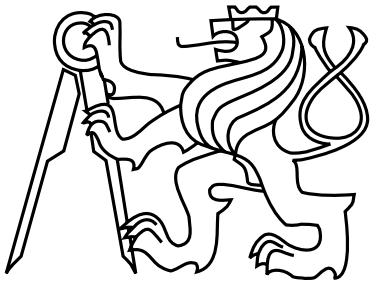
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor's Thesis

Single-Vehicle DARP Optimization for Ridesharing Using Operational Research Methods

Pavel Martinec

Supervisor: Ing. David Fiedler

Study Programme: Open Informatics

Field of Study: Artificial Intelligence and Computer Science

May, 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Martinec Pavel** Personal ID number: **483734**

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

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Single-Vehicle DARP Optimization for Ridesharing Using Operational Research Methods

Bachelor’s thesis title in Czech:

Optimalizace single-vehicle DARP pro sdílení jízd pomocí technik operační analýzy

Guidelines:

1. Analyze the operational research technics for dial-a-ride problem (DARP) and vehicle routing problem with time windows.
2. Based on your analysis, choose an exact method or technique that promises to solve the single-vehicle DARP (SVDARP) efficiently.
3. Implement the proposed method in the DARP benchmark framework.
4. Compare your solution to a baseline backtracking solution.
5. Measure the efficiency for a ridesharing algorithm that uses SVDARP as a subproblem: the vehicle-group assignment method.

Bibliography / sources:

[1] S. C. Ho, W. Y. Szeto, Y.-H. Kuo, J. M. Y. Leung, M. Petering, and T. W. H. Tou, ‘A survey of dial-a-ride problems: Literature review and recent developments’, Transportation Research Part B: Methodological, vol. 111, pp. 395–421, May 2018, doi: 10.1016/j.trb.2018.02.001.
[2] J.-F. Cordeau and G. Laporte, ‘The dial-a-ride problem: models and algorithms’, Ann Oper Res, vol. 153, no. 1, pp. 29–46, Sep. 2007, doi: 10.1007/s10479-007-0170-8.
[3] J.-F. Cordeau, ‘A Branch-and-Cut Algorithm for the Dial-a-Ride Problem’, Operations Research, vol. 54, no. 3, pp. 573–586, Jun. 2006, doi: 10.1287/opre.1060.0283.
[4] S. Ropke, J.-F. Cordeau, and G. Laporte, ‘Models and branch-and-cut algorithms for pickup and delivery problems with time windows’, Networks, vol. 49, no. 4, pp. 258–272, 2007, doi: 10.1002/net.20177.

Name and workplace of bachelor’s thesis supervisor:

Ing. David Fiedler, Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **06.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. David Fiedler Supervisor’s signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

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

. _____ _____

Date of assignment receipt | Student's signature

Aknowledgements

I would like to thank my supervisor Ing. David Fiedler, for his help, valuable advice, and comments throughout my bachelor's thesis. I would also like to thank my family for their support, especially my girlfriend, who has always been my support during writing.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, May 2021 .....

Abstract

This bachelor thesis deals with the acceleration of single-vehicle dial-a-ride problem (SVDARP) method using operational research techniques. Our motivation was to speed up SVDARP algorithm in cases, where SVDARP occurs as a subproblem of more complex algorithms, in order to speed up these algorithms. A key aspect also was to focus on instances with large number of requests, where the SVDARP algorithm using backtracking runs slow. The aim was to analyze current research in the area of the dial-a-ride problem (DARP), and based on this analysis propose an exact method adapted to solve SVDARP. We selected a three-index formulation of the DARP, which we modified for the SVDARP problem and then solved using branch-and-cut method. Subsequently, we implemented the formulation with all the improvements (arc elimination, time-window tightening, own cutting planes in branch-and-cut algorithm). The result of our work is the implemented SVDARP algorithm and its comparison with the original method using backtracking. Our approach speeds up the SVDARP calculation for instances with more than 12 requests. We found out that a greater improvement comes with larger instances compared to the original algorithm using backtracking.

Keywords: SVDARP, ridesharing, three-index formulation

Abstrakt

Tato bakalářská práce se zabývá zrychlením metody SVDARP za pomocí technik operační analýzy. Motivací práce bylo zrychlení SVDARP algoritmu v případech, kde se vyskytuje jako subproblém komplexnějších algoritmů, abychom tyto algoritmy zrychlili. Klíčovým aspektem bylo též zaměření na instance s větším počtem žádostí o přepravu, kde SVDARP algoritmus používající backtracking pracuje pomalu. Cílem bylo zanalyzovat aktuální výzkum v problematice řešení problému DARP a na základě této analýzy navrhnout exaktní metodu, kterou lze upravit pro řešení SVDARP. Na základě analýzy jsme vybrali tříindexovou formulaci problému DARP, kterou jsme upravili pro řešení SVDARP a následně řešili pomocí branch-and-cut metody. Formulaci jsme poté implementovali se všemi vylepšeními (eliminace hran, zúžení časových oken, vlastní řezné roviny v branch-and-cut algoritmu). Výsledkem práce je tak implementovaný SVDARP algoritmus a jeho následné porovnání s původní metodou využívající backtracking. Naším přístupem se podařilo zrychlit výpočet SVDARP u instancí s více jak 12 žádostmi o přepravu. Zjistili jsme, že s většími instancemi přichází i větší zrychlení oproti původnímu algoritmu využívající backtracking.

Klíčová slova: SVDARP, ridesharing, tříindexová formulace

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Transport in large cities has recently become an increasingly discussed topic. Politicians and the general public are increasing the requirements on public transport, especially regarding comfort, accessibility, and environmental issues. It is impossible not to notice the low occupancy of passenger vehicles during morning or evening traffic jams, where there are usually only one or two people in a car. We can easily deduce that some of them have almost the same route. A possible solution to this trend may be mobility-on-demand (MoD) transport solutions, which takes an imaginary place between public and passenger transport. MoD is a system of on-demand transportation services. This system involves passenger mobility and goods delivery, in order to optimize cost, journey time and other attributes. When we consider the passenger services, MoD includes carsharing, bikesharing, ridesharing and other shared means of transport [1]. One of the benefits of MoD is greater travel comfort, as it offers pickup and transport to a specific location outside established routes and public transport stops. In our work we focus on ridesharing. This way of transportation improves the efficiency and consequently, is more cost-efficient compared to transportation by private car. However, compared to public transport, it is less ecological and it is not so cost-efficient.

As mentioned by Ho et al. [2], the first use of DAR (dial-a-ride) service was provided in 1970 in the USA in Mansfield. The service found application mainly for disabled people, who could use it to transport to the hospital for a medical examination, a store, or a nearest public transport stop. Another possibility of using the DAR mode of transport is to replace public transport in less busy times when it is not worthwhile for carriers to provide services, or in less populated areas, where public transport also does not pay off [2].

SVDARP is a problem we need to solve in MoD systems without ridesharing, but also, it appears as a subproblem of a more complex ridesharing algorithms. The primary motivation of this work is to speed up the calculation of the SVDARP in order to speed up those complex algorithms. We also focus on big instances, where the SVDARP algorithm using backtracking is slow.

In the Chapter 2, we formulate dial-a-ride problem (DARP) and single-vehicle DARP (SVDARP), which are problems associated with ridesharing. We also present different techniques of solving DARP. Based on the literature research, we selected a specific solution – three-index formulation. Chapter 3 describes the three-index formulation and also introduces techniques to optimize it. The theoretical part is followed by Chapter 4 about implementation. In the Chapter 5 we compare and evaluate our implemented SVDARP algorithms with the SVDARP algorithm using backtracking.

Chapter 2

DARP Solution Methods

In this chapter, we introduce the problem of DARP and SVDARP. Furthermore, we outline the methods by which we can solve these problems.

2.1 DARP

Dial-a-ride problem (DARP) is a generalization of other vehicle routing problems like Pickup and Delivery Vehicle Routing problems (PDVRP) and the Vehicle Routing Problem with Time Windows (VRPTW). The main difference between DARP and other routing problems is the emphasis on passanger transportation and their comfort [3]. It is critical to consider that we plan a customer's route because the time of planned journey should be appropriate, and at the same time, we should not transport customers without any intermediate stop due to loss of efficiency [4].

DARP is a mathematical problem that deals with providing door-to-door transportation for people. This problem assumes that a set of users will create requests specifying the location from which they want to travel and their destination. These requests also specifies the pickup and drop-off times in time intervals called time windows. We also specify the service time (time needed for pickup/drop-off customer) and maximum travel time for each user. The maximum length of the entire journey, the vehicle's capacity, the starting and destination locations of the vehicle are specified for each vehicle [5].

We can solve DARP in two modes, static or dynamic. In the case of static, we know all requests before we plan the route. In the case of dynamic mode, we obtain requests during the route, and the planning is adjusted in real-time to meet all conditions [6].

We can also divide DARP into other categories; it can be deterministic or stochastics. In deterministic DARP, we know information with certainty when we make a decision. In a stochastic DARP, we assume some degree of uncertainty.

Information may be uncertain or unknown, but we know the probability and distribution of possible values in decision time. This uncertainty may relate, for example, to predict future requests or to possible delays due to congestion [2].

2.1.1 SVDARP

Single-vehicle DARP is one of the simplified versions of DARP. As the name suggests, this version only deals with the DARP solution for one vehicle. All other DARP conditions remain the same. We still try to plan the optimal route based on previously known data from users, i.e., their location, pickup and drop-off times specified by time windows, service duration. We also get data specified for the whole SVDARP problem: maximum ride time, maximum capacity of the vehicle, the maximum length of the entire journey, start and final location of the vehicle.

SVDARP is a problem we need to solve in MoD systems without ridesharing, but also, it appears as a subproblem of a more complex ridesharing algorithms. The primary motivation of this work is to speed up the calculation of the SVDARP solution so that the whole algorithm for solving DARP runs faster.

In some algorithms for solving DARP, SVDARP occurs as a subproblem. Our goal is to speed up the SVDARP algorithm so that the whole algorithm for solving DARP runs faster.

2.2 Exact Algorithms

Exact methods are mainly based on branch-and-bound algorithms, and they try to solve primarily deterministic and static problems. A disadvantage of exact algorithms is that they can solve only a limited instance size in a reasonable amount of computional time [2].

2.2.1 Branch-and-bound

The branch-and-bound (B&B) is a name for the algorithm that allows us to find exact solutions for some NP-hard problems. B&B also helps us solve integer programming problems.

Using branching, we divide a master problem into a set of subproblems which are solved separately. The solution of each subproblem is upper bound for all branches of this subproblem. If explored subproblem cannot produce a better solution than the current best solution, we discard the subproblem's whole branch. We recursively apply branching and bounding for all subproblems. When the tree search is complete, best solution is returned [7, 8].

When solving integer programming problems, we relax integrality constraints, and then we solve LP relaxation of the problem. In branching, we introduce additional constraints of subproblems to reduce the feasible area of the problem and keep all optimal integral solutions [8].

2.2.2 Branch-and-cut

Branch-and-cut (B&C) is a method that combines two algorithms, previously defined B&B algorithm and cutting planes algorithm. Using the cutting planes algorithm, we add cutting planes to subproblems generated by the B&B method in order to strengthen the LP relaxation of the problem. And therefore in B&C, we improve the LP relaxation by removing an optimal solution of LP relaxation to better approximate the integer programming problem [9].

The first branch and cut algorithm for DARP was introduced in 2006 [6]. In this work, the authors presented the three-index formulation for DARP and algorithms for eliminating unnecessary edges and narrowing time windows. This work also presents new valid inequalities for DARP and valid inequalities previously introduced for similar problems (traveling salesman, the vehicle routing, pickup, and delivery problems). These inequalities for similar problems are still applicable to the DARP algorithm [6].

Another article focused on the branch and cut algorithm was published a year later, in 2007. In this article, the authors present a two-index formulation together with its valid inequalities. This two-index formulation proved to be more efficient than the already mentioned three-index formulation, which enabled solving bigger instances [10].

2.2.3 Branch-and-price

Branch-and-price (B&P) algorithms are based on column generation, but also on B&B. This technique allows us to manage larger instances of mixed-integer linear programming (MILP) compared to B&B method. In the branch and price algorithm, we divide the problem into a master problem and a pricing subproblem. The master problem tries to reduce the complexity by excluding the columns from the LP relaxation. Subsequently, when we solve the pricing subproblem at each node of the B&B tree, it generates new columns, which can be added to the master problem. With this step, we reduce the B&B tree and improve the LP relaxation [2].

One of the works dealing with the branch and price algorithm in DARP is work by Parragh et al. [11]. Using this algorithm, the authors managed to solve instances with a maximum size of 40 requests [11].

2.2.4 Branch-and-price-and-cut

Branch-and-price-and-cut (B&P&C) algorithm combines the advantages of both previous methods. When executing the procedure, this algorithm adds cutting planes to LP relaxations. The algorithm also creates problems of significantly smaller size through the columns generated by the subproblem solution [2].

In 2014, the most efficient exact algorithm was introduced in [12]. It introduced a column generation algorithm, which can handle both dynamic and ordinary time windows. In the branch-and-cut part, the authors of this work used the presented valid inequalities from [6] and [10]. This algorithm was able to solve all instances from the dataset presented in [6]. This dataset contains instances for 2 to 8 vehicles, and the largest instance has 96 requests. At the same time, this algorithm was able to solve these instances one order of magnitude faster than previous exact approaches based on B&C or B&P&C [12].

2.3 Heuristics and Metaheuristics Algorithms

Unlike exact methods, heuristic and metaheuristic methods can find an optimal solution, but it is not guaranteed. Thanks to this relaxation, they are faster and able to solve much bigger instances, and consequently, meet real applications' needs. According to [13], metaheuristics is a general algorithm that can be applied to various optimization problems, while heuristics is an algorithm for solving a specific problem.

2.3.1 Insertion Heuristics

One of the first insertion heuristics was described by the authors of [14]. This algorithm consists of two steps: finding a possible customer assignment to the vehicle and subsequent optimization, which tries to minimize the cost of serving the customer with this vehicle. It sequentially tries to assign requests among all possible insertion among all vehicles to satisfy the requests and secure the smallest price increase. The authors, using this method, were able to find solutions to problems that had up to 2500 requests, which served 30 cars [14].

Nowadays, this heuristics is constantly improved and widely used. Newer and more advanced versions of this heuristics can be found in the articles [15] and [16]. The authors of these articles were able to speed up this algorithm so that they solved a large number of instances.

As mentioned in [2], insertion heuristics can be very useful in solving dynamic DARP because they can quickly find a possible solution. There is also mentioned, that insertion heuristics are often used as an algorithm for computing n initial solutions for other heuristics methods.

2.3.2 Tabu Search

Tabu search is a metaheuristics that allows local search algorithms to leave the local optimum during the search. The main element of this metaheuristics is the so-called taboo list of a certain length, on which it writes the last visited positions. As long as these positions are on the list, the algorithm excludes them from the

search. This limitation allows us to make moves that will not improve the current solution, but we can escape from the local optimum and find a better solution [17].

One of the first articles about tabu search for DARP solutions is [4]. The authors were able to solve instances that have up to 395 requests using this metaheuristics. As the authors mention in their conclusion, this algorithm is also very flexible, and it can be easily extended to solve a problem with many depots or with multiple vehicle types.

2.3.3 Variable Neighborhood Search

Variable neighborhood search is another metaheuristics improving local search. The main idea of this metaheuristics is to regularly change the neighborhood, which ensures that we find a local minimum, but also that we can escape from this local minimum [18].

The first variable neighborhood search algorithm for DARP was introduced in 2010. Using this approach, the authors of this work were able to find a better solution for 16 of the 20 instances [19]. Calculations were performed on instances from the article [4].

2.3.4 (Adaptive) Large Neighborhood Search

Another way to solve the DARP problem is a large neighborhood search. This is a metaheuristics based on two methods: destroy and repair. Firstly we destroy part of the solution, secondly, we use the repair method to insert the destroyed nodes so that we get the best possible resulting solution [20].

Adaptive large neighborhood search is an extension of the large neighborhood search heuristics, in which we can use multiple destroy and repair methods. Each of these methods has a weight that changes dynamically during the algorithm [20].

In 2019 an adaptive large neighborhood search metaheuristics for DARP was introduced. That is not the first usage of large neighborhood search in DARP. However, it outperformed all previously created algorithms on the instances presented in [4]. The authors of this work followed on previously presented works, which they managed to improve [21].

2.3.5 Genetic Algorithms

Genetic algorithms are metaheuristics that are based on evolution. At the beginning of the algorithm, we generate initial population. We randomly select individuals using a probability that depends on the individual member's quality from this population. Then we randomly apply crossover operations to get offspring. We may also add a mutation phase during crossing. We then use the resulting solutions to repeat this process of evolution until we have a good enough population [22].

Related work to genetic algorithms in DARP is [23], where the authors created a model of a genetic algorithm based on a general model. Another work dealing with this topic is [24], where authors propose another genetic algorithm model.

2.3.6 Hybrid Algorithms

The number of algorithms that use more than one method to solve DARP has increased in recent years. We call such algorithms hybrid algorithms. We can divide hybrid algorithms into several categories depending on the methods used in the algorithm. We can have a hybrid algorithm consisting of multiple metaheuristic algorithms and a combination of an exact algorithm and metaheuristics. The later approach Parragh et al. introduce in [11], where the problem is solved using the branch and price algorithm, but the subproblems are then solved using the variable neighborhood search.

Chapter 3

Methodology

This work aims to find the fastest possible exact algorithm for solving SVDARP.

First, we introduce a mathematical formulation for solving DARP: its original version and the version for SVDARP. Subsequently, we present individual valid inequalities that apply to this formulation. In the next section 3.2, we focus on preprocessing possibilities to simplify the created model as much as possible. The last two sections deal with the search of violated valid inequalities and their application, i.e., the addition of cutting planes.

3.1 Backtracking Algorithm

In this section, we introduce the backtracking algorithm for the SVDARP. This algorithm is already implemented and our goal was to replace it with a faster algorithm.

The backtracking is a recursive algorithm using a depth-first search. In each node, we try to add requests to the current plan so that all constraints of the DARP algorithm are met. If we are in a node, in which it is no longer possible to add another request to the plan, we return to the node one level higher and we change its value. We recursively repeat this operation until we find a solution. Normally, the algorithm searches this tree until it finds a solution. In our case, the algorithm is still searching to find the best solution.

3.2 Three-index Formulation

In this section, we present a mathematical three-index formulation of the DARP as it was introduced in [3]. It is a MIP (mixed-integer program) which allow us to solve DARP using MIP solvers.

As in the work [3] and [6], let n be the total number of requests. DARP is formulated on a complete directed graph $G = (V, A)$, where $V = P \cup D \cup \{0, 2n+1\}$.

$P = \{1, ..., n\}$ is a set of pickup nodes, $D = \{n+1, ..., 2n\}$ is a set of drop-off nodes, 0 represents a start depot and $2n+1$ represents a destination depot. Each request consists of the pickup node i and the drop-off node $n+i$, where $i \in P$ and $n+i \in D$. K denotes the set of vehicles. Each vehicle $k \in K$ has a maximum route duration T_k and a maximum capacity Q_k. With each node $i \in V$ is coupled a time window $[\, e_i, l_i]$, where e_i is the earliest arrival time and l_i is the latest arrival time. For each node $i \in V$ is also defined a service duration $d_i > 0$, $d_0 = d_{2n+1} = 0$ and a load q_i, so that $q_0 = q_{n+1} = 0$ and $q_i = q_{n+i}$ $\forall i \in \{1...n\}$. With each $arc(i, j) \in V$ is associated the travel time t_{ij} and a routing cost c_{ij}. L is the maximum ride time of user.

We have a three-index binary variable x_{ij}^k. If $x_{ij}^k = 1$, $arc(i, j)$ is passed by a vehicle $k \in K$. Let u_i^k be the variable representing the start of the service at node $i \in V$ for the vehicle $k \in K$, r_i^k is a variable of the ride time of the user i in the vehicle $k \in K$ and w_i^k is a variable defining the number of customers in the vehicle $k \in K$ after visiting the node $i \in V$. Variables r_i^k and w_i^k are integer variables, u_i^k is a continuos variable. Minimize

$$\sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij}^k x_{ij}^k \tag{3.1}$$

subject to:

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \qquad (i \in P), \tag{3.2}$$

$$\sum_{i \in V} x_{0i}^k = \sum_{i \in V} x_{i,2n+1}^k = 1 \qquad (k \in K), \tag{3.3}$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{n+i,j}^k = 0 \qquad (i \in P, k \in K), \tag{3.4}$$

$$\sum_{j \in V} x_{ji}^k - \sum_{j \in V} x_{ij}^k = 0 \qquad (i \in P \cup D, k \in K), \tag{3.5}$$

$$u_j^k \geq (u_i^k + d_i + t_{ij}) x_{ij}^k \qquad (i, j \in V, k \in K), \tag{3.6}$$

$$w_j^k \geq (w_i^k + q_j) x_{ij}^k \qquad (i, j \in V, k \in K), \tag{3.7}$$

$$r_i^k = u_{n+i}^k - (u_i^k + d_i) \qquad (i \in P, k \in K), \tag{3.8}$$

$$u_{2n+1}^k - u_0^k \leq T_k \qquad (k \in K), \tag{3.9}$$

$$e_i \leq u_i^k \leq l_i \qquad (i \in V, k \in K), \tag{3.10}$$

$$t_{i,n+i} \leq r_i^k \leq L \qquad (i \in P, k \in K), \tag{3.11}$$

$$max\{0, q_i\} \leq w_i^k \leq min\{Q_k, Q_k + q_i\} \qquad (i \in V, k \in K). \tag{3.12}$$

Objective (3.1) is to minimize the price of serving all requests for transportation. Constraint (3.2) ensures that each request is served by exactly one vehicle, constraint (3.3) then guarantees that each vehicle starts and ends at a given depot. Constraint (3.4) tells us that if we start processing a request, we must complete

it, while constraint (3.5) guarantees that if we get into a node, we also leave it. Constraint (3.5) does not apply to the depot nodes. Constraint (3.6) ensures that values of service start variable are correct in each node, constraint (3.7) then determines the number of people in the vehicle. Constraint (3.8) ensures that the variable r_i, which is the total time that the customer i spends in the car, has the correct value. Constraint (3.9) guarantees that the transport time does not exceed the maximum travel time of the vehicle, while constraint (3.10) guarantees that the customer is picked up and dropped off in the time window he has specified. Constraint (3.11) ensures that the customer's transport does not take longer than the specified maximum ride time of the user. Constraint (3.12) guarantees that the number of people does not exceed the maximum capacity of the vehicle and does not drop below 0.

During the implementation of the three-index formulation itself, we identified an error in the mathematical formulation. In the work [3] constraint defined as $r_i^k \geq u_{n+i}^k - (u_i^k + d_i)$, where r_i^k is the travel time of the user i in the vehicle k, u_i^k is the time of an arrival to the start node of the user i, u_{n+i}^k is the time of arriving in the end node of the user i and d_i is the customer service duration, was incorrectly stated. In this constraint, the operator *greaterthan* is incorrectly specified; in its place should be the operator equal. This error was corrected by reading [6] where this constraint is given correctly.

3.2.1 Single-vehicle Three-index Formulation

This work aims to speed up the calculation of the route for one vehicle. The original formulation considers one or more vehicles, but can be simplified to work with only one vehicle. To obtain the formulation for SVDARP, we have to omit the set of all vehicles K from the original formulation. We can also change the indexing of variables because our formulation considers only one vehicle. Except for this change, all constraints remain the same and apply to the single-vehicle formulation. Minimize

$$\sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{3.13}$$

subject to:

$$\sum_{j \in V} x_{ij} = 1 \qquad (i \in P), \tag{3.14}$$

$$\sum_{i \in V} x_{0i} = \sum_{i \in V} x_{i,2n+1} = 1 \tag{3.15}$$

$$\sum_{j \in V} x_{ij} - \sum_{j \in V} x_{n+i,j} = 0 \qquad (i \in P), \tag{3.16}$$

$$\sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} = 0 \qquad (i \in P \cup D), \tag{3.17}$$

$$u_j \geq (u_i + d_i + t_{ij})x_{ij} \qquad (i, j \in V), \qquad (3.18)$$
$$w_j \geq (w_i + q_j)x_{ij} \qquad (i, j \in V), \qquad (3.19)$$
$$r_i = u_{n+i} - (u_i + d_i) \qquad (i \in P), \qquad (3.20)$$
$$u_{2n+1} - u_0 \leq T \qquad (3.21)$$
$$e_i \leq u_i \leq l_i \qquad (i \in V), \qquad (3.22)$$
$$t_{i,n+i} \leq r_i \leq L \qquad (i \in P), \qquad (3.23)$$
$$max\{0, q_i\} \leq w_i \leq min\{Q, Q + q_i\} \qquad (i \in V). \qquad (3.24)$$

3.3 Valid Inequalities

In this section, we introduce valid inequalities. Valid inequality is a constraint introduced for a specific problem. Its purpose is to reduce the feasible area of this problem and at the same time, not to eliminate any integer solution. We do not need them for the three-index formulation itself, but these inequalities can enhance its LP-relaxation as a result of the reduced feasible area [6].

To describe these inequalities, we need to introduce additional sets and functions. We define $\bar{S} = \{i \in V \mid i \notin S\}$, where $S \subseteq V$. We also define $\delta(S) = \delta^+(S) \cup \delta^-(S)$, where $\delta^+(S) = \{(i, j) \in A \mid i \in S, j \notin S\}$ and $\delta^-(S) = \{(i, j) \in A \mid i \notin S, j \in S\}$. We also introduce $x(S) = \sum_{i,j \in S} x_{ij}$ [6].

3.3.1 Bounds on Time and Load Variables

We can strengthen u_i variables as follows [6]:

$$u_i \geq e_i + \sum_{j \in V \setminus \{i\}} max\{0, e_j - e_i + d_j + t_{ij}\}x_{ji}, \qquad (3.25)$$

$$u_i \leq l_i - \sum_{j \in V \setminus \{i\}} max\{0, l_i - l_j + d_i + t_{ij}\}x_{ij}. \qquad (3.26)$$

We can also strengthen the bounds to load variables w_i [6]:

$$w_i \geq max\{0, q_i\} + \sum_{j \in V \setminus \{i\}} max\{0, q_j\}x_{ji}, \qquad (3.27)$$

$$w_i \leq min\{Q, Q + q_i\} - \left(Q - \max_{j \in V \setminus \{i\}} \{q_j\} - q_i\right) x_{0i} - \sum_{j \in V \setminus \{i\}} max\{0, q_j\}x_{ij}. \qquad (3.28)$$

3.3.2 Subtour Elimination Constraints

To define subtour elimination constraints, we need to introduce additional sets. Let $\pi(S) = \{i \in P \mid n+i \in S\}$ and $\sigma(S) = \{n+i \in D \mid i \in S\}$, where $S \subseteq P \cup D$. π can be denoted as set of ancestors, σ as a set of successors [6].

For $S \subseteq P \cup D$ we define the so-called successor inequality (3.29) and also the so-called predecessor inequality (3.30) [6]:

$$x(S) + \sum_{i \in \bar{S} \cap \sigma(S)} \sum_{j \in S} x_{ij} + \sum_{i \in \bar{S} \setminus \sigma(s)} \sum_{j \in S \cap \sigma(S)} x_{ij} \leq |S| - 1, \tag{3.29}$$

$$x(S) + \sum_{i \in S} \sum_{j \in \bar{S} \cap \pi(S)} x_{ij} + \sum_{i \in S \cap \pi(S)} \sum_{j \in \bar{S} \setminus \pi(s)} x_{ij} \leq |S| - 1. \tag{3.30}$$

For the set $S \subseteq P \cup D$, where $S = \{i_1, i_2, ..., i_h\}$ the following two inequalities are valid for DARP [6]:

$$\sum_{j=1}^{h-1} x_{i_j,i_{j+1}} + x_{i_h,i_1} + 2\sum_{j=2}^{h-1} x_{i_j,i_1} + \sum_{j=3}^{h-1}\sum_{l=2}^{j-1} x_{i_j,i_l} + \sum_{n+i_p \in \bar{S} \cap \sigma(S)} x_{n+i_p,i_1} \leq h-1, \tag{3.31}$$

$$\sum_{j=1}^{h-1} x_{i_j,i_{j+1}} + x_{i_h,i_1} + 2\sum_{j=3}^{h} x_{i_1,i_j} + \sum_{j=4}^{h}\sum_{l=3}^{j-1} x_{i_j,i_l} + \sum_{i_p \in \bar{S} \cap \pi(S)} x_{i_1,i_p} \leq h-1. \tag{3.32}$$

3.3.3 Capacity Constraints

In capacity constraints, we limit the set S only to $S \subseteq P$ or $S \subseteq D$. Valid inequality is $x(\delta(S)) \geq 2R(S)$, where $R(S)$ is the minimum number of vehicles needed to serve all nodes in the set S. The calculation of $R(S)$ is demanding and therefore we replace it by the approximation $\lceil q(S)/Q \rceil$, where $q(S) = \left|\sum_{i \in S} q_i\right|$ [6].

3.3.4 Precedence Constraints

If we have a set $S = \{0, n+i\}$ for at least one $i \in P$, it holds that node i must be visited before node $n+i$, then $x(S) \leq |S| - 2$ and at the same time $x(\delta(S)) \geq 3$. Same statement also applies to $S = \{i, 2n+1\}$ for at least one $i \in P$ [6].

3.3.5 Generalized Order Constraints

Let us have the sets $U_1, \ldots, U_m \subset V$, which are disjunct to each other. Next, we have $i_1, \ldots, i_m \in P$ and it holds that $0, 2n+1 \notin U_l$ and $i_l, n+i_{l+1} \in U_1$ for $l = 1, \ldots, m$ where $i_{m+1} = i_1$. The following inequalities are valid for DARP [6]:

$$\sum_{l=1}^{m} x(U_1) \leq \sum_{l=1}^{m} |U_l| - m - 1 \tag{3.33}$$

$$\sum_{l=1}^{m} x(U_1) + \sum_{l=2}^{m-1} x_{i_1,i_l} \sum_{l=3}^{m} x_{i_1,n+i_l} \leq \sum_{l=1}^{m} |U_l| - m - 1 \tag{3.34}$$

$$\sum_{l=1}^{m} x(U_1) + \sum_{l=2}^{m-2} x_{n+i_1,i_l} \sum_{l=2}^{m-1} x_{n+i_1,n+i_l} \leq \sum_{l=1}^{m} |U_l| - m - 1 \tag{3.35}$$

3.3.6 Infeasible Path Constraints

Triangular inequality is not guaranteed in general DARP, but in this constraint we assume that the triangular inequality holds for travel times. This means that the inequality cannot be always used. For any oriented path $P = \{i, k_1, k_2, ..., k_p, n+i\}$, for which $t_{i,k_1} + d_{k_1} + t_{k_1,k_2+d_{k_2}+\cdots+t_{k_p,n+i}>L}$ is an inequality (3.36) valid for DARP [6].

$$x_{i,k_1} + \sum_{h=1}^{p-1} x_{k_h,k_{h+1}} + x_{k_p,n+i} \leq p - 1 \tag{3.36}$$

3.4 Preprocessing Steps

This section introduces ways how to easily reduce the time it takes to resolve a three-index formulation using preprocessing steps. These are mainly a time window tightening and an arc elimination.

3.4.1 Time Window Tightening

Using time window tightening, we can narrow the time windows of pickup and drop-off nodes so they are as small as possible and thus we reduce the complexity of the problem.

We can divide transport requests into two groups. In one group, a passenger travels from home to his destination. The passenger, therefore, specifies the time window in which he wants to get to the destination. In the case of a second group, a passenger returns home from his current destination, so it is important for him the time window in which he will be picked up from the destination [6].

If the passenger travels from home to the specific destination, we can set $e_i = max\{0, e_{n+i} - L - d_i\}$ and $l_i = min\{l_{n+i} - t_{i,n+i} - d_i, T\}$ in the start node. In the other case, when the passenger returns home, we can set the end node to $e_{n+i} = max\{0, e_i + d_i + t_{i,n+i}\}$ and $l_{n+i} = min\{l_i + d_i + L, T\}$ [6].

We can also narrow the time windows directly in the nodes that represent the depot. For the start and end depot we can use the same formulation for the minimum and maximum time, namely $e_0 = e_{2n+1} = \min_{i \in P \cup D}\{e_i - t_{0i}\}$, $l_0 = l_{2n+1} = \max_{i \in P \cup D}\{l_i + d_i + t_{i,2n+1}\}$ [6].

3.4.2 Arc Elimination

Three-index formulation is defined on a complete graph. However, we can exclude some edges from the graph because we know they do not belong to the feasible solution [6].

We can remove all the edges leading to themselves ($arc(i,i)\ i \in V$). Furthermore, we can remove all edges leading from a drop-off node to a pickup node between nodes of the same request ($arc(n+i,i)\ \forall\ i \in P$). Unnecessary edges are also those that lead from the initial depot to any drop-off node ($arc(0, n+i)\ for\ i \in P$) as well as all edges leading from any pickup node to the end depot ($arc(i, 2n+1)\ for\ i \in P$) [6].

We can also remove all edges that cannot be traversed without violating the time window range of the start and end node ($arc(i,j)$ with $i,j \in V$ and $e_i + d_i + t_{ij} > l_j$). In the same way, we can remove the edge leading from the pickup node of the last customer to the pickup location of another customer. Also we can remove the edge leading from this customer to the drop-off node of the currently transported customer. In both cases, we can remove the edges under the condition of exceeding the maximum travel time of the already transported customer ($arcs(i,j)$ and $(j, n+i)$ where $i \in P,\ j \in V$ if $t_{ij} + d_j + t_{j,n+j} > L$) [6].

There are also other elimination rules. These rules were first introduced in [25] and they mainly use a combination of time windows and pairing constraints. If the path $\mathcal{P} = \{j, i, n+j, n+i\}$ is infeasible, then we can exclude $arc = (i, n+j)$. If the path $\mathcal{P} = \{i, n+i, j, n+j\}$ is infeasible, then we can exclude $arc = (n+i, j)$. Furthermore, we can eliminate $arc = (i,j)$ if the paths $\mathcal{P}_1 = \{i, j, n+i, n+j\}$ and $\mathcal{P}_2 = \{i, j, n+j, n+i\}$ are infeasible, as well as we can exclude $arc = (n+i, n+j)$ if the paths $\mathcal{P}_1 = \{i, j, n+i, n+j\}$ and $\mathcal{P}_2 = \{j, i, n+i, n+j\}$ are infeasible [6].

3.5 Initial Pool of Inequalities

The authors of the work [6] introduced the initial pool of inequalities. These inequalities are checked for each node in the B&B tree. If an inequality is violated, it is added as a cutting plane, so we achieve better LP-relaxation. The authors have chosen these constraints so the time required for processing one node does

not increase significantly, and at the same time, the number of constraints in the pool is not too large [6].

To the initial pool of inequalities, we add all *bounds on time and load variables* valid inequalities that can be generated based on the current data and inequalities presented in section 3.3.1. For all $i, j \in P \cup D$ [6].

We will also add simple variants of the *subtour elimination constraints* presented in section 3.3.2 to the initial pool of inequalities. These inequalities prevent the solution from being split into several independent routes. Inequality (3.29) generates these cases for each pair of $i, j \in P$ [6]:

- $S = \{i, j\} \Rightarrow x_{ij} + x_{ji} + x_{n+i,j} + x_{n+j,i} \leq 1$,
- $S = \{i, n+j\} \Rightarrow x_{i,n+j} + x_{n+j,i} + x_{n+i,n+j} \leq 1$,
- $S = \{i, n+i, j\} \Rightarrow x_{ij} + x_{ji} + x_{i,n+i} + x_{j,n+i} + x_{n+i,j} + x_{n+j,i} + x_{n+j,n+i} \leq 2$.

Inequality (3.30) then provides these cases for each pair of $i, j \in P$ [6]:

- $S = \{n+i, n+j\} \Rightarrow x_{n+i,n+j} + x_{n+j,n+i} + x_{n+i,j} + x_{n+j,i} \leq 1$,
- $S = \{i, n+j\} \Rightarrow x_{i,n+j} + x_{n+j,i} + x_{ij} \leq 1$,
- $S = \{i, n+i, n+j\} \Rightarrow x_{i,n+j} + x_{n+j,i} + x_{i,n+i} + x_{n+j,n+i} + x_{n+i,n+j} + x_{ij} + x_{n+i,j} \leq 2$.

Inequalities (3.31) and (3.32) generate the following constraints [6]:

- $S = \{n+i, j, n+i\} \Rightarrow x_{n+i,j} + 2x_{j,n+i} + x_{ji} + x_{i,n+i} + x_{n+j,n+i} \leq 2$,
- $S = \{i, n+i, n+j\} \Rightarrow x_{i,n+i} + x_{n+i,n+j} + x_{n+j,i} + 2x_{i,n+j} + x_{ij} \leq 2$.

For the subset $S = \{i, j, n+i, n+j\}$ we also generate an inequality for all $i, j \in P$ where $x(S) \leq 3$ [6].

Precedence inequalities generated from section 3.3.4 are also added to the initial pool of inequalities. These constraints specify that some nodes cannot be visited before visiting another node. Precedence inequalities generate following constraints for $i, j \in P$ [6]:

- $S = \{0, i, n+j\} \Rightarrow x_{0i} + x_{i,n+j} + x_{n+j,i} \leq 1$,
- $S = \{i, n+j, 2n+1\} \Rightarrow x_{i,n+j} + x_{n+j,i} + x_{n+j,2n+1} \leq 1$,
- $S = \{0, i, n+i, n+j\} \Rightarrow x_{0i} + x_{i,n+i} + x_{i,n+j} + x_{n+j,i} + x_{n+i,n+j} + x_{n+j,n+i} \leq 2$,
- $S = \{i, j, n+j, 2n+1\} \Rightarrow x_{ij} + x_{ji} + x_{i,n+j} + x_{n+j,i} + x_{j,n+j} + x_{n+j,2n+1} \leq 2$.

To our initial pool of inequalities we also include a *generalized order inequlity*, which is generated from the constraints from section 3.3.5, for $i, j \in P$ [6]:

- $x_{i,n+j} + x_{n+j,i} + x_{n+i,j} + x_{j,n+i} \leq 1$.

We also check *infeasible path constraint* which are defined in section 3.3.6 for each pair of request. These inequalities are part of the initial pool of inequalities. If the inequality $t_{ij} + d_j + t_{j,n+j}d_{n+j} + t_{n+j,n+j} > L$ is true, then we generate the following constraint where $i, j \in P$ [6]:

- $x_{ij} + x_{j,n+j} + x_{n+j,n+i} \leq 1$.

3.6 Separation Heuristics

This section describes the separation heuristics that we use to identify other violated inequalities. We identify these inequalities using various heuristics to find other violated inequalities that allow us to add our own cutting plane. Therefore, we improve the current LP-relaxation. This heuristics consists of the following heuristics that are executed sequentially.

3.6.1 Subtour Elimination Constraints

To find the violation of inequality (3.29), we use tabu search heuristics and the fact that $2x(S) + x(\delta(S)) = 2\,|S|$ is a feasible integer solution. Using the equation

$$x(\delta(S)) - 2 \sum_{i \in \bar{S} \cap \sigma(S)} \sum_{j \in S} x_{ij} - 2 \sum_{i \in \bar{S} \setminus \sigma(S)} \sum_{j \in S \cap \sigma(S)} x_{ij} < 2 \tag{3.37}$$

we can find sets S that violate inequality (3.29). In the case of tabu search, we start the search with an empty set S. In each iteration, we add or remove a node so that the left side of (3.37) is as small as possible. If we remove the node, we must not insert it again for a certain number of iterations. We perform this iteration 25 times in our implementation. For each set S, we check whether inequality (3.31) is not violated. For (3.31) is i_1 node with the largest outgoing flow. We apply the same procedure for inequality (3.30) together with (3.32) where node with the largest incoming flow is i_1 [6].

3.6.2 Capacity Constraints

To find sets S such that $q(S) > Q$ and at the same time $x(\delta(S)) < 4$, we use the tabu search. We start with a random subset of $S \subseteq P$ or $S \subseteq D$. In each iteration, we add or remove a node with S to minimize the value of $x(\delta(S))$ and at the same time satisfy the condition $q(S) > Q$. Tabu search runs for 25 iterations [6].

3.6.3 Generalized Order Constraints

We use a simple heuristics to identify inequality violations of (3.33). We focus only on the special case where $m = 2$ and $|U_1| = |U_2| = 3$. Firstly, we create subsets where $U_1 = \{i, n + j\}$ and $U_2 = \{j, n + i\}$ for all $i, j \in P$. Secondly, we try to find nodes k_1 and k_2 so that the values of $x(U_1)$ and $x(U_2)$ are maximal. Finally, we check whether the inequality is not violated [6].

We also use heuristics to identify the violation of inequality (3.34) and (3.35). We use a special case where $m = 3$ and $|U_1| = |U_2| = 2$. For each $i \in P$, we find $j \in P$, which maximizes $x_{i,n+j} + x_{n+j,i} + x_{ij}$ in the case of (3.34). In the case of (3.35) we find $j \in P$ for each $i \in P$, which maximizes $x_{i,n+j} + x_{n+j,i} + x_{n+i,n+j}$. In both cases, we look for $k \in P$ for which the left side of the inequality is maximal [6].

3.6.4 Infeasible Path Inequalities

We use path-construction heuristics to find violated inequalities. We apply this heuristics to each user $i \in P$. The heuristics starts at node i and looks for another node at each step to maximize the x_{k_j,k_l} values, where k_j is the current node and k_l is the next node in path. The heuristics may end in three ways: when a cycle is detected, when we reach the node $n + i$, or the node $2n + 1$. If the heuristics ends in node $2n + 1$, we check the path to see if it does not violate inequality (3.36) [6].

Chapter 4

Implementation

The main goal of this work is to speed up the calculation of the optimal solution for SVDARP. Finding the optimal route for one vehicle is a crucial element of the vehicle group assignment (VGA) algorithm [26]. This algorithm finds the optimal solution by generating all possible groups of requests. The VGA algorithm then optimally assigns these groups to individual vehicles. Our implemented SVDARP algorithm can also be used to solve these request groups.

At the beginning of the implementation, we had DARP benchmark software[1]. The purpose of this program is to create a unified interface for running and exporting the results of various methods that can be used to solve DARP. Currently, this program can solve DARP using insertion heuristics and VGA. There is also special runner for SVDARP, which we used to test our and backtracking SVDARP independently. The VGA method is important for our implementation, because our goal is to speed up SVDARP, which is solved as a subproblem in VGA method. Whole DARP benchmark software is written in C++ and uses Gurobi Optimizer as an integer-linear programming (ILP) solver.

In the Figure 4.1 we see a simplified illustration of the DARP benchmark using the VGA method or the SVDARP algorithm independently. In the `DARP_benchmark class`, based on the input arguments, the algorithm selects a solution using `VGA_solver` or `SVDARP_Runner`. It can solve both of these problems using our implemented algorithm in the `Single_vehicle_DARP_optimized_ILP` class or the original algorithm in the `Single_vehicle_DARP_optimized` class.

After studying current researches about solving DARP, we decided to implement the three-index formulation for DARP as described in Chapter 3.

The implementation consists of two classes: `Single_vehicle_DARP_optimized_ILP` and `SVDARP_Callback`. In the `Single_vehicle_DARP_optimized_ILP` class the implementation of the three-index formulation is located. This class has the same public methods as the original backtracking class `Single_vehicle_DARP_optimized`.

[1] https://gitlab.fel.cvut.cz/fiedlda1/darp-benchmark/-/tree/threeIndexFormulation

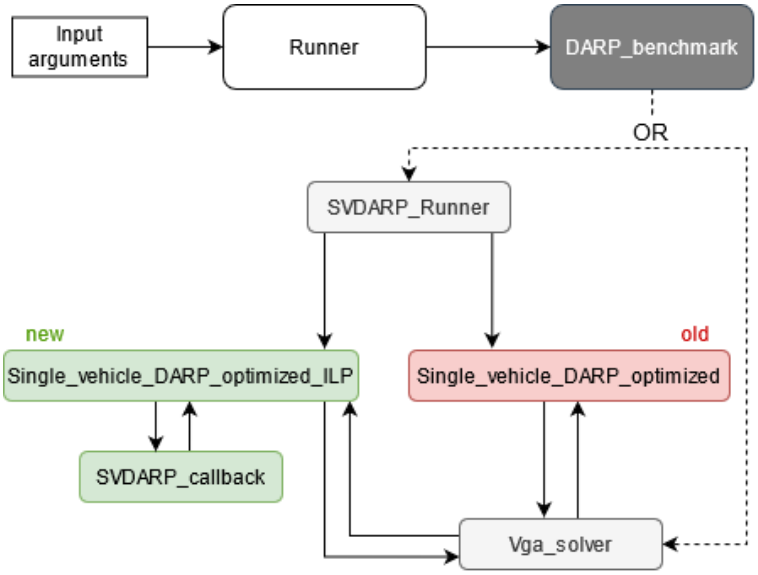


Figure 4.1: DARP benchmark software diagram showing the evaluation of VGA or SVDARP algorithm.

In this class, there are also all supporting methods for the proper creation of the formulation, methods for time-window tightening, and methods for arc elimination. The `SVDARP_Callback` class inherits from the `GRBCallback` class. The `GRBCallback` class allows us to add our cuts to the currently solved problem. The `SVDARP_Callback` class contains all the methods for finding violated inequalities and adding own cutting planes as we describe in Section 3.5 and 3.6.

In the Figure 4.2 we see an illustration of the implementation of sections from the Chapter 3 converted to individual methods in our implementation.

4.1 Auxiliary Programs

We also created other utilities for creating and processing data. In the `VGA_solver` class, we created the save_request_group method. This method is used in compute_groups_for_vehicle method, which generates all possible groups of SVDARP requests. Our method saves the created SVDARP instances to a disk. Therefore, we can create a large number of test instances of the SVDARP algorithm.

We also created two simple python scripts. The first script repeatedly allows us to run our chosen version of the program with all the parameters. We created the method to run one instance, the method to run all instances in file and the method to run instances in multiple files. The script has not input arguments from the command line. However, we created blocks of code in the main method that

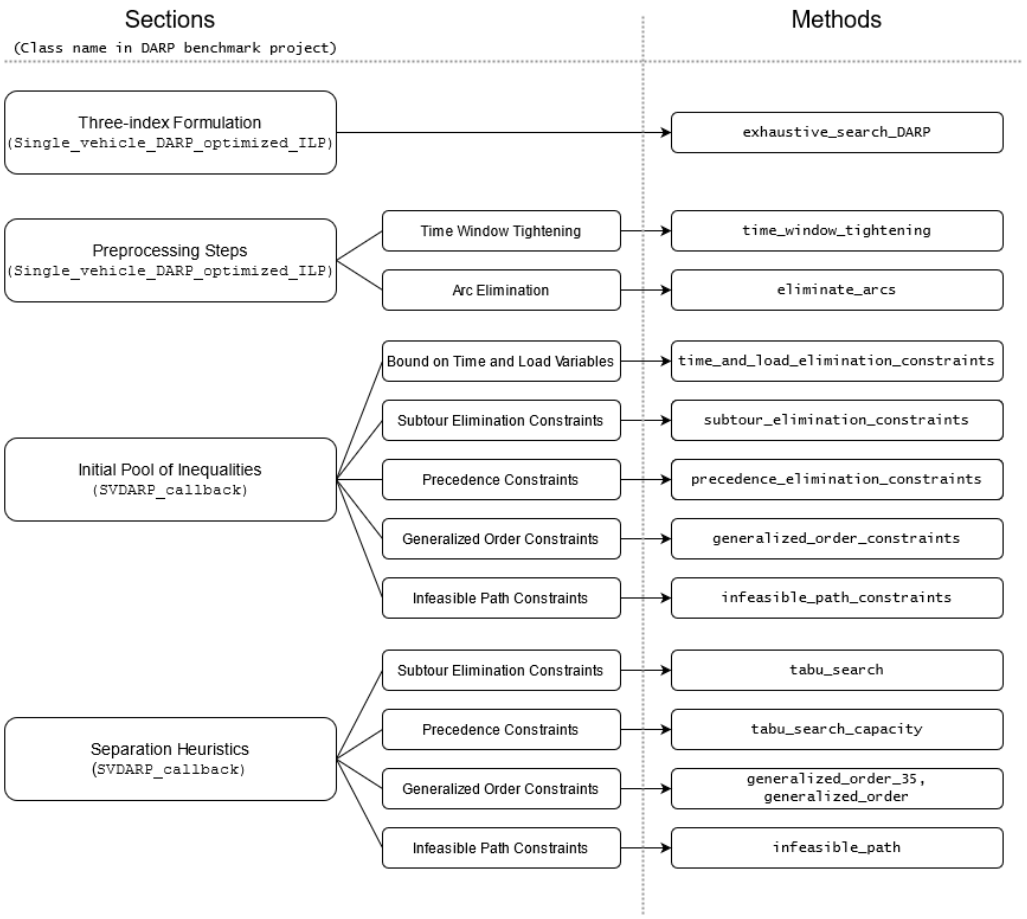


Figure 4.2: Methodology sections mapped to implemented SVDARP classes and methods.

allow us to run the evaluation of all instances at once. Using this script, we can process a large number of input instances automatically.

The second script allows us to process all the outputs from individual calls and unify this data into one JSON file. We use this JSON file in this script to automatically create all the tables and figures that are in this thesis.

Chapter 5

Results

In this chapter we compare the performance of our implemented algorithms for SVDARP calculation with the original algorithm using backtracking. We always compare 3 versions of the SVDARP algorithm: the original version which used backtracking (BCK), three-index formulation with all preprocessing procedures (TIF) and three-index formulation with all preprocessing procedures and with own cuts (TIFOC).

We test the implementation of SVDARP methods directly on the instances we created for SVDARP, but also in the VGA method which can process standard instances for DARP.

5.1 Stand Alone SVDARP

In this section, we perform a comparison of stand alone SVDARP algorithms, which are not used as a subproblem of another algorithm.

5.1.1 Instances

We use two sets of instances to test SVDARP algorithms. The first set contains instance groups. Each group has 100 instances with the same request size. For each request size (5-15), we have a group of feasible and infeasible instances. To generate these instances, we used a VGA solver, which creates instances of various sizes within its functionality. We also created an additional code that saved these instances. We saved all generated instances and then reduced their number using the modulo function so that we have 100 instances for each group. The second set consists of individual instances. In this set, we have a total of 28 instances containing request size between 5 to 18; for each size from this interval, there are always two instances in the set. These instances are obtained from already solved DARP instances, which were presented in [4]. These solved instances have 24-144 requests and to process them, we need 3-13 vehicles. We took a plan for

an individual vehicles from each solution we had available and created an instance from it that we know is feasible for the SVDARP algorithm.

5.1.2 Comparison

In Table 5.1 we see a comparison of the runtime of individual SVDARP algorithms on the first set of instances. In the columns, there is the average runtime of all instances in the group. For the methods TIF and TIFOC, we also present a comparison using a speedup. Their speedup was calculated as a proportion of values of BCK method and TIF and TIFOC methods. The names of the request groups refer to their instance sizes. When this name contains only a number, the instances in the group are feasible. The groups with infeasible instances are marked with "_inf".

Request group	BCK mean	TIF mean	TIF speedup	TIFOC mean	TIFOC speedup
05	0.12	10.80	0.01	12.60	0.01
05_inf	0.09	10.05	0.01	12.75	0.01
06	0.68	10.96	0.06	13.31	0.05
06_inf	1.92	9.41	0.20	11.01	0.17
07	1.68	12.26	0.14	15.63	0.11
07_inf	1.42	12.49	0.11	14.33	0.10
08	2.84	16.82	0.17	18.63	0.15
08_inf	2.04	10.47	0.19	14.65	0.14
09	6.28	14.41	0.44	17.73	0.35
09_inf	5.21	13.00	0.40	15.91	0.33
10	11.98	18.37	0.65	21.90	0.55
10_inf	8.15	13.05	0.62	18.65	0.44
11	18.47	17.78	1.04	24.97	0.74
11_inf	15.00	16.57	0.91	18.07	0.83
12	36.76	20.95	1.75	32.49	1.13
12_inf	30.14	17.57	1.72	18.93	1.59
13	64.91	21.11	3.07	35.54	1.83
13_inf	47.60	14.95	3.18	18.53	2.57
14	99.35	27.85	3.57	43.70	2.27
14_inf	66.19	15.14	4.37	20.45	3.24
15	157.57	29.36	5.37	52.99	2.97
15_inf	107.17	15.62	6.86	21.28	5.04

Table 5.1: Runtime (miliseconds) and speedup of new SVDARP implementations compared to the original algorithm on the first dataset.

Figure 5.1 shows a comparison of the runtime of instance groups from the first dataset.

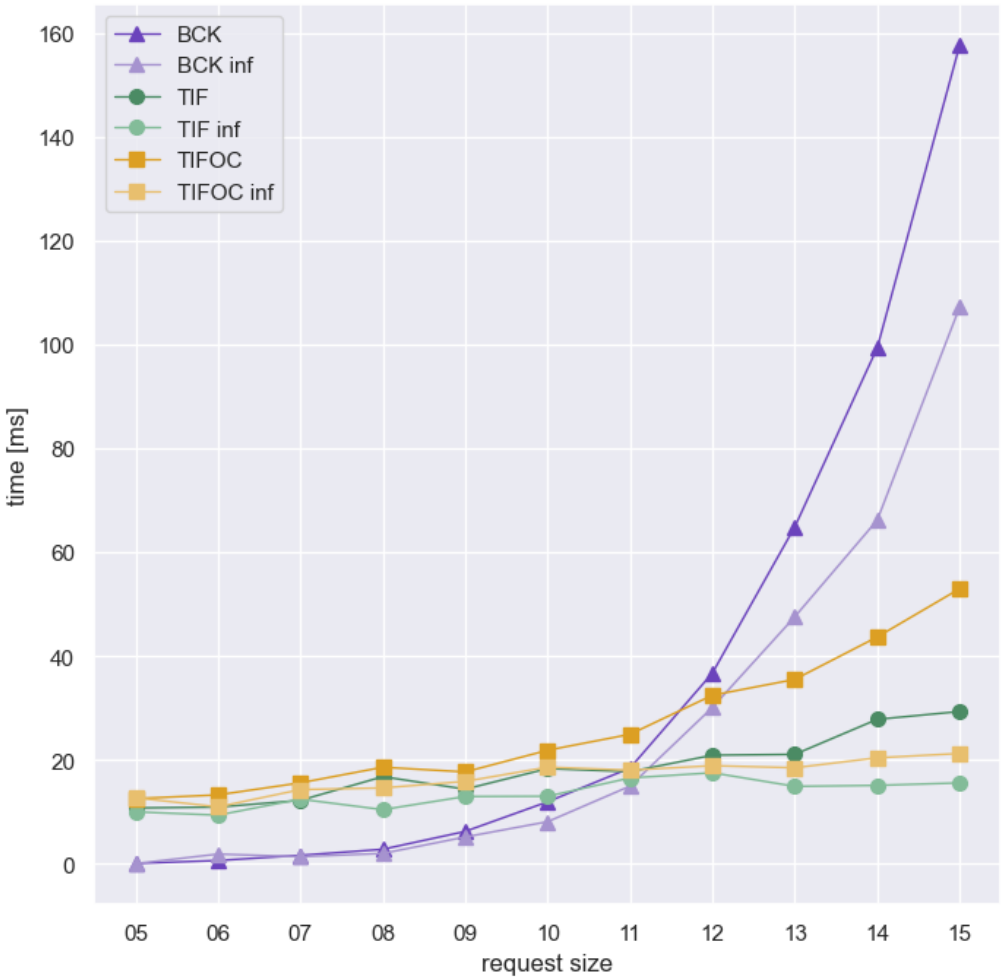


Figure 5.1: Runtime of different SVDARP implementation on the instance group from the first dataset.

In Figures 5.2 and 5.3 we compare the effect of individual preprocessing steps on TIF. We use other versions of the SVDARP algorithm, namely: three-index formulation with time windows (TIFTW), three-index formulation with arc elimination (TIFAE) and three-index formulation with no preprocessing procedures (TIFNO). Figure 5.2 compare effect of preprocessing steps on feasible groups, Figure 5.3 on infeasible groups. We compare the implementation of TIFNO with TIFTW, TIFAE, and TIF. We run these implementations on first dataset. We calculate the average runtime of each request group for each preprocessing method.

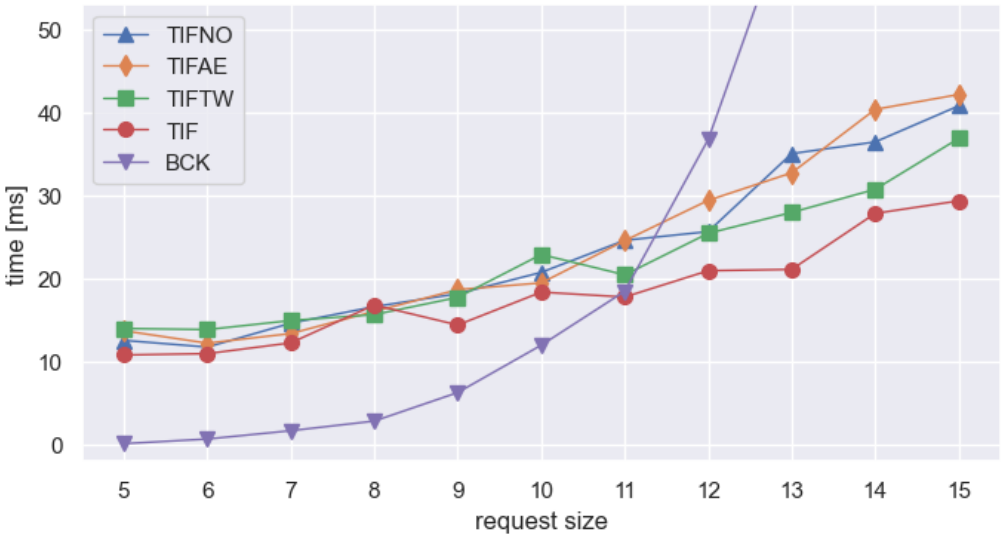


Figure 5.2: Effects of preprocessing steps on the feasible groups

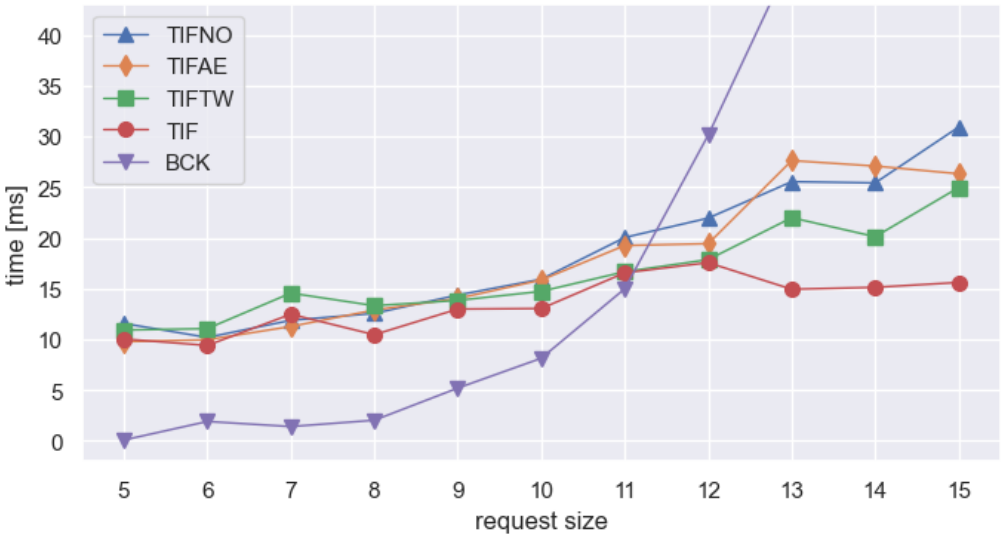


Figure 5.3: Effects of preprocessing steps on the infeasible groups

In Table 5.2, we see the runtime on the instances from the second set. We ran all instances five times and then averaged the results. Again, we compare the original algorithm with the implementation of TIF and TIFOC. As in the previous case, we also calculate a speedup over the original implementation.

Instance name	Request count	BCK mean	TIF mean	TIF speedup	TIFOC mean	TIFOC speedup
pr06.v2	10	1.6	15.2	0.11	12.4	0.13
pr15.v11	10	1.0	21.0	0.05	32.2	0.03
pr13.v5	12	7.8	55.9	0.14	110.4	0.07
pr19.v2	12	70.4	334.2	0.21	591.2	0.12
pr03.v1	14	12.2	203.8	0.06	1,312.3	0.01
pr17.v2	14	262.4	455.8	0.58	4,275.4	0.06
pr05.v10	16	439.8	52.3	8.41	408.0	1.08
pr17.v3	16	680.2	356.1	1.91	1,821.0	0.37
pr04.v3	18	73.2	86.3	0.85	149.3	0.49
pr17.v4	18	13,157.0	632.4	20.80	108,685.3	0.12
pr02.v5	20	5,597.0	97.2	57.58	70.6	79.28
pr12.v1	20	24,630.0	300.0	82.10	4,487.1	5.49
pr05.v1	22	2,076.0	314.0	6.61	1,339.9	1.55
pr20.v5	22	40,313.0	136.7	294.90	278.8	144.59
pr08.v3	24	566.4	54.2	10.45	92.3	6.14
pr12.v3	24	1,303,429.8	755.7	1724.80	31,449.6	41.45
pr01.v3	26	13,522.4	134.0	100.91	294.7	45.89
pr20.v2	26	980,286.8	190.1	5156.69	590.9	1658.97
pr03.v3	28	79,409.8	591.1	134.34	11,169.7	7.11
pr09.v6	28	124,205.0	606.9	204.65	15,616.9	7.95
pr05.v6	30	1,096,380.4	555.5	1973.68	6,573.6	166.79
pr06.v3	30	120,542.4	387.4	311.16	7,206.5	16.73
pr06.v5	32	221,016.6	547.7	403.54	9,606.3	23.01
pr10.v4	32	66,134.0	704.9	93.82	15,637.3	4.23
pr19.v1	34	-	4944.1	-	91547.4	-
pr20.v8	34	-	12358.7	-	59712.3	-
pr16.v2	36	-	7170.2	-	43811.3	-
pr16.v4	36	-	24357.4	-	34515.8	-

Table 5.2: Runtime (miliseconds) and speedup of SVDARP implementations compared to the original algorithm on the second dataset.

5.1.3 Discussion

From Figure 5.1 and the Table 5.1 it is obvious that we managed to accelerate the SVDARP algorithm using three-index formulation, but only for instances that have 12 or more requests. In our opinion, this is due to the fact that it always takes some minimal time to create and solve the ILP problem. The observation confirms the assumption that the solution time of instances using TIF for groups up to 10 requests takes a very similar time. The graphs also show that TIFOC is slower than TIF. This situation is probably caused by the overhead that accompanies

finding and creating our cuts. We can read from the table and graphs that the average time to solve infeasible instances is less than the average time to solve feasible instances.

From Figures 5.2 and 5.3 it can be read that the individual preprocessing procedures do not have a significant influence on the overall acceleration of the calculation. However, an implementation that includes time window tightening and arc elimination is 29 % faster than versions without any preprocessing. We can also notice that these steps speed up solving infeasible instances more than feasible.

From Table 5.2, where we have a comparison of instances from the second set, we can read that the time required to solve an instance depends not only on the size of the instance but also on the complexity of the instance. This situation can be seen, for example, on instances of size 12, where it takes much time for BCK to solve one instance of this size, while the other instance of size 12 is solved faster than some instances of smaller size. As in the first dataset, we can say that the BCK is more suitable for smaller instances, approximately up to 8 requests. At the same time we can say that TOFOC is slower than the version without its own cuts.

5.2 SVDARP in VGA

In this section, we test our SVDARP implementations in the VGA method. In the VGA method, SVDARP is repeatedly called on both large and small instances and also both on feasible and infeasible instances, which tests whether the use of our SVDARP algorithm is more advantageous overall.

5.2.1 Instances

As a test set, we use instances for DARP, which were introduced in [6]. From this set, we select only instances with a smaller number of requests (a2-16, a2-20, a3-18, a4-16) so that we can solve them with our algorithms. This dataset also contains even larger instances, but we were unable to solve any of them within the 3-hours time limit.

5.2.2 Comparison

In Table 5.3 we see a comparison of SVDARP algorithms when called from the VGA method. We run all algorithms five times on each instance and the table shows the average of all evaluations. For the new SVDARP algorithms, we also present the acceleration compared to the original method.

Instance name	Request count	BCK mean	TIF mean	TIF speedup	TIFOC mean	TIFOC speedup
a2-16	16	22,124	29,275	0.76	28,722	0.77
a2-20	20	6,941,211	1,648,283	4.21	1,724,930	4.02
a3-18	18	29,427	38,432	0.77	39,322	0.75
a4-16	16	1,018	5,461	0.19	5,592	0.18

Table 5.3: Runtime (miliseconds) and speedup of new SVDARP implementations in VGA algorithm compared to the original algorithm on the DARP instances.

5.2.3 Discussion

From Table 5.3 we can read that for instances with less than 20 requests, the use of new implementations using ILP does not pay off. The results from the previous section explain this behavior. In the case of instances with a request size smaller than 20, the VGA method mainly solves problems of a size up to 12 requests, in which is faster the SVDARP method using backtracking. In the case of instances with 20 or more requests, the advantage of the algorithm using TIF shows the benefit, because the method has to deal with larger requests. We should also note that if we look at the time difference for instance a2-16, the BCK algorithm runs for about 22.1 seconds, while for the TIF algorithm, it takes about 29.3 seconds to solve the instance, which makes a difference of 7.2 seconds. However, for instance a2-20, the BCK algorithm solves the instance under 2 hours, while the TIF algorithm solves the instance in less than 30 minutes, making a difference of more than 1.5 hours. Thus, we can say that SVDARP implementation using TIF is more advantageous in absolute numbers.

Chapter 6

Conclusion

This bachelor thesis focuses on finding and implementing an efficient exact algorithm for solving single-vehicle dial-a-ride problem (SVDARP). Our main motivation was to speed up the SVDARP algorithm, so that we could achieve speedup of runtime in the complex algorithms, where SVDARP is solved as a subproblem. The key part of the speedup was the acceleration of instances with large number of requests, where the previous algorithm was slow. We introduced a modified version of the three-index formulation for SVDARP. This version is based on the three-index formulation for dial-a-ride problem (DARP) presented in [6].

In the first part, we introduced DARP and SVDARP and we made an introduction into methods that can be used to solve these problems. Subsequently, we presented our exact method for solving SVDARP based on the DARP solution method from [6]. This method uses a three-index formulation, which can be solved using the branch-and-cut (B&C) algorithm. It also contains optimizations such as identifying and removing unnecessary edges, time windows tightening, identifying violated valid inequalities, and adding custom cuts. In the implementation chapter, we introduced the classes in which our implementation is located. We also presented utilities created to run our implementation and data processing easily. We found out that our method is more convenient for SVDARP if the instance has approximately 12 or more requests. We also found out that custom cuts do not improve the speed of the instance evaluation, as it was slower than the version without custom cuts. When solving DARP instances using a VGA algorithm which contains SVDARP as a subproblem, we found out that our implementation might be faster for datasets that have 20 or more requests.

The goals of this work can be considered as fulfilled. We managed to find and implement an efficient algorithm for the SVDARP. With help of our implemented algorithm, we can effectively solve larger instances where the algorithm based on backtracking was slow.

We see a possible extension of this work in an implementation of the exact algorithm B&P&C according to [12]. The algorithm in the mentioned work is the

fastest exact algorithm for DARP introduced so far. To implement this algorithm, our work can be extended by the B&P part.

Bibliography

[1] Susan Shaheen and Adam Cohen. Mobility on Demand (MOD) and Mobility as a Service (MaaS) How Are They Similar and Different?, March 2019.

[2] Sin C. Ho, W.Y. Szeto, Yong-Hong Kuo, Janny M.Y. Leung, Matthew Petering, and Terence W.H. Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, May 2018.

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

[4] Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[5] Karl Doerner and Juan José Salazar González. Chapter 7: Pickup-and-Delivery Problems for People Transportation. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 193–212. SIAM - Society for Industrial and Applied Mathematics, 2nd revised ed. edition edition, 2014.

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

[7] L. G. Mitten. Branch-and-Bound Methods: General Formulation and Properties. *Operations Research*, 18(1):24–34, 1970. Publisher: INFORMS.

[8] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[9] John E. Mitchell. BRANCH-AND-CUT ALGORITHMS FOR INTEGER PROGRAMMING, August 2001.

[10] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007. Publisher: John Wiley & Sons, Ltd.

[11] Sophie N. Parragh, Jorge Pinho de Sousa, and Bernardo Almada-Lobo. The Dial-a-Ride Problem with Split Requests and Profits. *Transportation Science*, 49(2):311–334, 2014. Publisher: INFORMS.

[12] Timo Gschwind and Stefan Irnich. Effective Handling of Dynamic Time Windows and Its Application to Solving the Dial-a-Ride Problem. *Transportation Science*, 49(2):335–354, September 2014. Publisher: INFORMS.

[13] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[14] Jang-Jei Jaw, Amedeo R. Odoni, Harilaos N. Psaraftis, and Nigel H.M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[15] Ying Luo and Paul Schonfeld. A rejected-reinsertion heuristic for the static Dial-A-Ride Problem. *Transportation Research Part B: Methodological*, 41(7):736–755, August 2007.

[16] Lauri Häme. An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows. *European Journal of Operational Research*, 209(1):11–22, 2011.

[17] Michel Gendreau. An Introduction to Tabu Search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 37–54. Springer US, Boston, MA, 2003.

[18] Pierre Hansen and Nenad Mladenović. Variable Neighborhood Search. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 313–337. Springer US, Boston, MA, 2014.

[19] Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6):1129–1138, 2010.

[20] David Pisinger and Stefan Ropke. Large Neighborhood Search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 399–419. Springer US, Boston, MA, 2010.

[21] Timo Gschwind and Michael Drexl. Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem. *Transportation Science*, 53(2):480–491, January 2019. Publisher: INFORMS.

[22] Wonjae Lee and Hak-Young Kim. Genetic algorithm implementation in Python. In *Fourth Annual ACIS International Conference on Computer and*

Information Science (ICIS'05), pages 8–11, July 2005. Journal Abbreviation: Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05).

[23] Claudio Cubillos, Enrique Urra, and Nibaldo Rodríguez. Application of Genetic Algorithms for the DARPTW Problem. *INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL; Vol 4 No 2 (2009): International Journal of Computers Communications & Control (June)*, 2009.

[24] R M Jorgensen, J Larsen, and K B Bergvinsdottir. Solving the Dial-a-Ride problem using genetic algorithms. *Journal of the Operational Research Society*, 58(10):1321–1331, 2007. Publisher: Taylor & Francis.

[25] Yvan Dumas, Jacques Desrosiers, and François Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, September 1991.

[26] Michal Čáp and Javier Alonso Mora. Multi-objective analysis of ridesharing in automated mobility-on-demand. *Proceedings of RSS 2018: Robotics-Science and Systems XIV*, 2018.