen. Der FA ist eine neue naturbasierte Metaheuristik, die sich vom Verhalten von Glühwürmchen inspiriert und die das Konzept von Schwarmintelligenz anwendet, um mathematische Funktionen zu optimieren, indem sie auf quasi-optimale Lösungen sucht. Anhand dieses Verfahrens wollen wir einen Löser des DARP modellieren und dann implementieren, der den riesigen kombinatorischen von Problemintanzen generierten Suchraum effizienterweise erforscht. Letztendlich weist ein durchgeführter Leistungsvergleich zwischen der vorgeschlagen Methode und den Algorithmen aus der wissenschaftlichen Literatur darauf hin, inwiefern unsere Implementierung des FA andere Ansätze an Leistung übertrifft.

Schlagwörter: Dial-a-ride-Problem. Firefly-Algorithmus. Metaheuristik. Ganzzahlige Optimierung. Tourenplanung.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AMPL	A Mathematical Programming Language
CPU	Central Processing Unit
DARP	Dial-a-Ride Problem
FA	Firefly Algorithm
GA	Genetic Algorithms
GLPK	GNU Linear Programming Kit
MD-H-DARP	Multi-Depot Heterogeneous Dial-A-Ride Problem
PDVRP	Pickup and Delivery Vehicle Routing Problem
PSO	Particle Swarm Optimization
RAM	Random Access Memory
VRPTW	Vehicle Routing Problem with Time Windows

CONTENTS

1 INTRODUCTION

In current urban areas, mainly in very populated cities, there is a huge number of mobility problems. These problems have been seen with much interest by the scientific community, which has been strongly contributing to the improvement of transportation and logistic networks, thus promoting advances towards a better urban mobility.

The here addressed problem is known as the **Dial-a-ride problem (DARP)**. Shortly, it consists of a system with a set of requests of pick-up and delivery entered by customers and a fleet of vehicles. The goal is planning the route of the vehicles and the assignment of requests to them in a feasible way, since there are constraints from both the requests and the vehicles to which a solution is subject. These can be several conditions, like location and time limit for picking up and delivering or how much time each vehicle can operate. Besides, it is not only searched a feasible solution but an optimal one that minimizes the operation costs, here seen as the total time of operation, defined by a function.

Many difficulties are found when trying to solve the described problem, the combinatorial nature of its solution space make it hard to treat for large inputs because of the high complexity, it leads then to a special difficulty in building a scalable application.

1.1 Definition

Cordeau and Laporte (2007, p. 29) states the problem very briefly:

> "The Dial-a-Ride Problem (DARP) consists of designing vehicle routes and schedules for n users who specify pick-up and delivery requests between origins and destinations. The aim is to plan a set of m minimum cost vehicle routes capable of accommodating as many users as possible, under a set of constraints."

The author summarizes very well, however, a closer look is to be taken. The instances of the problem specify the following properties, that are taken as input by the solver. Firstly, there is an **amount of homogeneous vehicles**, that is to say, no distinction should be made concerning types of buses. Secondly, it is assumed a **single depot**, which the vehicles start from and which they arrive to. Next, the cars have the attributes of **capacity**, in other words, how many passengers it can carry at a time, and its **maximal duration time** that the car cannot exceed. Then, the passengers have a **maximum allowed travel time** to guarantee their comfort. Finally, there are n **requests**, which have these features:

- Pick-up location;
- Destination location;
- Time window for picking up (a time interval in which the vehicle is expected to be at the site);
- Time window for delivering;
- Time needed for boarding or alighting;
- Quantity of passengers;

Additionally, it is possible for a car to move from every location to any other location, the time needed to travel between any pair is considered to be the linear distance between these two points.

To sum up, these are the parameters that form the constraints of the problem and must be considered. In regard to the goal, there are two concepts that can be elucidated, namely feasible and optimal solution. The former is defined thusly:

Definition 1 (Feasible solution)**.** *Determination of the routes of the vehicles, such that, every request is picked up and then delivered by one and only one vehicle, which fulfill every condition of the requests and of itself.*

In order to make it clearer, let a route be defined as:

Definition 2 (Route)**.** *The order of the locations through which a vehicle passes. It starts and ends at the depot and it is possible to be executed without breaking the conditions of time windows of the respective requests considering the boarding, alighting and travel times.*

Every route has then attached a duration time, which can be calculated based on its requests. Summing up the duration of every route in a solution, we get its total duration time. Therewith is an optimal solution defined:

Definition 3 (Optimal solution)**.** *A feasible solution whose duration time is less than or equal to any other feasible solution's duration time.*

In conclusion, given an instance of the Dial-a-ride problem, it is sought an optimal solution.

1.2 Approach

In this work two approaches shall be taken, in order to solve the optimization problem. The exact approach consists of mathematically modeling the problem as an **integer linear program** and then transcribing it into a **mathematical programming language** aiming executing it with a generic linear program solver. The near-optimal approach shall then analyze and develop a program that implements the **firefly metaheuristic** to efficiently seek near-optimal solutions.

Finally, the goal of the research is to compare the performance of the two different models, both through the execution time and the solution quality as well as through the possibility of dealing with large inputs. Besides the evaluation through the the mentioned comparison, a second evaluation is conducted by assessing the firefly metaheuristic implementation when solving the instances used in other works of the literature.

1.3 Justification

It is expected that the results of this work may bring relevant contributions to the handling of the presented problem, and even of other ones. By having two distinct approaches it is possible to compare results regarding important features of the problem, such as scalability, feasibility and deviation to an optimal solution. Furthermore, the application of the relatively new firefly algorithm to the problem can show how it performs in a such a solution space.

In addition to the contributions to the understanding of the behavior of swarm metaheuristics applied to optimization problems of transportation, the new procedure of solving the problem serves as a prototype and brings a new perspective to commercial applications which seek constantly to treat the problem in a more efficient and scalable way.

With regard to the current cities' mobility, there is a growing demand for an efficient alternative to the classical means of transportation. The implementation of such a system improves the possibility of movement of the population and makes it more efficient, since it allows the decrease of the number of cars that drive through the urban network everyday causing traffic jams in big cities. Moreover, it helps to solve a demanding problem in today's society where there is an increasing number of elderly or handicapped people, who have the right to mobility and need assistance to travel in the

town.

Also in an economical view this research contributes to the win of new markets by companies who aim to enter the branch of public transportation since its main goal is minimizing the operation costs. With a business model based on the reduction of transaction costs, a company can take great advantages against competitors in order to capture marketplace.

At last, the concerns of the proposed model shows also an ecological relevance by enabling the decrease of the emission of greenhouse gases in the urban area, directly, considering that in this case costs are direct proportional to the consume of fuel, and consequently to the emission of gases, such as CO_2, and indirectly by reducing the circulation of other automobiles.

2 LITERATURE REVIEW

According to Cordeau and Laporte (2007, p. 30), the Dial-a-ride Problem (DARP) is very similar to other problems studied in the scientific society, namely the *Pickup and Delivery Vehicle Routing Problem* (PDVRP) and the *Vehicle Routing Problem with Time Windows* (VRPTW), problems that have application in logistics. The authors point out that, what basically differs the DARP from these other ones is the human perspective, by the fact that people are transported. It often appears presenting two goals, minimizing operation costs subject to the constraints and maximizing the availability and quality of the service. The quality criteria include frequently aspects like route duration, customer waiting and ride time, maximum vehicle ride time, among others, and are usually treated as the constraints of the optimization problem.

Cordeau and Laporte (2007) realized a survey on the subject, they show different variations in the formulation and in the approach. Firstly, the DARP occurs in a **static** version in which the requests are known beforehand, this one we would like to treat here, alternatively, there is a **dynamic** version, dealt by Berbeglia, Cordeau, and Laporte (2010), in which requests can be entered during the operation of the transportation service.

Another distinction to be made is between the **homogeneous** and the **heterogeneous** Dial-a-ride Problem. The former proceeds on the assumption that every vehicle is equal. The latter on the other hand distinguish the vehicles either in capacity or in new features, such as space for wheelchair, etc. (Parragh et al. 2012, p. 593). In the same way, the objectives can differ by aiming **cost minimization** or **satisfied demand maximization** (Cordeau and Laporte 2007, p. 30). Urra, Cubillos, and Cabrera-Paniagua (2015) have succeeded on handling the second one. A last differentiation of the problem is done regarding the depots, which can be a **single** one or **multiple** ones, in this case referred to as *Multi-Depot Heterogeneous Dial-A-Ride Problem* (MD-H-DARP) (Braekers, Caris, and Janssens 2014, p. 166).

In their survey, Cordeau and Laporte (2007) identify in the literature three formulations. Two of them are described as a mathematical linear program and the other one as a scheduling problem. The **three-index mathematical formulation**, proposed by Cordeau (2006), uses binary three-index variables $x_{i,j}^k$ to assign routes to each bus and then to minimize costs. In addition, Ropke, Cordeau, and Laporte (2007) come up with a **two-index formulation** which ignores the maximum ride constraints in order to simplify the model.

Exact algorithms have been proposed by several researchers, to highlight are the **branch-and-cut** by Cordeau (2006) and by Ropke, Cordeau, and Laporte (2007). However, to speed up running times many other methods based on **metaheuristics** have been suggested in the literature, among them are the **genetic algorithms** by Jorgensen, Larsen, and Bergvinsdottir (2007), the **simulated annealing** by Zidi et al. (2012), the **granular tabu search** by Kirchler and Wolfler Calvo (2013) and the approach with **large neighborhood search** performed by Parragh and Schmid (2013). Lately, Urra, Cubillos, and Cabrera-Paniagua (2015) published a new method with the so-called **hyperheuristic**.

All in all, none of the studied methods takes a swarm-based metaheuristic approach to solve the problem. Swarm intelligence is a technique applied in the computer science, more precisely in artificial intelligence and operations research, that is based on observations of nature patterns and behaviors, *particle swarm optimization* (PSO), *ant colony* and the *firefly algorithm* (FA) are example of techniques that apply these ideas. In this point of view, these metaheuristics resemble the *genetic algorithms* (GA), since GAs are also nature-based, but they differ in the fact that, genetic algorithms have mutation and crossover operators and are based on the theory of evolution of the species, whereas swarm intelligence techniques are based purely on the behavior of swarms (Yang 2012, p. 189-190).

In this work we apply the **firefly metaheuristic**, that has been showing good results in the solution of nonlinear global optimization problems. It was introduced by Yang (2009), who compares it against the PSO by running simulations in a variety of objective functions and concludes affirming that the FA can outperform the PSO and that it is potentially more powerful in solving $\mathcal{NP}$-hard problems. Additionally, Yang (2012) presents a theoretical analysis on swarm intelligence having as study cases the firefly algorithm and particle swarm optimization. Yang and He (2013) introduce the FA by approaching parameter settings, complexity and applications with examples, at the end he draws a conclusion showing a growing application of the method in the scientific community and foreseeing an expansion of the subject and the improvement of the metaheuristic.

The firefly algorithm has also been applied to discrete optimization problems, also known as **integer programming**. Jati and Suyanto (2011) address the classical **Travelling Salesman Problem** with the help of the FA. In the same way, Apostolopoulos and Vlachos (2010) deal with another combinatorial optimization problem, the *Economic Emissions Load Dispatch Problem*. At last, Sayadi, Ramezanian, and Ghaffari-Nasab (2010) and Sayadi, Hafezalkotob, and Naini (2013) also utilize this technique, in order to

solve other $\mathcal{NP}$-hard problems.

2.1 Instances

A set of benchmark instances were created by Cordeau (2006), these ones have been extensively used in other works in the literature for the DARP. Therefore, they constitute the instance dataset for this work. Originally, Cordeau (2006) came up with a collection of 12 random generated problem instances which were lately expanded by Ropke, Cordeau, and Laporte (2007) who proposed 12 new larger and harder to solve ones. The same authors provided the optimal solutions for the whole dataset, whereas Parragh and Schmid (2013) presented the state-of-the-art heuristic approach.

The instances are described in text files which describe the parameters of the problem. The number of requests varies from 16 until 96 and contain always the load of one passenger, it means, it is assumed one passenger per request. Similarly, the boarding time is considered to be 3 units of time for every request. The number of vehicles varies as well, between 2 and 8 cars which have a capacity of 3 passengers in every case.

The locations where the requests are to be served are randomly generated, according to a uniform distribution, in a plan space inside the range of [-10, 10] x [-10, 10]. The depot is set in the center of the area, that is, the origin (0,0). The costs of traveling between any two stops is equals to the travel time between them which is assumed to be simply the *Euclidean distance* between the two points.

The number of requests is always even and can be divided into two groups, while a half of them are **outbound** requests the other half are **inbound** requests. The former means that the constraint of time window exists only at the arrival time at the destination, the latter, on the other hand, has this kind of constraints only at the departure time. To better understand, one can cite the example from Cordeau and Laporte (2007, p. 29), they compare it with the case of patients who need transportation from home to the hospital, they suggest that the time window constraint should be set in order to come to the hospital on time, that would be the so-called outbound request, likewise, patients should be picked up from the hospital in a specific time interval, that would then be an inbound request.

In the dataset considered here the time window at the arrival at the destiny $[e_{n+i}, l_{n+i}]$ for an outbound request i comprehend a 15 minute interval. So, let T be the maximal route time allowed for the buses, with it, l_{n+i} is randomly picked from the interval [60, T] and, as a consequence, e_{n+i} is calculated by subtracting 15 from the upper limit, i.e.

$e_{n+i} = l_{n+i} - 15$. Similarly, for an inbound request i, e_i is chosen from the interval [0, T-60] and then $l_i = e_i + 15$.

Lastly, the maximal route time T, also referred to as planning horizon, varies between 240 and 720 and the maximal ride time L of a passenger is constantly 30 for every instance.

The Table 2.1 lists all the instances along with their names and properties. The columns *Request* and *Buses* show the quantity of each of these attributes, the column *Capacity* tells the maximal load of the buses. The *Optimum* is the cost of the minimal solution which has been calculated by Ropke, Cordeau, and Laporte (2007, p. 270). At last, the column *CPU*[1] refers to the processing time in minutes provided by the metaheuristic implementation with *variable neighborhood search* from **parragh_introducing_2011** (executed on a 3.2 GHz *Intel Pentium D* CPU with 4 GB of RAM) while *CPU*[2] refers to the running time in minutes of the state-of-the-art heuristic implementation with *large neighborhood search* by Parragh and Schmid (2013) (executed on an *Intel Xeon* CPU at 2.67 GHz with 24 GB of RAM).

Table 2.1: Characteristics of the problem instances

Instance	Requests	Buses	Capacity	Optimum	CPU[1]	CPU[2]
a2-16	16	2	3	294.25	68.2	0.12
a2-20	20	2	3	344.83	133.8	0.28
a2-24	24	2	3	431.12	187.8	0.35
a3-18	18	3	3	300.48	45.4	-
a3-24	24	3	3	344.83	86.8	0.29
a3-30	30	3	3	494.85	105.6	0.50
a3-36	36	3	3	583.19	162.6	0.83
a4-16	16	4	3	282.68	26.0	-
a4-24	24	4	3	375.02	50.8	-
a4-32	32	4	3	485.50	86.0	0.55
a4-40	40	4	3	557.69	130.6	0.78
a4-48	48	4	3	668.82	253.8	1.62
a5-40	40	5	3	498.41	-	0.85
a5-50	50	5	3	686.62	-	1.60
a5-60	60	5	3	808.42	-	2.51
a6-48	48	6	3	604.12	-	1.14
a6-60	60	6	3	819.25	-	2.29
a6-72	72	6	3	916.05	-	4.43
a7-56	56	7	3	724.04	-	1.67
a7-70	70	7	3	889.12	-	2.88
a7-84	84	7	3	1033.37	-	7.04
a8-64	64	8	3	747.46	-	2.14
a8-80	80	8	3	945.73	-	5.73
a8-96	96	8	3	1232.61	-	9.92

Source: (**parragh_introducing_2011**) and (Parragh and Schmid 2013, p. 496)

3 THEORETICAL BASIS

This chapter presents a theoretical framework that configure a background necessary to understand, analyze, design and implement the solution for the addressed problem.

3.1 Linear and Integer Programming

Linear Programming is a technique in mathematics used for **optimization of linear functions**. With it, it is possible to model optimization problems in which a cost (or utility) linear function is to be minimized (or maximized) given a set of constraints which the function's independent variables are subject to. In short, it enables the formulation of optimization problems in a very simple mathematical language.

To formulate a linear program three specifications are required. At first, the so-called decision variables x_i, that are the ones which values are assigned to. Secondly, the objective function, that is a linear function $f(x)$ to be optimized. Lastly, the constraints, that are linear inequalities describing relations between the decision variables and parameters of the problem instance. Therefore, it is expected from a solver, given a linear program and data parameters, to deliver the set of assignments of values to the decision variables that minimize or maximize the objective function respecting the constraints. The most prominent method for solving linear programming is the **simplex method** (Shenoy 2007, p. 5-6).

As a result, a linear program with n variables and m constraints for the minimization of a function can be written as follows.

$$\begin{aligned} \textbf{minimize} \quad & \sum_{i=1}^{n} c_i x_i & \\ \textbf{subject to} \quad & \sum_{i=1}^{n} a_{ki} x_i \leq b_m & \forall k = 1, 2, ..., m \\ & x_i \geq 0 & \forall i = 1, 2, ..., n \end{aligned}$$

However, there are problems where real values are not accepted as assignment to variables, because they require integer values. These problem are to be treated differently by adding an additional integer-value constraint to the domain of the variables ($x_i \in \mathbb{Z}$). A technique derived from linear programming addresses the issue, namely **integer pro-**

gramming. Although there are other more specific concepts for problems that are similar to integer programming, such as mixed integer programming and zero-one programming, they all can be handled by quite the same principles (Shenoy 2007, p. 175). As an implication, the simplex method is not able to solve this type of optimization problem neither is, generally, numerical approximation a good solution. Thus, there are several algorithm in the literature, for instance the **branch-and-bound** and the **cutting planes** are widespread and one of the most common.

The integer-value constraint implies a vast difference regarding the complexity of the integer problem when compared to the linear one. It is then discussed, whether these problems can be solved in polynomial time, referred to as *certificate of optimality*. Wolsey and Nemhauser (2014) point out that there are algorithms capable of solving linear programming in polynomial time, whereas no such algorithm is known to integer programming. Moreover, many known $\mathcal{NP}$-complete problems, such as the TSP and the knapsack problem, can be reduced to an instance of integer programming. It turns out that integer programming is not only $\mathcal{NP}$-hard but $\mathcal{NP}$-complete as well (Schrijver 1998, p. 21).

3.2 Metaheuristics

osman_meta-heuristics_2012 state in their work: “Meta-heuristics are the most recent development in approximate search methods for solving complex optimization problems.”. As he follows, a metaheuristic guides the development of algorithms using artificial intelligence, biological, physical, mathematical and natural phenomena as source of inspiration. Firstly should be understood what heuristics are. According to Carvalho, Savransky, and Wei (2004, p. 13), there is no common definition for heuristics. They suggest that: “Heuristics can be rules, strategies, principles or methods for increasing the effectiveness of a problem resolution [...]. [They] neither provide direct and definite answers, nor guaranty a solution for a problem.”. On the whole, heuristics are a powerful weapon to attack difficult combinatorial optimization problems.

On metaheuristics the concept is concise, **osman_meta-heuristics_2012** explain in the following way.

> “A Meta-heuristic is an interactive generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space using learning strategies to structure information in order to find efficiently near-optimal solutions.”

Therefore, metaheuristics do not necessarily deliver an optimal solution, but the

benefit in terms of performance overcome the possibility of not obtaining a proved optimal solution. Even though, the late advances in the field have contributed strongly in order to get very good in quality results (ibid.).

As a metaheuristc gives quasi instructions to write an algorithm, it is common to almost every technique four basic elements which should be modeled. First, a **solution representation** is to be determined, this should be able to describe any possible solution and to be evaluated by a function which determines its cost or benefit. Secondly, the algorithm should **initialize solutions**, it means, there is a procedure to create the so-called initial solutions. Thirdly, **new solutions** should be generated from existing ones, in some metaheuristics this step is called *movement*, *genetic operator*, among other terms. Finally, a **stop criterion** is established in order to determine the termination of the method.

The metaheuristics can be classified in two types, the ones based on **local search** (or neighborhood search) and the others based on **nature observations**. The former explores the search space in search of an optimal solution by generating new solutions that lie near the current analyzed one, it then requires a move mechanism besides a *selection* and a *acceptance* criteria, which decide whether a new generated solution is approved in order to progress with the iterations. Whereas simpler methods, such as *variable neighbohood search*, have the drawback of tending to a local optimum, others, like *simulated annealing*, *tabu search* and *GRASP*, aim to workaround the issue. In contrast, the nature-inspired metaheuristcs are generally based on the behavior of populations of living beings and how the its members interact. They are characterized by having a set of solutions, instead of only one, with which local searches are performed in different areas of the search space (**osman_meta-heuristics_2012**). Examples of this kind of method are the *genetic algorithms*, *particle swarm optimization*, *ant colony* and the *firefly algorithm*.

3.3 The Firefly Algorithm

The **firefly algorithm** (FA), or firefly metaheuristic, was brought forth by Yang (2009). It is a **nature-inspired metaheuristic** based on observations of fireflies populations that resembles other techniques, such as the *particle swarm optimization* and the *bacterial foraging algorithm*. The following explanations relies on the work of (ibid.).

A firefly is a flying insect that emits flashes of light from its tail. This luminous signal serves then as a mean of communication between these animals, that is used for sexual attraction and selection. So, the main conception in the FA is related to this light

emission. Let the **position of a firefly** in the space be a unique representation of a solution to a optimization problem, thus this position can be represented through a mathematical vector. Now, for reason of abstraction, let the **intensity of the light** emitted by the firefly be proportional to the function to be optimized and assume that the **attractiveness between the fireflies** is stronger when the light intensity is greater. Consequently, one can construct a model that takes these factor into account in order to **simulate the movements** of the several fireflies. As a result, due to the tendency to a convergence of them to a set of optimal positions (solutions), it is possible to use the mechanism for optimization.

For the abstraction of the natural system three principles are to be considered:

1. The fireflies are attracted to each other regardless of their sex;
2. The attractiveness is proportional to the brightness so that a less brighter firefly moves in the direction of the brighter one. In the same way, the brightness is relative to the observer, thus, the farther away two fireflies are from each other, the weaker the attractiveness between them is. If there is no firefly brighter than a particular one, it moves randomly;
3. The brightness is determined by the landscape of the objective function.

Hence, the firefly algorithm can be described as follows.

Algorithm 1 Firefly Algorithm

function FIREFLY ALGORITHM
  Define a objective function $f(x), \mathbf{x} = (x_1, ..., x_d)^T$
  Generate initial population of fireflies $x_i (i = 1, 2, ..., n)$
  Let the intensity I_i be proportional to $f(x_i)$
  while t < MaxGeneration **do**
    for $i = 1 : n$ **do**
      for $j = 1 : i$ **do**
        if $I_j \cdot Attractiveness > I_i$ **then**
          Move firefly i towards j
        end if
      end for
    end for
    if Firefly i did not move **then**
      Move i randomly
    end if
    Rank the fireflies to determine the current best
  end while
  return Best solution
end function

In the algorithm it is assumed that the brightness of a firefly in a position $\mathbf{x}$ is determined by the objective function applied to the vector $\mathbf{x}$ so that $I(\mathbf{x}) \propto f(\mathbf{x})$. However, the brightness perception depends on the distance r between the vectors. So, an attractiveness function $\beta(r)$ is to be defined. Given a medium absorption coefficient γ and the brightness β_0 in the source , this function varies according to the inverse square law and can be written as

$$\beta(r) = \beta_0 e^{-\gamma r^2},$$

which is, after a series expansion analysis, equivalent to

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2}.$$

Concerning the distance, it can be, as commonly, defined as the euclidean distance $r_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{id} - x_{jd})^2}$ although another distance definition may be used depending on the type of the problem (Yang 2012, p. 193). At last, the movement is calculated by a sum of vectors which characterize that a firefly will move a larger distance towards the other one, as the brightness is stronger and the current distance is shorter. Hence, one can describe it as

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \beta_0 e^{-\gamma r_{ij}^2} + \alpha_t \epsilon^{\mathbf{t}},$$

where the third term models a stochastic variation and t is the iteration number. In the formula α^t represents the radius in which the random variation is allowed and $\epsilon^{\mathbf{t}}$ is a vector of numbers sampled from a random distribution. This distribution can be uniform, Gaussian or also another one depending on the desired effect.

Yang (2009, p. 177) carries out a statistical comparison between the FA, PSO and GA with a set of functions. The firefly algorithm outperformed both in every presented case. The author states also that "FA is potentially more powerful in solving $\mathcal{NP}$-hard problems", which is the case of the DARP. Insights about parameter settings, complexity, recent applications and efficiency analysis can be found in Yang and He (2013).

4 ARGUMENTATION

The objective of this chapter is to present in details the design and implementation of the two approaches taken in order to solve the DARP. Firstly, the integer programming formulation is explained and discussed. Then, the model for the firefly metaheuristic is introduced showing the main procedures and decisions.

4.1 Integer Linear Programming Model

As the DARP is an optimization problem, one can define it as an mathematical integer program. On the one hand, This is a powerful way to solve this kind of problem, since it is relatively easy to formulate and quite flexible in adding new rules and conditions. On the other hand, the solving by a generic solver or by an exact algorithm is usually cost intensive and limited by the strictness of the linear program formulation.

4.1.1 Mathematical Formulation

In this work it is considered the **three-index formulation** proposed by Cordeau (2006, p. 574-575). For this purpose, consider n the number of requests of a problem instance. The DARP can then be defined through a directed graph $G(V, A)$, where V is the set of vertices representing stop locations. This set can be partitioned in three subsets, (1) the initial and final depot, that contains the vertices 0 and $2n + 1$; (2) the pick-up points comprehending the vertices $P = \{1, \ldots, n\}$; (3) the deliver points including the vertices $D = \{n + 1, \ldots, 2n\}$. This way, one define a request as a ordered pair $(i, n + i)$. Additionally, A is the set of edges that interconnect the locations to represent the travel time y_{ij} and the travel costs c_{ij}^k for a vehicle k. With that, G becomes a complete digraph, that is to say, there is exact two edges connecting every pair of distinct vertices, one edge in each direction. For reason of simplification, one can assume an undirected graph, where the costs of driving from a stop i to another j are the same as from j to i, for the same reason, one can set the costs $c_{ij}^k = t_{ij}$.

Moreover, every vertex has associated a load q_i that tells the quantity of passenger boarding or alighting in the point, these values are then positive or negative, respectively, thus, $q_i \geq 1, (\forall i \in P)$ and $q_i = -q_{i-n}, (\forall i \in D)$ and $q_i = 0, (\forall i \in \{0, 2n + 1\})$.

Likewise, the vertices indicate also a duration di which is a positive number that informs the time necessary to realize the service at that stop. Furthermore, a time window $[e_i, l_i]$ is associated to every vertex and represents, respectively, the earliest and the latest time at which the service may start at the location.

Regarding the vehicle attributes, let K be the quantity of buses at disposal, so, it is defined a vehicle capacity Q_k and a maximal route duration T_k, but, as we are considering the homogeneous version of the DARP, the indexation by $k \in K$ does not play an important role. At the last, the maximal route duration in respect of the passenger is given by the constant L.

Having considered all these parameters, it is possible to define the set of decision variables which identify a solution to the addressed problem. So, for each edge $(i, j) \in A$ and vehicle $k \in K$ there is a variable x_{ij}^k which is equal to *one* if the vehicle passes through this edge and equal to *zero* otherwise. Further, let u_i^k be the time at which the car k starts the boarding at the location i, w_i^k the load of k as leaving that point, and r_i^k the travel time of the users of the request identified by i. The linear program is then written as follows.

$$\textbf{minimize} \quad \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} c_{ij}^k x_{ij}^k \tag{1}$$

$$\textbf{subject to} \quad \sum_{k\in K}\sum_{j\in V} x_{ij}^k = 1 \qquad \forall i \in P \tag{2}$$

$$\sum_{i\in V} x_{0i}^k = \sum_{i\in V} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{3}$$

$$\sum_{j\in V} x_{ij}^k - \sum_{j\in V} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K \tag{4}$$

$$\sum_{j\in V} x_{ji}^k - \sum_{j\in V} x_{ij}^k = 0 \qquad \forall i \in P \cup D, k \in K \tag{5}$$

$$u_j^k \geq (u_i^k + d_i + t_{ij})x_{ij}^k \qquad \forall i, j \in V, k \in K \tag{6}$$

$$w_j^k \geq (w_i^k + q_j)x_{ij}^k \qquad \forall i, j \in V, k \in K \tag{7}$$

$$r_i^k \geq u_{n+i}^k - (u_i^k + d_i) \qquad \forall i \in P, k \in K \tag{8}$$

$$u_{2n+1}^k - u_0^k \leq T_k \qquad \forall k \in K \tag{9}$$

$$e_i \leq u_i^k \leq l_i \qquad \forall i \in V, k \in K \tag{10}$$

$$t_{i,n+i} \leq r_i^k \leq L \qquad \forall i \in P, k \in K \tag{11}$$

$$max\{0, q_i\} \leq w_i^k \leq min\{Q_k, Q_k + q_i\} \qquad \forall i \in V, k \in K \tag{12}$$

$$x_{ij}^k \in \mathbb{B} \qquad \forall i, j \in V, k \in K \tag{13}$$

In the program above, the objective function (1) minimizes the total operation costs which is calculated by the sum of the costs of every arc (i, j) where a vehicle travels through, since in this case on would have $x_{ij}^k = 1$. The equation (2) ensure that every request is taken by one and only one bus. Equation (3) model the single depot rule, which says that every vehicle starts and ends at the garage, identified by the vertices 0 and $2n + 1$. Equation (4) gives consistency to the assignments of vehicles to requests, since it constraints the solutions such that the bus k that picks up a request i is the same that delivers it to the final destination $n+i$. Analogously, (5) guarantee route consistency, that means that the bus that arrives at a determined site is the same to departure from this.

In the next constraints are defined the complementary decision variables. These are represented in the conditions (6), (7) and (8), which indirectly determine values to the time in which a vehicle k starts serving i (u_i^k), the load of k when leaving vertex i (w_i^k) and the ride time of the passengers served by i (r_i^k), respectively. Constraint (9) assures that a vehicle does not ride longer that it is allowed, the idea is that by subtracting the initial time of the route from the time when the vehicle terminates it, yields the the total ride time of a bus and then the condition can be easily expressed. Additionally, the equation (10) guarantees the restriction of time windows from the requests. In the same way, equation (11) guarantees that the ride time r_i^k of the passengers of a request i does not exceed the limit L.

Lastly, the consistency of the vehicle load is determined by the constraint (12). Note that, in the case which the vehicle picks passenger up, i.e. $q_i > 0$, this condition ensures a minimum load of q_i and a maximum of the load limit Q_k, the integrity of the rule is then fully covered by the equation (7). In the case which passengers drop off, i.e. $q_i < 0$, the constraint guarantees, besides a positive load, a superior value according to the load limit and the number of alighting passengers. Finally, the line (13) define the binary domain restriction over the variable x_{ij}^k.

4.1.2 Linearization

This proposed formulation is nevertheless not completely linear, since the the constraints (6) and (7) contain a multiplication of variables $u_i^k \cdot x_{ij}^k$ and $w_i^k \cdot x_{ij}^k$, respectively. Thus, they should pass through a linearization process which adds the constants M_{ij}^k and

W_{ij}^k. As a result, these two equations are replaced by the following ones.

$$u_j^k \geq u_i^k + d_i + t_{ij} - M_{ij}^k(1 - x_{ij}^k) \qquad \forall i, j \in V, k \in K \tag{14}$$

$$w_j^k \geq w_i^k + q_j - W_{ij}^k(1 - x_{ij}^k) \qquad \forall i, j \in V, k \in K \tag{15}$$

In order that these new constraints are valid, the constants M and W are subject to the next presented conditions.

$$M_{ij}^k \geq max\{0, l_i + d_i + t_{ij} - e_j\} \qquad \forall i, j \in V, k \in K \tag{16}$$

$$W_{ij}^k \geq min\{Q_k, Q_k + q_i\} \qquad \forall i, j \in V, k \in K \tag{17}$$

This procedure closely resembles *Miller-Tucker-Zemlin* subtour elimination constraints for the TSP (Cordeau 2006, p. 575). Yet the current formulation is not able to run in a generic linear programming solver. Then, in order to workaround the issue, it is enough to assign M_{ij}^k and W_{ij}^k a sufficient large positive number such that the condition (16) and (17) hold always true for the problem instances (Häll et al. 2009, p. 44).

4.1.3 Implementation

As part of the approach of this research, we seek exact solutions for a set of instances available in the literature. The first step in order to achieve that is transcribing the above detailed mathematical model into a programming language. The chosen language was AMPL which permits a quasi direct translation of the integer programming description to AMPL's notation (Fourer, Gay, and Kernighan 1990, p. 520). Additionally, in a lower level lies the GLPK (GNU Linear Programming Kit) solver whose role is to interpret the written program, admit the input, validate the linear model, execute an optimization algorithm and deliver an optimal solution by outputting the assignment of value to decision variables and calculating the objective function. Details of the experiments are showed in the corresponding chapter.

4.2 Firefly Metaheuristic Model

This section aims to explain the design of the Firefly Metaheuristic applied to the problem treated here. This approach consists of viewing the solutions for the DARP

as vectors in a multidimensional space. These solutions are represented by the fireflies, which can movement themselves in the search space by changing the components of the vector. There is a function which takes a vector as input and gives a real number that indicate how bright the firefly is, that is to say, how good the solution is.

The challenge is to fit the problem in this model, so that it can be computed in a effective and efficient way. This includes defining the vector of a solution, the brightness function, distance and movements in the space, the parameters, the randomness and the evolution of the optimization process.

4.2.1 Vector Representation of the Solution

In this model any solution can be represented through a natural number vector. This vector is an element of the search space. In order to understand its construction it is possible to split it into two parts. The first has always one component which describe the assignment of requests to buses. The second part has as many components as there are buses and each one represents a cycle in the requests graph, it means, the route that the bus executes in the solution. So a solution $S \in \mathbb{N} \times \underbrace{\mathbb{N}}_{k-times}$ for k buses.

4.2.1.1 Modeling the Assignment of Requests to Buses

The delegation of n requests to k buses can be represented by an n-tuple in which every element i represents the bus assigned to the i-th request, thus, it varies from 0 to $k-1$. Furthermore, with a simple combinatorial analysis one can show that the number of combinations of values that such tuple can get is k^n. One can also want to arrange all of these value combinations in the form of a tree so that, given a natural number between 0 and $k^n - 1$ uniquely identifying a tuple, a computational procedure can generate this one in a relatively simple fashion.

We then want to use this **unique enumeration** of the tuples as the first component of the above defined vectorial solution representation. For that, the arrangement of the n-tuples is structured in a **full tree** with a height of n where every non-leaf node represents a tuple component and has k children, since every component takes k possible values. Additionally, the leaf nodes are enumerated from 0 to k^n, therefore, representing the possible combinations. Thus, every walk on the tree, from the root node until a leaf, yields a possible request-bus assignment, in turn, every possibility can then be yielded by

a unique walk.

At last, the process of transforming the first component of the solution vector into a tuple which describes the assignment of the requests to the buses can be specified by a bijective function illustrated on the following algorithm.

Algorithm 2 Transformation Vector-Solution

function TRANSFORM COMPONENT INTO ASSIGNMENT REQUEST-BUS(x)
  $a \leftarrow x$
  for $i = 1 : n$ **do**
    $T_i \leftarrow \lfloor \frac{a}{k^{n-i}} \rfloor$
    $a \leftarrow a \bmod k^{n-i}$
  end for
  return T
end function

4.2.1.2 Modeling the Routes of the Buses

It was shown that the first component of the solution vector models the assignment of requests to buses. Likewise, this section deals with the k other components of that vector which concern the configuration of the respective vehicle routes. In order to have a complete representation, we show next how the order, in which a bus serves its requests, can be described.

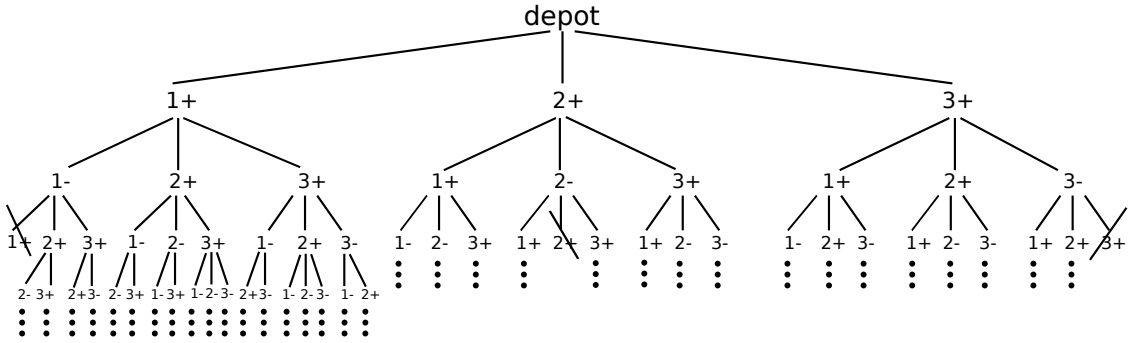
For this purpose, an analysis similar to the modeling of the previous section can be applied to the modeling of the routes of the buses. In this case, with the difference that the **permutation of the boarding and alighting locations** of the requests are mainly taken into consideration. Besides, other constraints are applied to the numerical representation of a route, namely that an alighting node of a given request cannot occur before the boarding of the same and that a vehicle is never allowed to carry more passengers than its capacity.

To exemplify, the Figure 4.1 illustrates an incomplete version of the route tree of a bus which three requests was assigned to. Each level i of the tree represents the possible locations where a vehicle can be at its i-th stop, so that every walk starting from the root, which is the start point, i.e. the depot, and ending at a leaf node characterizes a possible route. The nodes are identified by the number of the respective request followed by a positive or a negative sign, the former indicates a pick-up site, while the latter indicates the request deliver site.

However, unlike the codification of the request-bus assignment, in which every

node opens k new nodes in the lower level, there are more complex restrictions in the structure of the tree described here. These are shown by diagonal strokes in the Figure 4.1 which cut out cases of places repetition in the route, what would not make sense, since every location is accessed only once. In conclusion, **every walk from root to leaf generates a meaningful route**, that is to say, it contains no repetitions and the every pick-up occurs before the respective deliver, even though it may not be feasible concerning the problem constraints, because it yet does not consider vehicle load and time windows.

Figure 4.1: Route possibilities represented through a tree



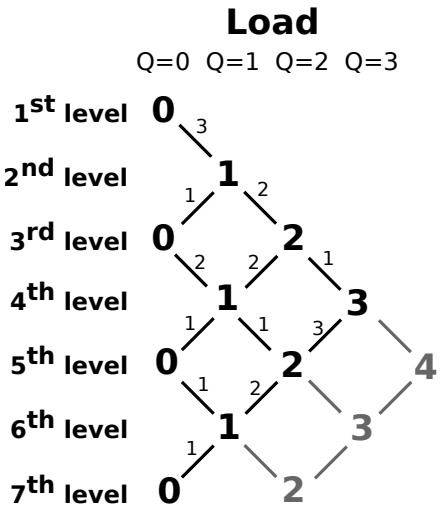
Source: Own authorship

Nevertheless, in order to build a function that allows us to have a mapping from each walk to a natural number, thus enumerating all the possible routes of a bus, is necessary to know, given a depth in the tree, how many leaves are under each vertex of this depth. Once this information is available, the transformation function can be computed in a quite efficient way.

Determining how the tree structure is at each level and altogether how many leaves there are depends exclusively on the load of the bus in a given vertex, it means, it depends on how many requests are being carried and how many there are still to pick up in a specific moment. The Figure 4.2 helps to explain this structure. In the illustration each node represents the load of the bus at the respective level, the further down it gets, the greater its respective depth becomes. The edges under a node tells how many children a vertex with the respective load yields. For instance, at the first level the vehicle is in the depot and therefore empty ($Q = 0$), at the second level, i.e. the first passenger pick-up, there are three possibilities that generate a bus load of 1. In the next steps, every one of the nodes can either pick up one new passenger, who has not been picked up yet, or drop one of q passengers, where q is the current vehicle load. Note that, in this example, for reason of simplification, it is assumed that there is *only one passenger per request*, an

expansion to *many passengers per request* can be similarly done.

Figure 4.2: Graph describing the structure of the route tree



Source: Own authorship

From the graph illustrated in the Figure 4.2 it is possible to extract two matrices in order to represent the structure, one for the pick-up cases (*edges leaving nodes to the right*) and one for the drop-off cases (*edges leaving nodes to the left*). Respectively, the number on these edges are denoted in the following matrices Q_{in} and Q_{out}.

$$Q_{in} = \begin{bmatrix} 3 & 0 & 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, Q_{out} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \end{bmatrix}$$

Each row of the matrix maps a car load, starting from the first row with $Q = 0$. Additionally, each column maps a level, starting from the one most at left. So, the element a_{ij}^{in} of the matrix Q_{in} equals the number of children created by a node equivalent to a load $q = i$ with level j, that lead to an increase of the load (*pick-up*). Analogously, the element a_{ij}^{out} of the matrix Q_{out} is equals to the number of children, that have a lesser load, yielded by every node with $q = i$ in the j-th level (*deliver*). Once again, for the simplification of the explanation, the capacity Q_k of the vehicle is not being considered, but it could be done by simply setting every element in a row $i > Q_k$ to zero.

As already mentioned before, in order to design an algorithm similar to the one in the Section 4.2.1.1, one would need to know at every node of the tree, i.e. every stop of

the route, how many leaves there are under this node, i.e. how many combination still can be done with the rest of the route. For that, a third matrix Q is generated by combining Q_{in} and Q_{out} inside the following algorithm. This new matrix then maps in each element a_{ij} the amount of leaf nodes under a vertex with $q = i$ in the j-th level.

Algorithm 3 Matrix Generation

function GENERATE MATRIX OF NUMBER OF LEAF NODES(Q^{in}, Q^{out})
Let n be the the number of requests and $\odot$ the element-wise multiplication
$Q_{2n+1} \leftarrow \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$
for $i = 2n : 1$ **do**
$Q_i \leftarrow Q_i^{in} \odot \begin{bmatrix} Q_{i+1,1..n-1} \\ 0 \end{bmatrix} + Q_i^{out} \odot \begin{bmatrix} 0 \\ Q_{i+1,2..n} \end{bmatrix}$
end for
return Q
end function

The result of applying the function for the matrices Q_{in} and Q_{out} is shown below. This provides additionally a new information about the feasible solutions interval of the correspondent component in the solution vector, namely the entry $Q_{1,1}$, which tells the total number of leaves in the tree, it means, the number of possible routes. Ultimately, it is useful for constraint checking and for an easy creation of the initial generation of fireflies.

$$Q = \begin{bmatrix} 90 & 0 & 6 & 0 & 1 & 0 & 1 \\ 0 & 30 & 0 & 3 & 0 & 1 & 0 \\ 0 & 0 & 12 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 \end{bmatrix}$$

With help of these three matrices and given a correspondent vector component value, the path of a bus can be computed by a transformation function. Below is shown a function that takes these arguments and delivers a path performed by a bus, where its elements do not represent the requests themselves, but the vertices of the walk in the tree, which is relative to each bus. A mapping to the absolute requests can be trivially done, once the requests assigned to each bus are known.

Algorithm 4 Transformation Vector-Solution

function TRANSFORM COMPONENT TO A WALK IN THE TREE(Q^{in}, Q^{out}, Q, x)
Let n be the the number of requests of the bus
$Path \leftarrow []$
$row \leftarrow 0$
$pointer \leftarrow 0$
for $col = 1 : 2n$ **do**

$$ChildrenSizes = \left[\begin{array}{c} \underbrace{Q_{row-1,col+1}}_{Q^{out}_{row,col} times} \\ \underbrace{Q_{row+1,col+1}}_{Q^{in}_{row,col} times} \end{array}\right]$$

for $child = 1 : Q^{in}_{row,col} + Q^{out}_{row,col}$ **do**
if $x - pointer < ChildrenSizes_{child}$ **then**
if $child < Q^{out}_{row,col}$ **then**
$row \leftarrow row - 1$
else
$row \leftarrow row + 1$
end if
else
$pointer \leftarrow pointer + ChildrenSizes_{child}$
end if
end for
$Path_{col} \leftarrow child$
end for
return $Path$
end function

At last, after transforming a vectorial representation of a solution of the DARP, which is inside its valid value interval, into the list of vehicle routes, the only infeasibility that there may be is the one concerning time windows and ride time constraints. This will be discussed in subsequent sections.

4.2.2 Distance

For reasons of efficiency of the implementation and numerical error the **Manhattan distance** can be used to approximate the distance between two vectors in the search space. Note that the *Manhattan distance* is very convenient for combinatorial applications because if the vectors are composed only by integer numbers, then its result will be

as well expressed in integer numbers. The distance is calculated as follows.

$$d(\mathbf{p}, \mathbf{q}) = \| \mathbf{p} - \mathbf{q} \| = \sum_{i=1}^{n} | \mathbf{p}_i - \mathbf{q}_i |$$

4.2.3 Attractiveness

As proposed by Yang (2009, p. 173) the attractiveness is calculated with the following quotient, which varies with the squared distance between two vectors.

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2}$$

However, without loss of quality in the method, the attractiveness can vary directly with the distance, as the search space is too large and concerns with numerical errors play an important role. Moreover, due to these concerns, we assume that $\beta_0 = 1$ and define the inverse function β^{-1} which will be useful for calculating the dislocation of a firefly with a minor error.

$$\beta^{-1}(r) = 1 + \gamma r$$

4.2.4 Randomization Term

The random term of the metaheuristic movement equation is by default $\alpha \cdot \epsilon$, where $\epsilon \sim N(0, 1)$. Though, as stated by Yang (2010, p. 80), "it is a good idea to replace α by αS_k where the scaling parameters $S_k (k = 1, ..., d)$ in the d dimensions should be determined by the actual scales of the problem of interest". As the size of each dimension are easily obtainable, they are used in this model. Moreover, Yang and He (2013, p. 37-38) presents an iterative change of the *alpha* parameter, by varying it according to the optimization evolution t. Thus, he introduces a new variable *delta*, so *alpha* is updated by the equation

$$\alpha_t = \alpha_{t-1} \cdot \delta, (0 < \delta < 1),$$

where α_0 is the initial scaling factor. Additionally, the author gives also an advice about setting *delta*: "δ is essentially a cooling factor. For most applications one can use $\delta = 0.95$ to 0.97".

At last, it is wanted an integer number for the stochastic term, as the search space is

discrete. In order to achieve that, rounding is performed, proper concerns about numerical errors should be taken in the implementation, so that they are reduced at most.

4.2.5 Movement in the Discrete Space

The movement of a firefly i towards j is determined by the following equation.

$$\mathbf{x_i^{t+1}} = \mathbf{x_i^t} + \beta(d(\mathbf{x_i^t}, \mathbf{x_j^t})) \cdot (\mathbf{x_j^t} - \mathbf{x_i^t}) + RandomTerm(t)$$

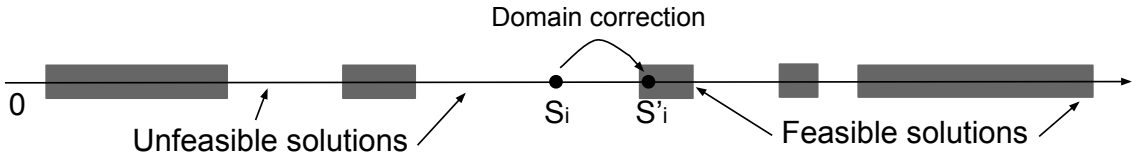
As the fireflies move in a discrete vector space, the terms of the sum above should also be integer numbers. The first and the last one are intrinsically in the set of the integers. Despite of that, the second term muss be rewritten, so that it delivers an integer number with the lowest possible error. Thus, we use the β^{-1} function, defined in the Section 4.2.3, and turn the multiplication into a division. Lastly, a rounding should be performed in the quotient, this is done by applying the floor function. As a result, we obtain the function below.

$$\mathbf{x_i^{t+1}} = \mathbf{x_i^t} + \lfloor \frac{(\mathbf{x_j^t} - \mathbf{x_i^t})}{\beta^{-1}(d(\mathbf{x_i^t}, \mathbf{x_j^t}))} \rfloor + RandomTerm(t)$$

It is important however to notice that the movement formula may position a firefly out of the interval where one can find solutions, since every dimension of the search space has a range in which meaningful routes can be interpreted from the firefly vector. On this issue the so-called **saturation arithmetic** is applied by *clipping* the result, in other words, if the numerical values resulted from the calculation are outside the domain, i.e. greater than its upper limit or less than zero, then the limit values are taken instead.

Moreover, even if the vector lies inside the mentioned domain, it can still deliver unfeasible solutions due to the time windows and passenger ride time constraints. Likewise, a domain adjustment procedure alters the vector in such a way that it then represents a similar but feasible solution. The operation is abstractly exemplified in the Figure 4.3. The graph illustrates one of the domains of the search space, the gray regions characterize range of values which hold valid solutions, the point S_i is a component of a vector and S_i' is the new component value after the application of the domain correction procedure.

Figure 4.3: Illustration of the domain correction procedure



Source: Own authorship

The **domain correction procedure** consists in rearranging the routes of every bus, such that the distance of between the new vector and the old one is as low as possible. The Figure 4.1 helps to clarify the situation, it shows how the routes can be represented by a tree, note also that a branch from the root to a leaf describes a possible vehicle route. The objective of the correction function is finding a new branch in the tree, that lies near the unfeasible one. To solve the problem, one can see it as a **constraint satisfaction problem**, where the set of variables are the request locations, their domains are the possible positions in the route order that they can be placed in, at last, the set of constraints are generated by the time window limitations which tell the if a locations muss be accessed before another one. In this work the renowned **Arc Consistency Algorithm #3** by Mackworth (1977) is implemented.

To add a final observation, it is to note that, despite of all the effort to correct the firefly vector, the adjustment function may not find a valid route, if for instance the assignment of requests to buses does not make it possible. In this case, the firefly lies out of domain and should have a null intensity.

4.2.6 Intensity Function

The intensity function models the brightness of a firefly and should be directly proportional to the *utility function* of the problem to be maximized. Since the problem's goal is to minimize the operation costs (or the operation time), one could then think of using the *negative cost function* as utility function, what is commonly done when treating optimization problems. However, the Firefly Algorithm requires a positive function instead, since the intensity is by definition non-negative.

Nevertheless, the cost function has a greatest possible value, that can be estimated by calculating the worst set of vehicle routes. Here it is proposed to sum the v most distance vertices in the requests graph, so that v is twice the number of requests since

every request has two correspondent vertices, one for boarding and one for alighting. Once this value is calculated, the costs of a given solution can be subtracted from it, in order to obtain an utility function. This process is equivalent to translating the negative cost function, i.e. shifting it upwards in the *Cartesian plane*.

So, to start with, assume the the route costs are equal to the route times, then, let m be the mentioned superior cost limit, n the number of requests and C the adjacency matrix of the requests graph with the costs between any stop location, note that C is a square matrix with $2n + 1$ rows which include the pick-up and delivery locations plus the depot. Finally, let f be a function that, given the solution vector $\mathbf{x}$, returns a binary matrix with the same shape than C and whose elements have the value 1 if a vehicle travels through the respective edge, and 0 otherwise. Hence, the intensity function I can be defined in the following formula.

$$I(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \text{ does not meet the constraints} \\ m - \sum_{i=1}^{2n+1} \sum_{j=1}^{2n+1} f(\mathbf{x})_{i,j} \cdot C_{i,j}, & \text{else} \end{cases}$$

4.2.7 Initial Solution

A initial set of initial solutions is calculated by simply generating for each component of the vector a random natural number in a valid interval. Firstly, the first component is randomized, its range is $[0, k^n)$, where n is the number of requests and k the number of buses. Secondly, the other components are randomized, their domain intervals are determined based on the assignment of requests to buses resulted from the first component randomization. This way, the first generation of fireflies can be efficiently created. Note that, there is no guarantee that these are feasible solutions, yet it does not pose an issue, since these vectors will be corrected throughout the iterations.

4.2.8 Parameters

The choice of the parameters is basically based on the work of Yang and He (2013, p. 37-38). For that, the scale L of the problem is taken into consideration, it represents namely the size of the search space and is calculated by multiplying the sizes of the

intervals of each dimension, in which the intensity function is defined. The parameter *gamma* is then set $\gamma = 1/\sqrt{L}$. In order to have a broad exploration of the space it is set $\alpha_0 = 0.1$, the parameter is then reduced along the optimization process. Lastly, it is set $\beta_0 = 1$, $\delta = 0.95$ and the number of fireflies to 40.

Regarding the stop criteria, the algorithm stops if there no improvement in the best solution after 200 iterations or the algorithm has executed 1000 iterations, whatever comes first.

4.2.9 Two-Phase Optimization

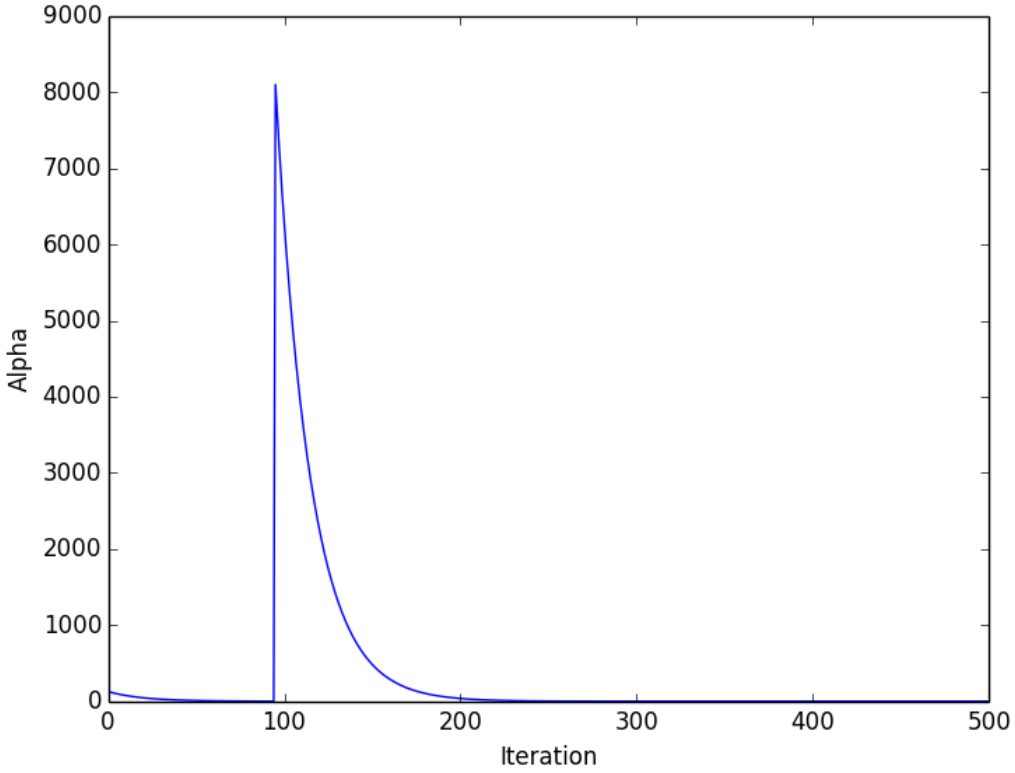
It is to note in a regular problem instance that the assignment of requests to bus has a greater weight it the cost function. Empirical observations suggest that this assignment is roughly the most determinant factor in the calculation of the costs. For instance, the grouping requests that are geographically near tends to show better results regardless of the route. Yet the route of each bus plays an important role in finding an optimal result.

Moreover, as the contribution to the cost function by the vector components that describe each bus' route is strongly dependent on which requests each bus is allocated to, it makes no sense trying to optimize vehicle routes in a stage where the assignment request-bus shows a large deviation from iteration to iteration.

On the basis of these observations, an optimization process with two phases is applied. In the model presented here, there is a first stage, where only the first vector component is optimized by moving the fireflies only in this dimension of the search space. In a second stage, the other components are optimized by anchoring the first component and allowing the movement of the fireflies in the other dimension.

The Figure 4.4 displays the evolution of the alpha parameter as described in the Section 4.2.4 in the two-phase optimization process. The X-axis shows the iteration number, while the Y-axis shows in the respective iteration the value of alpha scaled according to (i) the first dimension's range in the first phase, and (ii) the range of the other dimensions in the second phase.

Figure 4.4: Evolution of the alpha parameter



Source: Own authorship

4.2.10 Implementation

One of the main difficulties in implementing the algorithm is the magnitude that the numbers might assume. To workaround the problem the programming language *Python* was chosen, due to its native implementation of integers with unlimited size. Besides, the proposed model has a considerable quantity of matrix operations, therefore, the libraries *NumPy* and *Scipy* were used for that purpose. For drawing the graphs it was adopted the library *Matplotlib*, which has an easy-to-use interface for the programmer.

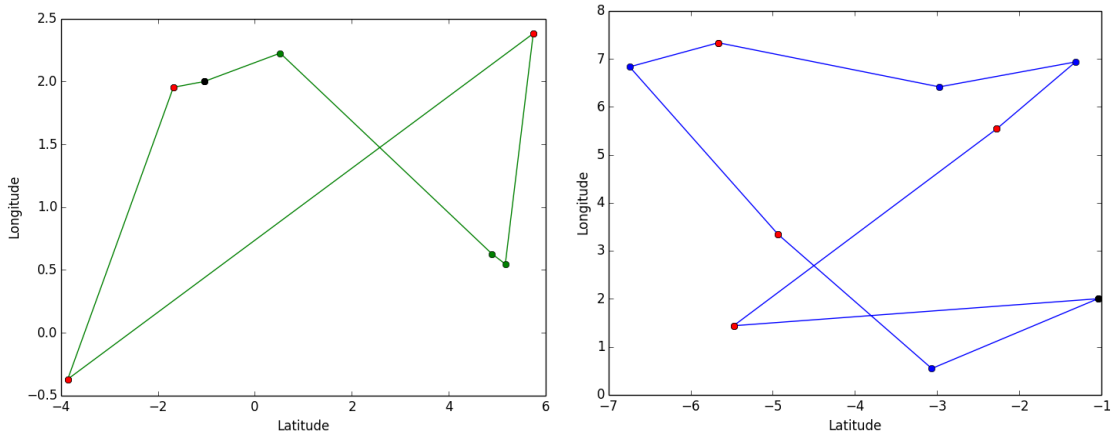
A special concern with the numerical operations and the data types had to be taken to ensure no integer overflows and a controlled numerical error at the lowest level.

To obtain a better performance and to have the most recent features available, the implementation was developed and executed on the newest possible tool versions. Namely, the versions 3.4 of *Python*, 1.10 of *NumPy*, 0.16 of *SciPy* and 1.3 of *Matplotlib*.

The Figure 4.5 shows an example of one of the output given by the implemented program, it displays the routes of two vehicles in a xy-plane. In each graph, the black dot

represents the depot and the red dots represent the drop-off stops, while the blue and the green dots are pick-up stops.

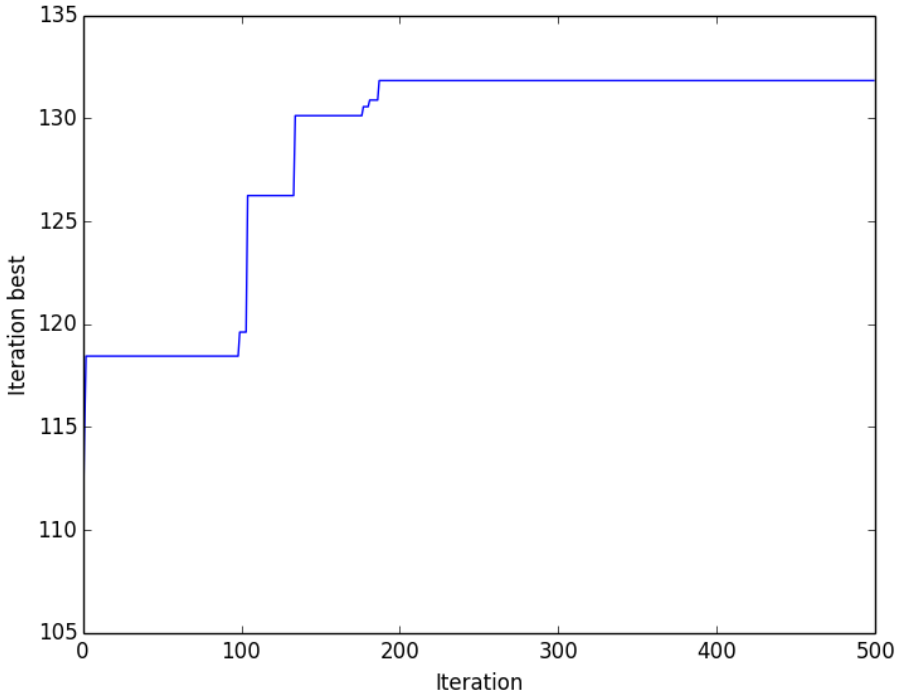
Figure 4.5: Example of output by the program



Source: Own authorship

The Figure 4.6 illustrates how the current best solution evolves along the algorithm execution with a test input. Observe that the Y-axis represents the utility function instead of the costs.

Figure 4.6: Evolution of the current best solution along the iterations



Source: Own authorship

5 EVALUATION

The objective of this chapter is evaluate the proposed implementation of the firefly metaheuristic by running it with several problem instances as input. The characteristics that are taken into consideration are the **result quality**, that is to say, how far the calculated cost for the best solution found is from the instance's optimal solution, the **algorithm progress**, which tells how better the best solution found is in comparison to the initial solution, and, at last, the **running time**.

Two distinct types of experiments are carried out. In the first place, the firefly approach is compared against the generic solver one by executing both programs with the same input. With it, it is expected to asses to what extent the metaheuristic implementation outperforms the generic solver. In the second place, the near-optimal approach is tested with instances that are recurrently used in the literature. Similarly, the deviation of the solution to the optimal and the program processing time are evaluated and compared to state-of-the-art methods.

In the total, the experiments include 24 instances which are solved in a computational environment with a 64bits *Intel Xeon* CPU at 2.3 GHz, 15 GB of main memory and a *Python* interpreter version 3.4 on a *Linux* operating system.

5.1 Results and Discussion

First of all, we wanted to evaluate the exact approach through the integer programming solver GLPK and how it performs in comparison to the near-optimal implementation. The instances were then executed with a memory limit of 7 GB. But, as one could have expected, the generic solver was not able to solve even the smallest instance *a2-16*, neither a feasible non-optimal solution could be found. Though, in order to present results of comparison, the instance *a2-16* was reduced in several smaller ones, between 3 and 8 requests.

The results are shown in the Figure 5.1, which summarizes the values that we are interested in. In the columns *Req*, *Bus* and *Cap* are listed the features *number of requests*, *number of buses* and *vehicle capacity*, respectively, for each instance. The columns *Opt.* and *CPU-GLPK* display the cost of the optimal solution found by the generic solver and the CPU time in minutes, respectively. The columns *Firefly Sol.* and *% Opt.* show the solution found by the metaheuristic approach and its deviation to the optimum, this is

calculated by simply dividing the value in *Opt.* by the one *Firefly Sol.*. Finally, the time in minutes of the solution through the Firefly algorithm can be seen in the last column.

Table 5.1: Results of the first evaluation

Inst.	Req	Bus	Cap	Opt.	CPU-GLPK	Firefly Sol.	% Opt.	CPU-Firefly
Test-1	3	2	3	58.05	0.1	69.75	83.2%	7.4
Test-2	4	2	3	68.10	0.9	98.51	69.1%	11.7
Test-3	5	2	3	76.27	2.0	134.29	56.8%	2.5
Test-4	6	2	3	96.54	45.0	149.32	64.6%	3.3
Test-5	8	2	3	-	-	158.67	-	5.7

Source: Own authorship

Note that, because it had reached the execution limit, the exact approach could not solve the instance *Test-5* of the experiment. This fact together with the growth of CPU time required for the GLPK to solve instances as they become bigger suggests an exponential increase in use of resources (e.g. processing time and memory) that the exact solver consumes. All in all, it confirms what one can generally expect from the generic linear programming solver when the growth of the input increases the dimension of the solution space, this is is the so-called *Bellman's curse of dimensionality*[1]

In light of the results, one can conclude that, on the one hand, GLPK is very effective for small instances, but on the other hand, due to its high complexity of both time and space, it is not capable of being applied for larger inputs, as in real applications. As opposed to that, the solution with the metaheuristic may be not so effective in delivering the optimal result. However, on the basis of its robustness, its efficiency when solving larger instances lies way above the exact approach. This can be clearly seen in the column *CPU-Firefly*, whereby the algorithm's computational time presents a relatively small variation and appears to have a steady growth as the inputs become larger. This observation is supported by the subsequent experiments.

The second conducted evaluation analyzed the following three aspects of the problem solver, which implemented the firefly algorithm: (i) how close to the optimum the metaheuristic approach can get, i.e. the solution quality; (ii) how the solutions are improved from the initial feasible solution until the final one, this indicates in what way the optimization method in fact helps in order to minimize the cost function; (iii) how, in a temporal perspective, the application behaves regarding the increase in the input size.

From the total of 24 instances 15 could be solved, the results are presented in the

1. See Bellman, Richard E. 2003. Dynamic Programming. Courier Corporation.

Table 5.2. The two first columns describe the characteristics of the data inputs, respectively name and optimal solution. The columns *Final Sol.* and *Optimality* show the best solution found and its proportion with respect to the instance optimum. The two last columns refer to the progress of the solutions, in *Initial Sol.* is the first yielded feasible solution and in *Dev. to Initial Sol.* is the deviation of the final result to the initial one, it is calculated by dividing the difference between these two values by the initial solution, in formula: $\textit{Dev. to Initial Sol.} = (\textit{Initial Sol.} - \textit{Final Sol.})/\textit{Initial Sol.}$.

Table 5.2: Results of the second evaluation − Quality and progress

Instance	Optimum	Final Sol.	Optimality	Initial Sol.	Dev. to Initial Sol.
a2-16	294.25	312.96	94.02%	335.85	6.81%
a2-20	344.83	373.89	92.23%	427.48	12.54%
a2-24	431.12	442.85	97.35%	496.92	10.88%
a3-18	300.48	347.10	86.57%	374.20	7.24%
a3-24	344.83	409.73	84.16%	456.86	10.32%
a3-30	494.85	614.13	80.58%	615.99	0.30%
a3-36	583.19	-	-	-	-
a4-16	282.68	310.84	90.94%	354.54	12.33%
a4-24	375.02	481.16	77.94%	567.63	15.23%
a4-32	485.50	639.73	75.89%	665.27	3.84%
a4-40	557.69	780.13	71.49%	841.92	7.34%
a4-48	668.82	956.90	69.89%	1032.38	7.31%
a5-40	498.41	779.32	63.95%	818.14	4.74%
a5-50	686.62	926.23	74.13%	1018.84	9.09%
a5-60	808.42	1195.94	67.60%	1257.38	4.89%
a6-48	604.12	993.08	60.83%	1025.55	3.17%

Source: Own authorship

Note that it was not possible to find a solution for the instance *a3-36*. That is mainly due to the fact that it has a high rate request per bus, as one can see in the Table 2.1. it implies that with a higher number of requests in a bus route it becomes very often impossible arranging the locations in an order that fulfills the problems constraints. On this issue, one should take into consideration, that there are two distinct tasks that are affected by the density of request per bus, firstly, the assignment of requests to buses, secondly, the arrangement of each bus' route. Then, as the former is performed merely randomly, the algorithm of domain correction, which is applied solely to the routes, can be quite ineffective, since a desired effect depends broadly on the assignment of locations to the specific route.

When considering the optimality of the results, one can ascertain that the imple-

mentation delivers very good solutions to the smaller instances, for instance, from the *a2-16* till the *a4-16* the mean is at approximately 90%. The other outcomes, which lie below the percentage of 80%, have a performance that should be already expected. At that, the search space becomes very large due to its high dimensionality, besides, the sparseness of the regions of feasible solutions increase, so that interaction between the fireflies can be less and less frequent, thus, almost annulling the optimization process.

Moreover, the balance between *exploration* and *exploitation* plays an important role. A policy of exploration tends to lead the optimization process away from local optima, avoiding low quality solutions, at the expense of increasing the processing time. By contrast, a policy of exploitation tends to a local search, delivering impaired solution, but in a short period of time. To sum up, whether the solution of an instances has a better or worse quality depends also on this balance, it can be eventually improved, though the price that one have to pay might be unwanted.

At last, the progress from a feasible initial solution to the best one has presented a variation at the experiments, namely from 0.3% to over 15%. That is mostly because of the domain correction procedure performed with the initial randomly distributed solution vectors. Since the set of initial solution happens to be highly probable to be invalid, they pass through the domain adjustment algorithm, whose heuristic for forming the feasible route may eventually be good enough to place the vector very close to the optimum.

[DESCRIBE THE TABLE]

Table 5.3: Results of the second evaluation − Running time

Instance	CPU[1]	CPU[2]	CPU-Firefly
a2-16	68.2	0.12	**63.3**
a2-20	133.8	0.28	160.5
a2-24	187.8	0.35	419.3
a3-18	45.4	-	**29.5**
a3-24	86.8	0.29	**77.4**
a3-30	105.6	0.50	151.2
a3-36	162.6	0.83	370.6
a4-16	26.0	-	**14.5**
a4-24	50.8	-	**37.0**
a4-32	86.0	0.55	129.4
a4-40	130.6	0.78	233.5
a4-48	253.8	1.62	**222.3**
a5-40	-	0.85	**156.8**
a5-50	-	1.60	**366.1**
a5-60	-	2.51	**323.1**
a6-48	-	1.14	**146.7**
a6-60	-	2.29	279.3
a6-72	-	4.43	648.7
a7-56	-	1.67	186.2
a7-70	-	2.88	348.5
a7-84	-	7.04	553.5
a8-64	-	2.14	101.8
a8-80	-	5.73	203.7
a8-96	-	9.92	477.4

Source: Own authorship

[DISCUSSION]

6 CONCLUSION

- Further Research - Possible improvements

BIBLIOGRAPHY

Apostolopoulos, Theofanis, and Aristidis Vlachos. 2010. "Application of the Firefly Algorithm for Solving the Economic Emissions Load Dispatch Problem". *International Journal of Combinatorics* 2011.

Berbeglia, Gerardo, Jean-François Cordeau, and Gilbert Laporte. 2010. "Dynamic pickup and delivery problems". *European Journal of Operational Research* 202 (1): 8–15.

Braekers, Kris, An Caris, and Gerrit K. Janssens. 2014. "Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots". *Transportation Research Part B: Methodological* 67:166–186.

Carvalho, Marco Aurelio de, Semyon D. Savransky, and Tz-Chin Wei. 2004. *121 Heuristics for Solving Problems.* Lulu.com.

Cordeau, Jean-François. 2006. "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem". *Operations Research* 54 (3): 573–586.

Cordeau, Jean-François, and Gilbert Laporte. 2007. "The dial-a-ride problem: models and algorithms". *Annals of Operations Research* 153 (1): 29–46.

Fourer, Robert, David M. Gay, and Brian W. Kernighan. 1990. "A Modeling Language for Mathematical Programming". *Management Science* 36 (5): 519–554.

———. 2003. *AMPL: A Modeling Language for Mathematical Programming.* 2nd. Pacific Grove, CA: Cengage Learning.

Gribkovskaia, Irina, Øyvind Halskau sr., Gilbert Laporte, and Martin Vlček. 2007. "General solutions to the single vehicle routing problem with pickups and deliveries". *European Journal of Operational Research* 180 (2): 568–584.

Häll, Carl H., Henrik Andersson, Jan T. Lundgren, and Peter Värbrand. 2009. "The Integrated Dial-a-Ride Problem". *Public Transport* 1 (1): 39–54.

Jati, Gilang Kusuma, and Suyanto. 2011. "Evolutionary Discrete Firefly Algorithm for Travelling Salesman Problem". In *Adaptive and Intelligent Systems,* 393–403. Lecture Notes in Computer Science 6943. Springer Berlin Heidelberg.

Jorgensen, Rene Munk, Jesper Larsen, and Kristin Berg Bergvinsdottir. 2007. “Solving the Dial-a-Ride problem using genetic algorithms”. *Journal of the Operational Research Society* 58 (10).

Kirchler, Dominik, and Roberto Wolfler Calvo. 2013. “A Granular Tabu Search algorithm for the Dial-a-Ride Problem”. *Transportation Research Part B: Methodological* 56:120–135.

Mackworth, Alan K. 1977. “Consistency in network of relations”. *Artificial Intelligence* 8 (1): 99–118.

Nanry, William P, and J Wesley Barnes. 2000. “Solving the pickup and delivery problem with time windows using reactive tabu search”. *Transportation Research Part B: Methodological* 34 (2): 107–121.

Oki, Eiji. 2012. *Linear Programming and Algorithms for Communication Networks: A Practical Guide to Network Design, Control, and Management.* CRC Press.

Osman, Ibrahim H., and James P. Kelly. 2012. *Meta-Heuristics: Theory and Applications.* Springer Science & Business Media.

Parragh, Sophie N. 2011. “Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem”. *Transportation Research Part C: Emerging Technologies* 19 (5): 912–930.

Parragh, Sophie N., Jean-François Cordeau, Karl F. Doerner, and Richard F. Hartl. 2012. “Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints”. *OR Spectrum* 34 (3): 593–633.

Parragh, Sophie N., and Verena Schmid. 2013. “Hybrid column generation and large neighborhood search for the dial-a-ride problem”. *Computers & Operations Research* 40 (1): 490–497.

Ropke, Stefan, Jean-François Cordeau, and Gilbert Laporte. 2007. “Models and branch-and-cut algorithms for pickup and delivery problems with time windows”. *Networks* 49 (4): 258–272.

Sayadi, Mohammad Kazem, Ashkan Hafezalkotob, and Seyed Gholamreza Jalali Naini. 2013. “Firefly-inspired algorithm for discrete optimization problems: An application to manufacturing cell formation”. *Journal of Manufacturing Systems* 32 (1): 78–84.

Sayadi, Mohammad Kazem, Reza Ramezanian, and Nader Ghaffari-Nasab. 2010. “A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems”. *International Journal of Industrial Engineering Computations* 1 (1): 1–10.

Schrijver, Alexander. 1998. *Theory of Linear and Integer Programming.* John Wiley & Sons.

Shenoy, G. V. 2007. *Linear Programming: Methods and Applications.* New Age International.

Urra, Enrique, Claudio Cubillos, and Daniel Cabrera-Paniagua. 2015. “A Hyperheuristic for the Dial-a-Ride Problem with Time Windows”. *Mathematical Problems in Engineering* 2015.

Wassan, Niaz A., and Gábor Nagy. 2014. “Vehicle routing problem with deliveries and pickups: Modelling issues and meta-heuristics solution approaches”. *International Journal of Transportation* 2 (1): 95–110.

Wassan, Niaz A., A. Hameed Wassan, and Gábor Nagy. 2008. “A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries”. *Journal of Combinatorial Optimization* 15 (4): 368–386.

Wolsey, Laurence A., and George L. Nemhauser. 2014. *Integer and Combinatorial Optimization.* John Wiley & Sons.

Yang, Xin-She. 2009. “Firefly Algorithms for Multimodal Optimization”. In *Stochastic Algorithms: Foundations and Applications,* 169–178. Lecture Notes in Computer Science 5792. Springer Berlin Heidelberg.

———. 2010. “Firefly algorithm, stochastic test functions and design optimisation”. *International Journal of Bio-Inspired Computation* 2 (2): 78–84.

———. 2012. “Efficiency Analysis of Swarm Intelligence and Randomization Techniques”. *Journal of Computational and Theoretical Nanoscience* 9 (2): 189–198.

Yang, Xin-She, and Xingshi He. 2013. “Firefly Algorithm: Recent Advances and Applications”. *International Journal of Swarm Intelligence* 1 (1): 36–50.

Zidi, Issam, Khaled Mesghouni, Kamel Zidi, and Khaled Ghedira. 2012. “A multi-objective simulated annealing for the multi-criteria dial a ride problem”. *Engineering Applications of Artificial Intelligence* 25 (6): 1121–1131.