CIÊNCIA DA COMPUTAÇÃO

UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Heuristic Approaches to the Dial-a-Ride Problem

Diego Paiva e Silva

JUIZ DE FORA

MARÇO, 2021

Heuristic Approaches to the Dial-a-Ride Problem

Diego Paiva e Silva

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Luciana Brugiolo Gonçalves

Coorientador: Stênio Sã Rosário Furtado Soares

JUIZ DE FORA

MARÇO, 2021

Heuristic Approaches to the Dial-a-Ride Problem

Diego Paiva e Silva

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

JUIZ DE FORA
11 DE MARÇO, 2021

Resumo

Dial-a-ride (DAR) é um modelo de serviço de transporte que consiste no compartilhamento de viagens por um conjunto de usuários que especificam suas origens, destinos e horários em que desejam ser atendidos. Tal serviço é realizado por uma frota de veículos que partem inicialmente de uma garagem de origem e, ao final do expediente, retornam a uma garagem de destino. O Problema *Dial-a-Ride* (DARP), que é $\mathcal{NP}$-difícil, consiste em obter um conjunto de rotas de custo mínimo que satisfaçam restrições operacionais. Este trabalho propõe, para o DARP, duas abordagens heurísticas diferentes (GRASP e ILS) que realizam a distribuição dos usuários e a programação dos veículos, visando minimizar uma função objetivo que consiste na distância total percorrida por todos os veículos. Experimentos computacionais foram realizados sobre um conjunto de instâncias da literatura, afim de avaliar a qualidade das abordagens propostas em termos de qualidade da solução e de tempo de computação.

Palavras-chave: Dial-a-Ride Problem, GRASP, ILS, Metaheurística.

Abstract

Dial-a-ride (DAR) is a transportation service model that consists of shared-ride trips by a set of users that specify their origins, destinations and desired times to be served. Such service is performed by a fleet of vehicles that initially depart from an origin depot and, at the end of the day, return to a destination depot. The *Dial-a-Ride Problem* (DARP), which is $\mathcal{NP}$-hard, consists of stipulating a set of minimum routes that satisfies operational constraints. This work proposes for the DARP two different heuristic approaches (GRASP and ILS) that perform users distribution and vehicle scheduling, aiming to minimize an objective function that consists on the total distance traveled by all vehicles. Computational experiments are performed on a set of instances from the literature in order to evaluate the quality of the proposed approaches in terms of solution quality and computing time.

Keywords: Dial-a-Ride Problem, GRASP, ILS, Metaheuristic.

Contents

Lista de Abreviações

ALNS	Adaptive large neighborhood search
CL	Candidate list
CS	Clustering search
DA	Deterministic annealing
DARP	Dial-a-ride problem
ELS	Evolutionary local search
GA	Genetic algorithm
GRASP	Greedy randomized adaptive search procedure
ILS	Iterated local search
RCL	Restricted candidate list
TS	Tabu search
VND	Variable neighborhood descent
VNS	Variable neighborhood search

1 Introduction

With the ever growing urbanization process, the advent of efficient public transportation systems becomes increasingly necessary. In this context, the *dial-a-ride* problem (DARP) arises in many different types of applications. It can be seen, e.g., in the transportation of elderly and disabled who do not have ease in using regular public systems, in the transportation of patients between hospitals, and in the transportation of rural residents who need to move frequently to urban centers for work/health reasons. As it is a service concerned with the transportation of people and not goods, there are some quality of service criteria (PAQUETTE; CORDEAU; LAPORTE, 2009) that must be taken into account when formulating models and algorithms.

In short, the DARP consists in designing a set of minimal routes to meet the transportation demand of a set of users. Every user submits a request, where it is specified their pickup and delivery location within a time window, which tells the minimum and maximum hours that service is allowed to begin at each location. Hence, the terms "user" and "request" are used interchangeably. The vehicle(s), departing from an origin depot, must serve all users while respecting time and capacity constraints, and must return to the destination depot by the end of the planning horizon (CORDEAU; LAPORTE, 2003). Figure 1.1 shows a graphical representation of the DARP.

Following (HO et al., 2018) taxonomy, the DARP can be classified according to two aspects: **(i)** the manner of making decisions (static or dynamic) and **(ii)** the knowledge of information (deterministic or stochastic). In the **static** case all requests are known a priori, while in the **dynamic** case the service operator is allowed to modify existing route plans in response to updated information concerning users or to unexpected disturbances such as delays and/or vehicle breakdowns. Regarding item **(ii)**, the DARP is said **deterministic** when there is certainty about the information available at the time of decision making, and is said **stochastic** otherwise. For instance, in a deterministic DARP, one would know a priori the time demanded to embark a disabled passenger, while in the stochastic case such information would be unknown or would have a probability value

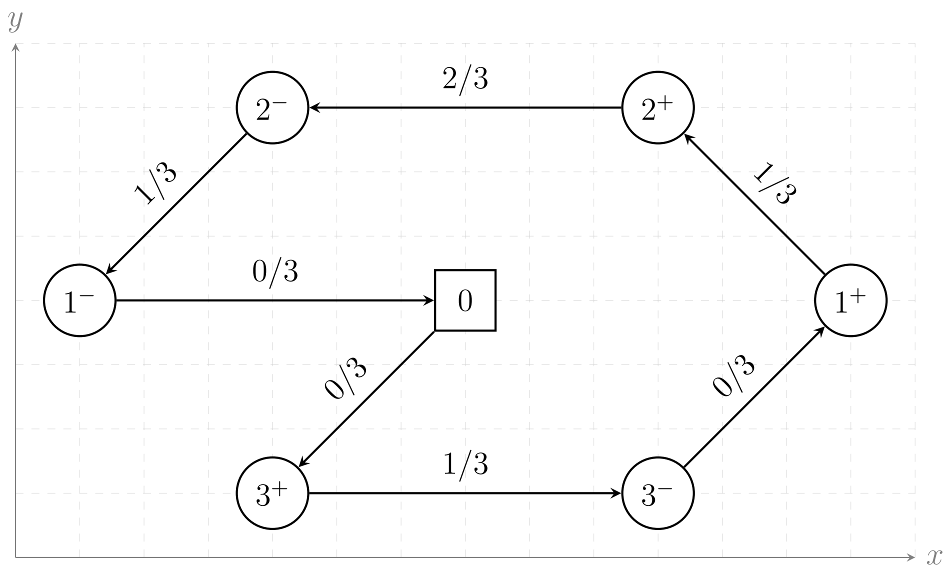


Figure 1.1: The vertex 0 denotes the origin and destination depot, while vertices u^+ and u^- denote the pickup and delivery locations of user u, respectively. The label on each edge represents the vehicle's occupancy rate, that is, the current load divided by the vehicle's capacity.

associated with.

In general, due to the difficulty of stipulating feasible routing and schedules to serve all users, and aiming at an improvement in the quality of *dial-a-ride* services, computerized planning and scheduling are higly necessary for systems of large demand. It was proven by (JR; KAKIVAYA; STONE, 1998) that the DARP is a $\mathcal{NP}$-hard problem, making it especially challenging of being solved for larger instances. Thus, the main objective of the work is to propose two heuristic approaches based on the GRASP and ILS metaheuristics, to the static-deterministic DARP. These techniques have been showing great success for combinatorial optimization problems in general, and the opportunity to extend them to the problem at hand makes it possible to measure their effectiveness and efficiency.

The rest of this work is organized as follows: Chapter 2 provides a review on the models and algorithms proposed for the DARP, as well as recent advances and contributions; Chapter 3 addresses the templates of the algorithms that will be used to solve the DARP; Chapter 4 introduces the formal definition and the mathematical model for the DARP; Chapter 5 presents the development of the proposed methodology; Chapter 6 addresses the computational experiments, as well as the details of the instances used and the execution environment. The conclusions of the study are discussed in Chapter 7.

2 Literature review

Dial-a-ride services are application oriented, that is, different types of applications may have different constraints and optimization goals. Thus, no standardized problem definition exists (PARRAGH; DOERNER; HARTL, 2010). However, (CORDEAU; LAPORTE, 2003) introduced a rather general problem description, taking in account time windows, maximum ride times and maximum route duration, while aiming to minimize the total routing costs. They also provided a set of benchmark instances that is widely adopted in the literature. Before that happened, different models and algorithms had been proposed for problems involving *dial-a-ride* services. In fact, the first occurrence refers to (WILSON et al., 1971). Notwithstanding, (PSARAFTIS, 1980; PSARAFTIS, 1983) analyzed and solved the single-vehicle DARP using dynamic programming, where the objective was to minimize the time needed to service all users, while (JAW et al., 1986) proposed one of the first heuristic approaches to the multi-vehicle DARP.

Anyway, it is possible to observe in the literature that there is much interest in focusing on the development of efficient and effective heuristic techniques rather than exact methods, due to the $\mathcal{NP}$-hardness character of the DARP (HO et al., 2018). Let us give some examples. (CORDEAU; LAPORTE, 2003) proposed a *Tabu Search* (TS) algorithm that was able to solve a set of 20 artificially generated instances ranging from 24 to 144 requests, along with a set of 6 real-life instances consisting of 200 or 295 requests and 15 or 20 vehicles, provided by a danish transporter. (JORGENSEN; LARSEN; BERGVINSDOTTIR, 2007) developed a *Genetic Algorithm* (GA) based in a cluster-first, route-second approach, for a slightly modified version of the DARP that consisted in a multi-criteria objective function which aimed to minimize the total routing costs and users inconvenience. (PARRAGH; DOERNER; HARTL, 2010) introduced a *Variable Neighborhood Search* (VNS) heuristic that applied a total of thirteen neighborhood structures, and was tested against the previously mentioned algorithms. Although they did not overcome the results of (CORDEAU; LAPORTE, 2003) in terms of average values, 16 new best solutions were identified, and they improved the results of the GA by 71.79%, on average.

In general, the DARP has been gaining increasing attention from researchers due to its broad range of variations. For instance, the heterogeneous *dial-a-ride* problem (HDARP) was introduced by (PARRAGH, 2011) after observations made at the Austrian Red Cross (ARC) in the field of patient and disabled people transportation. In this scenario, passengers may have a debilitating condition that requires the presence of some type of specific resource in the vehicle, such as stretchers and wheelchairs. As a result, each instance has a heterogeneous configuration of passengers and vehicles. The author proposes an exact *branch-and-cut* method and a heuristic technique based in the VNS algorithm. Moreover, (BRAEKERS; CARIS; JANSSENS, 2014) incorporated multiple depots (MD-H-DARP) and presented both a three-index and two-index formulation for the problem, while designing a *branch-and-cut* algorithm to solve small-sized instances and a *Deterministic Annealing* (DA) algorithm that outperformed the state-of-the-art method for both the DARP and HDARP. To the MD-H-DARP, (MALHEIROS et al., 2021) came up with a *math-heuristic*, combining the *Iterated Local Search* (ILS) metaheuristic with an exact technique based on a set partitioning approach. In addition to improving several best-known solutions and outperforming the DA algorithm, the authors provided 24 new, challenging instances ranging from 72 to 192 requests, and from 9 to 16 vehicles. A more recent research field that has been arising consists in the development of models and algorithms for the DARP that consider a fleet of electric vehicles, within the context of green logistics (PIMENTA et al., 2017; MASMOUDI et al., 2018; BONGIOVANNI; KASPI; GEROLIMINIS, 2019).

Returning to the more standardized DARP, (MASMOUDI et al., 2017) developed a hybrid GA that proved to be highly competitive, with an average deviation of 0.47% from the best-known solution on the average results, against 0.85% from the DA and 0.97% from the *Evolutionary Local Search* (ELS) of (CHASSAING; DUHAMEL; LACOMME, 2016). This represents the current state-of-the-art along with the *Adaptive Large Neighborhood Search* (ALNS) of (GSCHWIND; DREXL, 2019). The latter applied a strategy similar to the one described by (ROPKE; PISINGER, 2006) for the Pickup and Delivery Problem with Time Windows (PDPTW), but supplemented with additional removal procedures. Additionally, the authors designed a constant-time feasibility check for the DARP, but that requires a preprocessing step that runs in $O(r^3)$, where r is the number of stops in the route. On the 20 instances tested, the ALNS provided 8 new best-known solutions, and confirmed the other remaining 12, with an average deviation of 0.55% on the average results.

Finally, it is worth mentioning that study cases of real world DARPs are also a relevant topic in the literature. For example, (TOTH; VIGO, 1997) developed a computational approach to the shared transportation service in Bologna, Italy, which had far superior results than the previously handmade approach; (MUELAS; LATORRE; PEÑA, 2013) solved real instances provided by the San Francisco Municipal Transportation Agency, in the United States, through the application of the VNS metaheuristic, and (RODRIGUES et al., 2014) solved instances for the shared transportation service in Vitória, Brazil, using the *Clustering Search* (CS) metaheuristic.

For a complete review on models and algorithms for the DARP and its variants, the reader is referred to (HO et al., 2018) and (CORDEAU; LAPORTE, 2007).

3 Theoretical basis

This Chapter addresses the general characteristics of the algorithms that will be used to solve the DARP. Their application and the incorporation of problem-specific issues are discussed in Chapter 5.

3.1 Metaheuristic

Optimizing is the act of making the best or most effective use of a resource. This scenario occurs relatively often in many different real world applications, whether it is related to maximizing profit in a business model or minimizing routing costs in a network.

While exact algorithms do not usually supply a solution for that class of problems in a reasonable amount of time, one can think of developing different approaches for providing a relatively good and acceptable solution to the problem at hand. This led to the development of the so called metaheuristics, which (TALBI, 2009) defined as "upper level general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems."

But what is a heuristic? According to (FEIGENBAUM; FELDMAN, 1963) "a heuristic (heuristic rule, heuristic method) is a rule of thumb, strategy, trick, simplification, or any other kind of device which drastically limits search for solutions in large problem spaces. Heuristics do not guarantee optimal solutions; in fact, they do not guarantee any solution at all; all that can be said for a useful heuristic is that it offers solutions which are good enough most of the time."

Hence, metaheuristics can be seen as frameworks that allow the design of algorithms for optimization problems that are computationally intractable in practice. Although metaheuristics do not guarantee an optimal solution, they usually present solutions that are satisfactory in practice, and their performance outweighs the possibility of not obtaining a proven optimal solution. For an extensive review on metaheuristics, the reader is referred to (TALBI, 2009) and (GLOVER; KOCHENBERGER, 2006).

3.2 GRASP

The *Greedy Randomized Adaptive Search Procedure* (GRASP) metaheuristic (FEO; RESENDE, 1989) consists in a stochastic and iterative process in which each iteration if made up of two phases: **(i) construction phase**, where an initial solution is generated in a greedy-randomized fashion and **(ii) local search phase**, where the initial solution is refined through a local search procedure in order to establish a local optimum. The best overall solution is kept as the final result. The pseudocode for GRASP is shown in Algorithm 1.

Algorithm 1: GRASP

```
Input: Stopping criterion $c$ and randomness parameter $0 \leq \alpha \leq 1$.
Output: The best solution found for the optimization problem.
1  begin
2      $s^* \leftarrow \emptyset$, $f^* \leftarrow +\infty$;
3      while not $c$ do
4          $s_0 \leftarrow ConstructGreedyRandomizedSolution(\alpha)$;
5          $s' \leftarrow LocalSearch(s_0)$;
6          if $f(s') < f^*$ then
7              $s^* \leftarrow s'$;
8              $f^* \leftarrow f(s')$;
9          end if
10     end while
11     return $s^*$;
12 end
```

Besides requiring little programming effort to implement, GRASP has only a few parameters to be set and tuned, which makes it simple yet efficient. The stopping criterion c is usually set to a maximum number of iterations, although it is possible to adopt it as, e.g., maximum number of consecutive iterations without improvement or maximum running time.

3.2.1 Construction phase

The construction phase aims to generate a feasible solution. This is done iteratively in a greedy-randomized fashion, one element at a time. Elements are ranked according to a greedy function that measures the benefit of their insertion in the solution, and placed in a *candidate list* (CL). In each iteration a *restricted candidate list* (RCL) is built based in the α parameter. Then, an element is taken from RCL at random to join the solution and removed from CL thereafter. The adaptive nature of the algorithm comes from the fact that, at the end of each iteration, the remaining elements of CL are updated according to the newly modified solution. The pseudocode for the construction phase is shown in Algorithm 2.

Algorithm 2: Constructive algorithm for the GRASP

```
Input: Randomness parameter $0 \leq \alpha \leq 1$.
Output: A solution for the optimization problem.
1  begin
2      $s \leftarrow \emptyset$;
3      Initialize candidate list CL;
4      while CL $\neq \emptyset$ do
5          RCL $\leftarrow$ $ConstructRCL(\text{CL}, \alpha)$;
6          Randomly take an element $e \in$ RCL;
7          $s \leftarrow s \cup \{e\}$;
8          CL $\leftarrow$ CL $- \{e\}$;
9          Update CL;
10     end while
11     return $s$;
12 end
```

Regarding the construction of the RCL (line 5), the two most common approaches are either based in the quality of each element or in the cardinality of the candidate list. For the former, let $g(e)$ be the value of the greedy function applied to element $e \in CL$. Then, RCL $= \{e \in CL : g^{\min} \leq g(e) \leq g^{\min} + \alpha(g^{\max} - g^{\min})\}$, where $g^{\min} = \min\{g(e) : e \in CL\}$ and $g^{\max} = \max\{g(e) : e \in CL\}$. For the latter, one can simply limit the size of RCL to $1 + \alpha(|\text{CL}| - 1)$.

It is important to note that $\alpha = 0$ yields a purely greedy procedure, as RCL will contain only the element with the highest value of greedy function, while $\alpha = 1$ results in a completely random procedure, since RCL will be equal to CL.

3.2.2 Local search phase

A local search algorithm works by continuously searching the neighborhood of current solution and replacing it whenever a better solution is found. This process is repeated until a locally optimal solution has been reached. The neighborhood of a solution s is defined as a finite set of solutions $N(s) = \{s_1, \ldots, s_n\}$ and s is said to be *locally optimal* if $f(s) < f(s')$, $\forall s' \in N(s)$. Neighborhoods are generated by arbitrary **operators**, e.g., by swapping positions of distinct elements. Algorithm 3 states a generic local search procedure for GRASP.

Algorithm 3: Generic local search procedure for GRASP

```
   Input: Solution $s$.
   Output: $s$ as local optimal.
1  begin
2      while $\exists\ s' \in N(s) : f(s') < f(s)$ do
3          Find a better solution $s' \in N(s)$;
4          $s \leftarrow s'$;
5      end while
6      return $s$;
7  end
```

When selecting a better solution (line 3), one may adopt a **first improvement** strategy, in which the first improving solution in the neighborhood is selected, or a **best improvement** strategy, where the entire neighborhood is investigated and the selected solution is the one with the best cost with respect to the objective function. It is noticeable that the main difference between the two approaches lies in the computational performance.

As described by (FEO; RESENDE, 1995), "the key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution."

3.3 ILS

The *Iterated Local Search* (ILS) metaheuristic (LOURENÇO; MARTIN; STÜTZLE, 2003) is an iterative procedure that aims to escape from locally optimal solutions by sequentially applying a change (perturbation) in the current solution s^*, leading to an intermediate solution s'. Then, a local search step is performed in s', yielding a new solution $s^{*\prime}$. If $s^{*\prime}$ is accepted by means of a predefined acceptance criterion, the search continues from the new starting point $s^{*\prime}$; otherwise, one returns to s^*.

As depicted by the authors, the perturbation mechanism should not be neither too small nor too large. In case a small change is performed, s' may quickly degenerate to s^* during the local search step. On the other hand, if the perturbation is too large, s' will be random and the characteristics of s^* will be lost, resulting in a random restart type algorithm. A graphical representation of the ILS procedure is shown in Figure 3.1

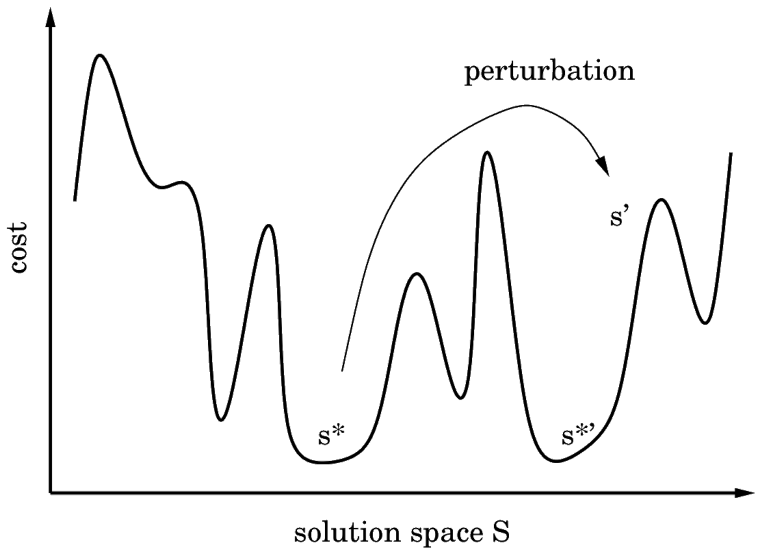


Figure 3.1: The ILS procedure (LOURENÇO; MARTIN; STÜTZLE, 2003).

The algorithmic structure of the ILS is presented in Algorithm 4. First, an initial solution is generated and a local optimum is established (lines 2-3). Then, while the stopping criterion c is not met (lines 5-7), solution s^* is perturbed and a local search procedure is applied in the hope of finding promising neighboring solutions. The *AcceptanceCriterion* procedure determines a new starting point for the search, based in the quality of the new obtained solution $s^{*\prime}$. At the end of the algorithm, s^* is returned as the final solution to the problem.

Algorithm 4: ILS

```
   Input: Stopping criterion $c$.
   Output: A solution to the optimization problem.
1  begin
2      $s_0 \leftarrow GenerateInitialSolution()$;
3      $s^* \leftarrow LocalSearch(s_0)$;
4      while not $c$ do
5          $s' \leftarrow Perturb(s^*)$;
6          $s^{*\prime} \leftarrow LocalSearch(s')$;
7          $s^* \leftarrow AcceptanceCriterion(s^*, s^{*\prime})$;
8      end while
9      return $s^*$;
10 end
```

In spite of being simple and easy to implement, the effectiveness of ILS depends mainly on the choice of the local search, the perturbation mechanism and the acceptance criterion.

3.4 VND

Variable Neighborhood Descent (VND) (MLADENOVIĆ; HANSEN, 1997) is a search heuristic used to diversify the exploration of neighborhoods for a solution. The procedure begins by defining a list of k neighborhood structures $NL = \{N_1, \ldots, N_k\}$ to be used during the search. Then, the general ideia of VND is to find a better solution in the first neighborhood. If such solution was found, the search is restarted using the first neighborhood; otherwise the search is performed in the next neighborhood (in this case N_2, if $k \geq 2$). The procedure ends when the k-th neighborhood is unable to improve the current solution, which will be a local optimum with respect to all other neighborhood structures. The order of the neighborhoods can be imposed by means of predefined criteria, e.g., through an increasing order of cardinality. The pseudocode for the VND is shown in Algorithm 5.

Algorithm 5: VND

```
   Input: Solution $s$.
   Output: $s$ as local optimal.
1  begin
2      $NL \leftarrow \{N_1, \ldots, N_k\}$;
3      $i \leftarrow 1$;
4      while $i \leqslant k$ do
5          Find a better solution $s'$ in $N_i \in NL$;
6          if $f(s') < f(s)$ then
7              $s \leftarrow s'$;
8              $i \leftarrow 1$;
9          end if
10         else
11             $i \leftarrow i + 1$;
12         end if
13     end while
14     return $s$;
15 end
```

In line 2, the list NL containing the k neighborhood structures is initialized. Variable i keeps track of the current neighborhood being used. As long as i points to a valid neighborhood, the search is performed (line 5). As discussed in Section 3.2.2, this step may follow a first improvement or a best improvement strategy. The resulting solution is evaluated and in case it is better than the current solution, s is updated accordingly and i is set to the first neighborhood (lines 6-8); otherwise the search moves to the next neighborhood (line 11). At the end of the procedure, s is returned as a local optimal.

The ideia behind VND is to provide means of exploring neighborhoods that are distant from each other, but without losing the favorable characteristics of the incumbent solution, in the hope of finding promising neighboring solutions.

4 The dial-a-ride problem

Consider n users to be served. The DARP can be modelled in a complete directed graph $G = (V, A)$, where $V = P \cup D \cup \{0, 2n+1\}$ and where $P = \{1, \ldots, n\}$ are pickup locations and $D = \{n+1, \ldots, 2n\}$ delivery locations, while vertices 0 and $2n+1$ denote the origin and destination depots, respectively. A request submited by a user $i \in P$ is made up by the pair of vertices $(i,\ n+i)$. Let K be the fleet of vehicles responsible for servicing the users. Every vehicle $k \in K$ has a capacity Q_k and the total duration of its route must not exceed T_k. Every vertex $i \in V$ is associated to **(i)** a time window $[e_i, l_i]$, which determines the minimum and maximum time that the vertex can be visited, **(ii)** a load q_i, which is always positive for origins, negative for destinations and zero for every other vertex and **(iii)** a service time d_i, which represents the time needed for the vehicle to embark/disembark the user. Each arc $(i, j) \in A$ is associated a with travel cost c_{ij} and a travel time t_{ij} for which the triangle inequality holds. The maximum ride time for each user is L.

The DARP is also a scheduling problem, since one needs to know the schedules for each vehicle in order to check if a given solution to the problem is reasonable. To this purpose, we define a set of decision variables. The arrival time of vehicle $k \in K$ in vertex $i \in V$ can be calculated as $A_i^k = D_{i-1}^k + t_{i-1,i}$. Service at i can not begin before e_i, therefore the service beginning time is defined as $B_i^k = \max\{e_i, A_i^k\}$. The vehicle departs from vertex i once the service has been completed, which results in the departure time $D_i^k = B_i^k + d_i$. Time spent waiting by the vehicle in i is equal to $W_i^k = B_i^k - A_i^k$ and its load imediately after visiting i is calculated as $Q_i^k = Q_{i-1}^k + q_i$. The ride time of user $i \in P$ can be computed as $L_i^k = B_{n+i}^k - D_i^k$. The goal of the problem is to find a solution s that minimizes total routing costs $c(s) = \sum_{\forall (i,j) \in s} c_{ij}$ while respecting all constraints.

Figure 4.1, adapted from (BREVET et al., 2019), provides a graphical understanding of time-related decision variables, where the possibilities of arriving before and after the time window of a vertex are shown. Table 4.1 gives a summary of the notations used throughout the work.

Notation	**Description**
Sets	
$P = \{1, \dots, n\}$	Pickup vertices
$D = \{n+1, \dots, 2n\}$	Delivery vertices
$V = P \cup D \cup \{0, 2n+1\}$	All vertices
K	Vehicles
Parameters	
Q_k	Capacity of vehicle $k \in K$
T_k	Maximum route duration of vehicle $k \in K$
e_i	Beginning of time window at vertex $i \in V$
l_i	End of time window at vertex $i \in V$
q_i	Required capacity by vertex $i \in V$
d_i	Service time at vertex $i \in V$
L	Maximum user ride time
c_{ij}	Travel cost between vertices $i, j \in V$
t_{ij}	Travel time between vertices $i, j \in V$
Variables	
A_i^k	Arrival time of vehicle $k \in K$ at vertex $i \in V$
B_i^k	Service beginning time of vehicle $k \in K$ at vertex $i \in V$
D_i^k	Departure time of vehicle $k \in K$ at vertex $i \in V$
W_i^k	Waiting time of vehicle $k \in K$ at vertex $i \in V$
Q_i^k	Load of vehicle $k \in K$ at vertex $i \in V$
L_i^k	Ride time of user $i \in P$ in vehicle $k \in K$

Table 4.1: Notations.

4.1 Formulation

This Section presents the mathematical formulation for the DARP based in the definition of (CORDEAU; LAPORTE, 2003). Although the proposed approach in this work does not use the model explicitly, it is important for understanding the restrictive aspects of the problem.

In addition to the data provided in Table 4.1, let $x_{ij}^k = 1$ if vehicle k travels from vertex i to vertex j; $x_{ij}^k = 0$ otherwise. The formulation can then be constructed as follows:

$$\text{Minimize} \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^k \tag{4.1a}$$

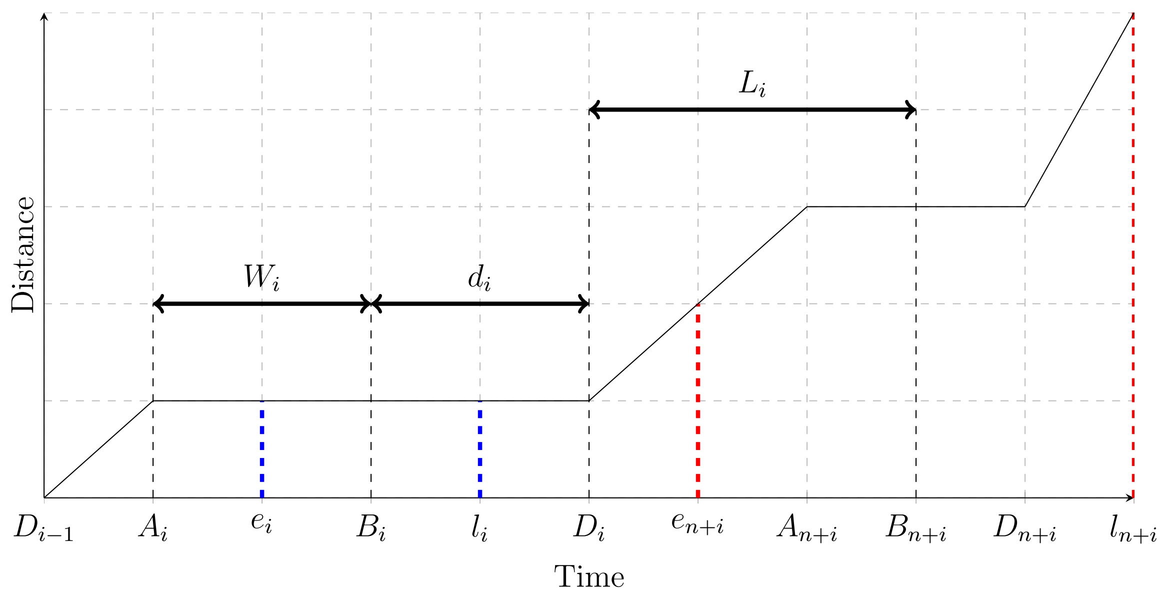


Figure 4.1: Visualization of time-related decision variables.

subject to

$$\sum_{k\in K}\sum_{j\in V} x_{ij}^k = 1 \qquad \forall i \in P \tag{4.1b}$$

$$\sum_{j\in V} x_{ij}^k - \sum_{j\in V} x_{n+i,j}^k = 0 \qquad \forall i \in P,\ k \in K \tag{4.1c}$$

$$\sum_{j\in V} x_{0j}^k = 1 \qquad \forall k \in K \tag{4.1d}$$

$$\sum_{j\in V} x_{ji}^k - \sum_{j\in V} x_{ij}^k = 0 \qquad \forall i \in P \cup D,\ k \in K \tag{4.1e}$$

$$\sum_{i\in V} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{4.1f}$$

$$B_j^k \geqslant (B_i^k + d_i + t_{ij})x_{ij}^k \qquad \forall i,j \in V,\ k \in K \tag{4.1g}$$

$$Q_j^k \geqslant (Q_i^k + q_j)x_{ij}^k \qquad \forall i,j \in V,\ k \in K \tag{4.1h}$$

$$L_i^k = B_{n+i}^k - (B_i^k + d_i) \qquad \forall i \in P,\ k \in K \tag{4.1i}$$

$$B_{2n+1}^k - B_0^k \leqslant T_k \qquad \forall k \in K \tag{4.1j}$$

$$e_i \leqslant B_i^k \leqslant l_i \qquad \forall i \in V,\ k \in K \tag{4.1k}$$

$$t_{i,n+i} \leqslant L_i^k \leqslant L \qquad \forall i \in P,\ k \in K \tag{4.1l}$$

$$\max\{0, q_i\} \leqslant Q_i^k \leqslant \min\{Q_k, Q_k + q_i\} \qquad \forall i \in V,\ k \in K \tag{4.1m}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall i,j \in V,\ k \in K \tag{4.1n}$$

The objective of the problem defined in (4.1a) is to minimize the total routing costs. Equations (4.1b) and (4.1c) guarantee that each user will be served only once and by the same vehicle. Constraints (4.1d), (4.1e) and (4.1f) ensure that each vehicle departs from the origin depot and returns to the destination depot. Consistency of time and load variables are imposed by constraints (4.1g) and (4.1h), respectively. The ride time for each user is determined by equation (4.1i) and bounded by equation (4.1l). Each user must be served within their time window, which is ensured by constraint (4.1k). Finally, constraint (4.1j) bounds the duration for each route while constraint (4.1m) ensures that vehicles will not have their maximum capacity violated at any time.

It is worth mentioning that as noted by (CORDEAU; LAPORTE, 2003), the time window at vertex i is only violated if and only if $B_i^k > l_i$ due to the fact that the arrival before e_i is allowed, which implies a waiting time $W_i^k > 0$.

5 Development

This chapter describes the solution methodology for the two proposed approaches, in addition to preprocessing routines and feasibility checking techniques.

5.1 Solution evaluation

Every solution generated by the algorithms must be evaluated in terms of its cost $c(s)$ and its feasibility. Based on the work of (CORDEAU; LAPORTE, 2003), it is stipulated an evaluation function $f(s)$ that allows to perform such analysis. In this function, besides cost $c(s)$, the following violations are taken in account: load $q(s)$, route duration $d(s)$, time window $w(s)$ and ride time $t(s)$. Each violation can be computed as follows:

$$q(s) = \sum_{i \in V} (Q_i^k - Q_k)^+ \tag{5.1}$$

$$d(s) = \sum_{i \in V} (B_{2n+1}^k - B_0^k - T_k)^+ \tag{5.2}$$

$$w(s) = \sum_{i \in V} (B_i^k - l_i)^+ \tag{5.3}$$

$$t(s) = \sum_{i \in P} (L_i^k - L)^+ \tag{5.4}$$

where $x^+ = \max\{0, x\}$ and $\forall\ k \in K$. The evaluation function $f(s)$ can then be defined as:

$$f(s) = c(s) + \mu q(s) + \beta d(s) + \gamma w(s) + \tau t(s) \tag{5.5}$$

Terms μ, β, γ and τ are penalty parameters for the constraints which they are associated. They are set to $\mu = \beta = \gamma = \tau = 1$. Only feasible solutions are selected during the course of the proposed algorithms. Let s^* be the best solution found so far. The necessary condition for a given solution s to become the new best solution is $f(s) < f(s^*)\ \wedge\ q(s) = d(s) = w(s) = t(s) = 0$.

In order to optimize route duration in contrast to time contraints, we introduce

The Eight Step Evaluation Scheme developed by (CORDEAU; LAPORTE, 2003). This scheme uses the concept of **forward time slack** proposed by (SAVELSBERGH, 1992) to the VRP and is adapted to the DARP. The forward time slack F_i for a given vertex i in the route gives the maximum amount of time that the departure from i can be delayed without causing violations in time window and ride time constraints, and is calculated as:

$$F_i = \min_{i \leqslant j \leqslant q} \left\{ \sum_{i<p\leqslant j} W_p^k + \left(\min\{l_j - B_j^k, L - P_j\}\right)^+ \right\} \tag{5.6}$$

where q is the last vertex in the route and P_j is the ride time of the user whose destination is $j \in D$ given that $j - n$ is visited before i on the route; $P_j = 0$ for all other j. Slack at vertex j is given by the cumulative waiting time up to j plus the minimum of the difference between the end of the time window and the beginning of service at j, and the difference between the maximum user ride time and P_j. Hence, the forward time slack at vertex i is the minimum of all slacks between i and the last vertex of the route.

Algorithm 6: The Eight Step Evaluation Scheme

```
1. $D_0^k \leftarrow e_0$;
2. Compute $A_i^k, B_i^k, D_i^k, W_i^k$ and $Q_i^k$ for each vertex $i$ in the route;
   If some $B_i^k > l_i^k$ or $Q_i^k > Q^k$, go to step 8.
3. Compute $F_0$;
4. $D_0^k \leftarrow e_0 + \min\left\{F_0, \sum_{0<p<q} W_p^k\right\}$;
5. Update $A_i^k, B_i^k, D_i^k$ and $W_i^k$ for each vertex $i$ in the route;
6. Compute $L_i^k$ for each request $i \in P$ in the route;
7. For each vertex $j \in P$ in the route
    (a) Compute $F_j$;
    (b) $W_j^k \leftarrow W_j^k + \min\left\{F_j, \sum_{j<p<q} W_p^k\right\}$; $B_j^k \leftarrow A_j^k + W_j^k$; $D_j^k \leftarrow B_j^k + d_j$;
    (c) Update $A_i^k, B_i^k, D_i^k$ and $W_i^k$ for each vertex $i$ that comes after $j$ in
        the route;
    (d) Update $L_i^k$ for each request $i \in P$ whose destination is after $j$;
8. Compute changes in violations of vehicle load, duration, time window
   and ride time constraints;
```

The Eight Step Evaluation Scheme is presented in Algorithm 6. First, time window violations are minimized in steps (1) and (2). In case violations regarding time window or load constraints are found, then an irreparable violation has been spotted and therefore the current solution will be infeasible. In steps (3) - (6) route duration is minimized without increasing time window violations, and in step (7) the ride time of each user is minimized by delaying the beginning of service in the origin vertex without increasing any other violation.

5.2 Preprocessing

To avoid unnecessary computations in the future and to improve performance, some preprocessing routines are applied in the input instance before running the proposed algorithm.

5.2.1 Time window tightening

According to (CORDEAU, 2006), time windows for each user can be tightened considering the planning horizon H (the amount of time in the future that is taken in account when preparing a strategic plan) as follows. In case of an outbound user, that is, a user where time window is imposed in the destination vertex, the time window at the origin vertex can be defined as $e_i = \max\{0, e_{i+n} - L - d_i\}$ e $l_i = \min\{l_{i+n} - t_{i,i+n} - d_i, H\}$. In case of an inbound user where time window is imposed in the origin vertex, time window tightening can be applied to the destination vertex by setting $e_{i+n} = \max\{0, e_i + d_i + t_{i,i+n}\}$ and $l_{i+n} = \min\{l_i + d_i + L, H\}$.

5.2.2 Arc initialization

The travel cost c_{ij} between two vertices $i, j \in V$ is computed as the euclidian distance between i and j and stored in a matrix to allow efficient access in $O(1)$. Based on DARP works and as a matter of abstraction we let $t_{ij} = c_{ij}$. Assuming that time is given in minutes and distance in kilometers, this assumption holds since the average travel speed between two vertices is equal to $\frac{c_{ij}}{t_{ij}}$ and is equivalent of considering that every vehicle travels with a constant speed of 1 km min^{-1} or 60 km h^{-1}.

5.3 Construction of initial solution

Solution s_0 used in both GRASP and ILS is initialized with $|K|$ routes (one route per available vehicle) containing the origin and destination depots. The *candidate list* (CL) is constructed in two steps. First, for each user $i \in P$, we compute the cheapest feasible insertion of i in every route of s_0. Then, the route that caused the least increase in the total routing costs is selected to join CL. In case a user can not be feasibly inserted in any of the previous routes, the user is associated with a dummy route of cost $+\infty$. The cheapest insertion of a user in a given route can be determined by evaluating every possible combination of the vertices i and $n + i$ and costs $O(\eta^2)$, where η is the length of the route being analyzed. Feasibility check is done by invoking Algorithm 6.

After sorting CL according to route costs, the *restricted candidate list* (RCL) is formed by the first α percent elements of CL. An element consisting of a user and its best route is randomly selected from RCL to join the current solution. In case the selected user has insertion cost equal to $+\infty$, a new vehicle is activated to accommodate them, thus s_0 will be infeasible. Finally, CL is updated by removing the current selected element and recomputing the best insertion route for every other user. This process is repeated until all users have been assigned.

5.4 Repair step

Solutions generated in construction phase might be infeasible with respect to one or more constraints. If that is the case, the resulting solution will have more vehicles than allowed and will not be eligible to local search. To avoid generation of infeasible solutions as much as possible, a repair step is provided. Let x be the number of extra vehicles in the solution. First, routes are sorted by decreasing size of distance since long routes often contain remote users that cause high routing costs as depicted by (MASMOUDI et al., 2017). Then, the top x routes are removed from solution and all users assigned to those routes are inserted in random order in their best routes in the set of remaining routes. The repair step will be sucessful if all users can be feasibly assigned to a vehicle; otherwise solution will remain infeasible and ineligible to local search phase.

5.5 VND-based local search

Local search operators in routing problems are classified either as intra-route or inter-route. An intra-route operator performs exchanges between users belonging to the same route, while in an inter-route operator users from different routes may be exchanged. In this work a single intra-route operator (relocate) and a pair of inter-route operators (shift 1-0 and 2-opt*) were adopted. In the hope of finding promising solutions, a best improvement strategy is used and neighborhoods are explored using VND metaheuristic (Algorithm 5).

5.5.1 Inter-route: 2-opt* (N_1)

This operator was introduced by (POTVIN; ROUSSEAU, 1995) after they realized that operator k-opt (LIN; KERNIGHAN, 1973) is not very suitable to problems with time windows because most of the exchanges do not preserve route orientation, thus yelding a high number of infeasible solutions. The 2-opt* works as follows: two routes p and q are selected and each have an arbitrary arc removed. The remaining segment of route p is appended to the end of route q and vice versa, yelding two new routes p' and q'. For the DARP only arcs where the vehicle is empty are eligible for removal, since origin and destination vertices of a given user must belong to the same route. This procedure if performed for every pair of elegible arcs and the selected solution will contain the best exchange found.

5.5.2 Intra-route: Relocate (N_2)

A user is removed from their route r and reinserted in their best position in r yielding a new route r'. This procedure is performed sequentially for all users and the solution consisting in the best reinsertion for each route is selected.

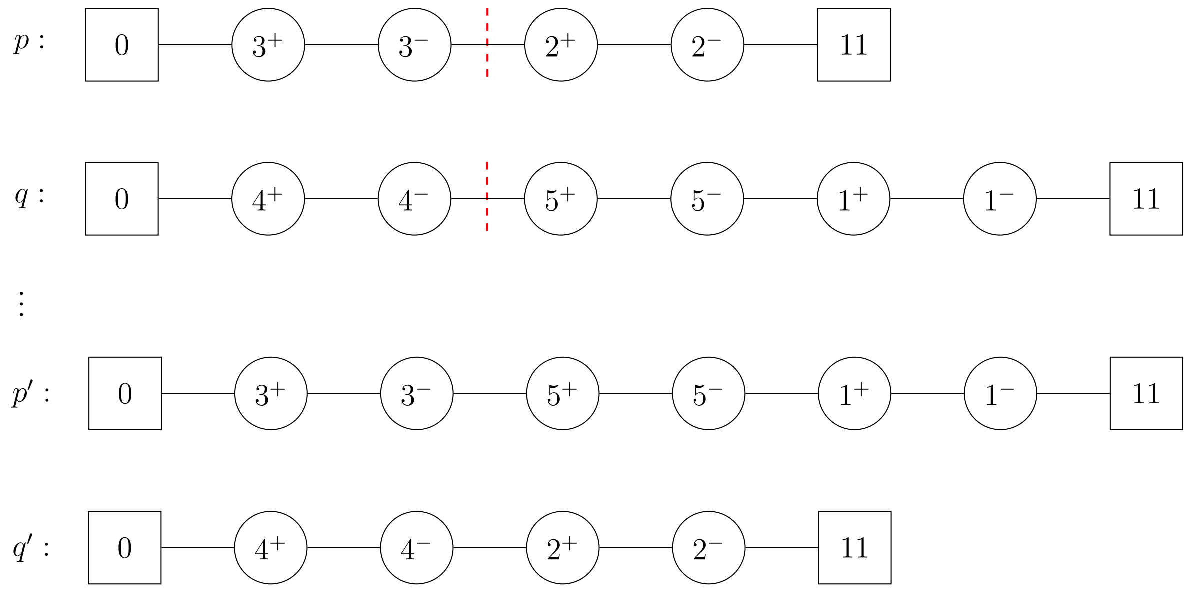


Figure 5.1: 2-opt* operator for routes p and q and the resulting routes p' and q'.

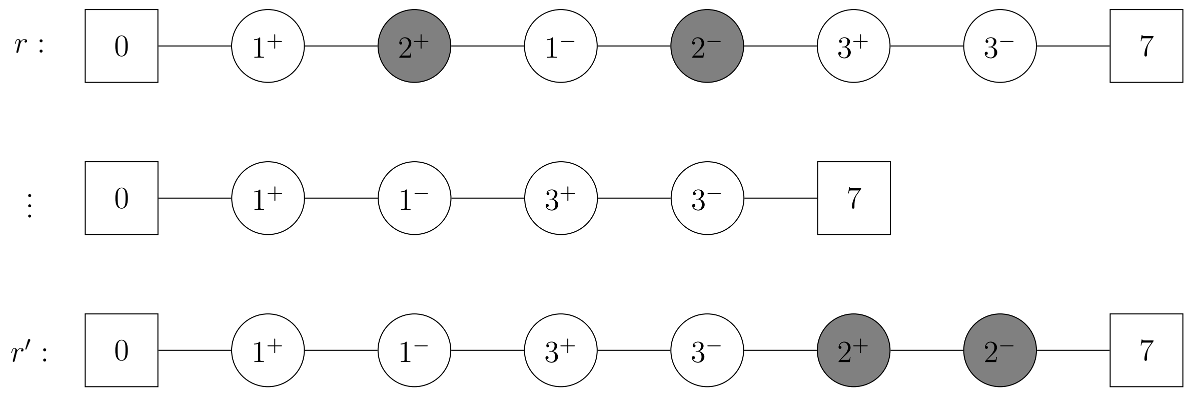


Figure 5.2: Relocate operator for user $(2^+, 2^-)$ and the resulting route r'.

5.5.3 Inter-route: Shift 1-0 (N_3)

A user is shifted from their route p and inserted in their best position in another route q, yielding two new routes p' and q'. This procedure is performed sequentially for all users and the selected solution is the one containing the pair of routes that provided the best shift operation.

5.6 GRASP applied to the DARP

The general structure of GRASP was presented in Algorithm 1. Nevertheless, some subtle changes were embedded in to suit the problem at hand. These changes are shown

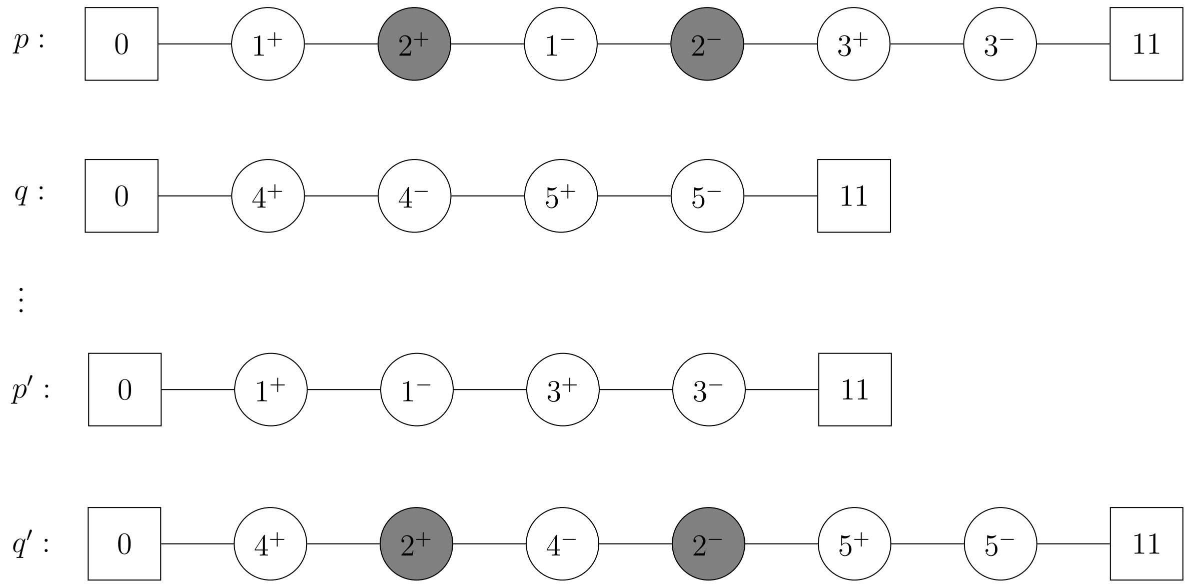


Figure 5.3: Shift 1-0 operator for user $(2^+, 2^-)$ and the resulting routes p' and q'.

in Algorithm 7.

Algorithm 7: GRASP applied to the DARP

```
Input: $n_{iter}$ and randomness parameter $0 \leq \alpha \leq 1$.
Output: The best solution found for the DARP instance.
1  begin
2      $s^* \leftarrow \emptyset$, $f^* \leftarrow +\infty$;
3      for $i \leftarrow 1$ to $n_{iter}$ do
4          $s_0 \leftarrow ConstructGreedyRandomizedSolution(\alpha)$;
5          if $s_0$ is not feasible then
6              $s_0 \leftarrow Repair(s_0)$;
7          end if
8          if $s_0$ is feasible then
9              $s' \leftarrow VND(s_0)$;
10             if $f(s') < f^*$ then
11                 $s^* \leftarrow s'$;
12                 $f^* \leftarrow f(s')$;
13             end if
14         end if
15     end for
16     return $s^*$;
17 end
```

First, the stopping criterion is set to a maximum number of iterations n_{iter} in order to equally distribute them among all threads that will be used. Second, a *repair step* (see Section 5.4) is provided in the hope of fixing infeasible solutions that might be generated by the constructive algorithm (see Section 5.3). Finally, the local search step

(VND) and the incumbent solution update are involved in a conditional statement where only feasible solutions are allowed.

5.7 ILS applied to the DARP

As done in the GRASP, the general ILS framework was slightly modified to the DARP as shown in Algorithm 8.

Algorithm 8: ILS applied to the DARP

```
Input: $n_{iter}$, $n_{imp}$ and randomness parameter $0 \leq \alpha \leq 1$.
Output: The best solution found for the DARP instance.
begin
    $s_0 \leftarrow \emptyset, f^* \leftarrow +\infty, j \leftarrow 0$;
    repeat
        $s_0 \leftarrow ConstructGreedyRandomizedSolution(\alpha)$;
        if $s_0$ is not feasible then
            $s_0 \leftarrow Repair(s_0)$;
        end if
    until $s_0$ is feasible;
    $s^* \leftarrow VND(s_0)$;
    for $i \leftarrow 1$ to $n_{iter}$ do
        $s' \leftarrow Perturb(s^*)$;
        if $s'$ is feasible then
            $s^{*\prime} \leftarrow VND(s')$;
            if $f(s^{*\prime}) < f^*$ then
                $s^* \leftarrow s^{*\prime}$;
                $f^* \leftarrow f(s^{*\prime})$;
                $j \leftarrow 0$;
            end if
            else
                $j \leftarrow j + 1$;
            end if
        end if
        else
            $j \leftarrow j + 1$;
        end if
        if $j = n_{imp}$ then
            break;
        end if
    end for
    return $s^*$;
end
```

The stopping criterion is defined as a maximum number of iterations n_{iter} or n_{imp}

iterations without improvement (whichever comes first). The reason for this choice is to seek a balance between computing time and solution quality. Due to the fact that only feasible solutions are allowed in the local search phase, the construction of initial solution is repeated until a feasible solution has been reached (lines 3-8). The acceptance criterion simply checks if the current solution is feasible and better than the incumbent solution.

5.7.1 Perturbation mechanism

In order to generate a new starting point for the local search phase, a basic perturbation mechanism is implemented. Three distinct non-empty routes $\{r_1, r_2, r_3\}$ are chosen at random and three requests $\{req_1, req_2, req_3\}$ are randomly selected, one on each route. Then, each of them are reinserted in their best positions in a cyclic fashion, that is, req_1 in r_2, req_2 in r_3 and req_3 in r_1.

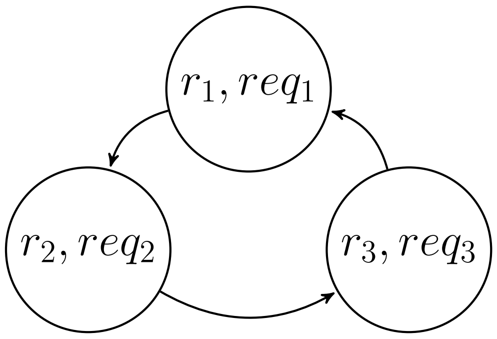


Figure 5.4: Pictorial representation of the perturbation mechanism.

6 Computational experiments

All experiments were performed in a i7-8565U laptop with 1.8GHz and 8 GB RAM. Both algorithms were implemented in C++. For the ILS only a single thread is allowed, while in the GRASP a total of 8 threads were adopted, using the OpenMP library (DAGUM; MENON, 1998) for parallel computing. In all tables, *gap* is calculated as $\frac{\overline{c(s)}-\text{BKS}}{\text{BKS}}$, where $\overline{c(s)}$ is the average cost of the obtained solutions and BKS is the cost of the best-known solution for the given instance.

6.1 Benchmark instances

The set of twenty randomly generated instances from (CORDEAU; LAPORTE, 2003) was used. The number of requests ranges from 24 to 144, where half of them are inbound and the other half are outbound requests (see Section 5.2.1). The length of the time window in instances R1a-R10a varies between 15 and 45, and between 30 and 90 in instances R1b-R10b. For each instance, all vertices are randomly generated in the square $[-10, 10]^2$ and the planning horizon is equal to $24 \times 60 = 1440$. For each vertex, the service time is set to 10, and a single type of user with a capacity requirement of 1 and maximum ride time of 90 is considered. Furthermore, a fleet of homogeneous vehicles with capacity of 6 is assumed, and their maximum route duration is set to 480. A brief summary of the benchmark instances with their respective best-known solutions to date is provided in Table 6.1.

Instance	Requests	Vehicles	BKS	Instance	Requests	Vehicles	BKS
R1a	24	3	190.02	R1b	24	3	164.46
R2a	48	5	301.34	R2b	48	5	295.66
R3a	72	7	532.00	R3b	72	7	484.83
R4a	96	9	570.25	R4b	96	9	529.33
R5a	120	11	625.64	R5b	120	11	573.56
R6a	144	13	783.78	R6b	144	13	725.22
R7a	36	4	291.71	R7b	36	4	248.21
R8a	72	6	487.84	R8b	72	6	458.73
R9a	108	8	653.94	R9b	108	8	592.33
R10a	144	10	845.47	R10b	144	10	783.81

Table 6.1: Benchmark instances.

6.2 Constructive algorithm evaluation

The randomness parameter α used in the procedure *ConstructGreedyRandomizedSolution* (see Section 5.3) used in both GRASP and ILS was set empirically. For each candidate value, the constructive algorithm was run a thousand times for each instance, and the gap from the best-known solution and the infeasibility rate was considered. Results are reported in Table 6.2.

Instance	BKS	$\alpha = 0.2$		$\alpha = 0.4$		$\alpha = 0.6$		$\alpha = 0.8$		$\alpha = 1.0$	
		Gap	Inf.	Gap	Inf.	Gap	Inf.	Gap	Inf.	Gap	Inf.
R1a	190.02	19.50	1.37	19.69	0.49	21.10	0.49	22.66	0.78	26.09	1.37
R2a	301.34	24.50	17.97	27.94	15.82	31.96	13.67	35.88	10.84	40.88	8.11
R3a	532.00	33.84	24.80	38.08	16.11	42.18	11.72	47.36	10.45	54.32	8.30
R4a	570.25	41.41	0.20	46.69	0.10	51.34	0.00	56.85	0.00	64.65	0.00
R5a	625.64	41.42	3.22	47.23	1.46	52.85	1.56	58.47	0.49	65.33	0.49
R6a	783.78	44.06	0.10	50.16	0.00	56.07	0.00	61.92	0.00	69.90	0.00
R7a	291.71	21.32	32.91	24.77	31.54	27.37	30.76	30.27	28.52	34.78	23.54
R8a	487.84	35.50	88.96	38.37	87.89	42.52	86.33	47.40	86.13	53.32	83.69
R9a	653.94	N.A	**100.0**	N.A	**100.0**	N.A	**100.0**	N.A	**100.0**	N.A	**100.0**
R10a	845.47	43.27	99.32	49.65	99.61	55.34	99.51	61.73	99.51	67.75	98.93
R1b	164.46	27.45	0.10	28.60	0.29	29.97	0.68	32.65	0.20	35.08	0.10
R2b	295.66	30.71	0.88	33.09	0.29	36.13	0.20	38.92	0.00	44.31	0.00
R3b	484.83	41.50	3.22	47.32	2.25	52.93	1.46	57.94	0.20	65.74	0.59
R4b	529.33	44.17	9.38	50.98	5.66	56.85	3.42	62.64	1.86	70.00	0.78
R5b	573.56	49.39	0.10	56.17	0.00	62.60	0.00	68.31	0.00	76.16	0.00
R6b	725.22	49.45	0.00	56.38	0.00	62.54	0.00	69.03	0.00	77.95	0.00
R7b	248.21	30.37	5.66	34.52	4.49	38.29	4.88	43.70	5.47	48.78	4.30
R8b	458.73	39.46	15.53	44.08	12.40	48.62	5.76	53.73	5.76	60.87	5.27
R9b	592.33	49.99	52.34	56.54	53.71	61.85	52.25	68.47	46.78	77.16	47.07
R10b	783.81	N.A	**100.0**	N.A	**100.0**	N.A	**100.0**	N.A	**100.0**	N.A	**100.0**
Avg.		**37.07**	**19.78**	**41.68**	**18.45**	**46.14**	**17.37**	**51.00**	**16.50**	**57.39**	**15.70**

Table 6.2: Results on five different values of α for the constructive algorithm.

Instances R9a and R10b were categorized as outliers, since no feasible solutions could be obtained, thus assigning their gaps to the value N.A (not achievable). Their values were not considered in the final average.

In any case, as the value of α increases, the quality of the obtained solutions worsens, but a smaller percentage of infeasible solutions is generated. On the one hand, it is desirable to generate high quality initial solutions in order to obtain promising final solutions, but it is also important to obtain the least possible number of infeasible solutions, especially for the GRASP, since only feasible solutions are eligible for local search. Plotting the average infeasibility rate against the average gap for each α (Figure 6.1), one can observe that the distance between the gaps of $\alpha = 0.2$ and $\alpha = 1.0$ is around 20%, while the distance between their infeasibility rates is only 4%. Due to the small difference in the infeasibility rate between all values of α when compared to the difference in the quality of their solutions, the value of $\alpha = 0.2$ was chosen for all subsequent numerical experiments.

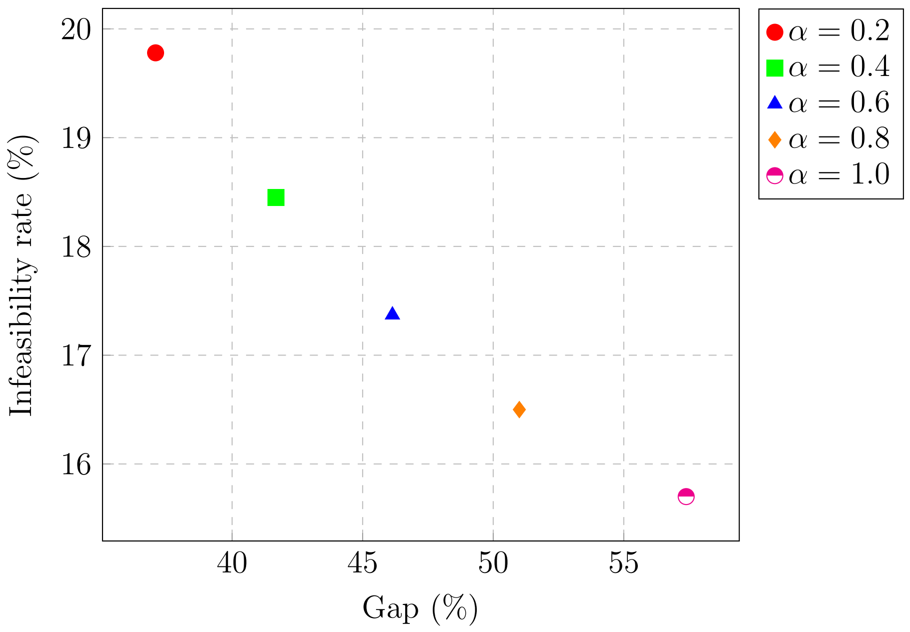


Figure 6.1: Gap versus the infeasibility rate for each candidate value of α.

It is worth mentioning that even though a constructed solution is infeasible, it might become feasible when the *repair step* is applied (see Section 5.4), potentially reducing the rate of infeasible solutions.

6.3 Results on the DARP instances

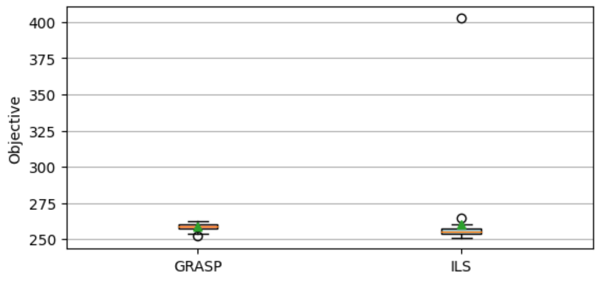
General results from 10 runs of each algorithm are illustrated in Table 6.3, where the CPU time is given in minutes. After preliminary tests, the maximum number of iterations n_{iter} was set to 2,048 in the GRASP and 20,000 in the ILS. Additionaly, the value of 2,500 was chosen as the maximum number of iterations without improvement (n_{imp}) in the ILS. The choice of such values was due to the fact that they provide a reasonable limit to the computing time, while allowing the algorithms to explore a potentially wide range of solutions. Instance R9a was run a single time with the stopping criterion being a 60 minute time limit, because it proved to be the most challenging instance for both algorithms, in terms of obtaining a feasible solution.

Instance	BKS	GRASP					ILS				
		Best	Gap^1	Avg.	Gap^2	CPU	Best	Gap^1	Avg.	Gap^2	CPU
R1a	190.02	190.02	**0.00**	190.69	**0.35**	0.26	190.02	**0.00**	195.18	2.72	0.62
R2a	301.34	303.41	0.69	306.67	**1.77**	1.97	301.34	**0.00**	314.04	4.21	4.49
R3a	532.00	556.03	4.52	560.53	**5.36**	5.78	538.72	**1.26**	573.86	7.87	5.41
R4a	570.25	620.05	8.73	626.99	9.95	16.71	584.26	**2.46**	599.91	**5.20**	10.85
R5a	625.64	683.12	9.19	699.27	11.77	31.32	655.46	**4.77**	666.36	**6.51**	13.87
R6a	783.78	878.38	12.07	889.31	13.46	54.59	817.61	**4.32**	827.59	**5.59**	17.49
R7a	291.71	295.59	1.33	297.99	**2.15**	0.68	293.39	**0.58**	339.22	16.29	2.10
R8a	487.84	526.57	7.94	537.77	**10.23**	3.46	511.06	**4.76**	593.42	21.64	1.80
R9a[a]	653.94	915.59	40.01	915.59	40.01	60.00	817.66	**25.04**	817.66	**25.04**	60.00
R10a	845.47	1032.94	22.17	1055.70	**24.87**	20.82	945.99	**11.89**	1068.43	26.37	4.16
R1b	164.46	168.35	2.37	169.16	**2.86**	0.42	167.11	**1.61**	175.67	6.82	2.87
R2b	295.66	305.10	3.19	308.01	4.18	3.10	299.70	**1.37**	303.71	**2.72**	7.61
R3b	484.83	525.24	8.33	528.92	9.09	10.47	495.73	**2.25**	501.94	**3.53**	7.94
R4b	529.33	580.57	9.68	589.96	11.45	26.01	541.22	**2.25**	560.08	**5.81**	17.42
R5b	573.56	637.21	11.10	650.34	13.39	60.92	599.88	**4.59**	607.52	**5.92**	29.43
R6b	725.22	820.85	13.19	836.36	15.33	94.74	763.51	**5.28**	778.23	**7.31**	34.00
R7b	248.21	253.51	2.14	259.42	**4.52**	1.20	251.96	**1.51**	271.78	9.50	3.05
R8b	458.73	486.35	6.02	498.46	8.66	9.89	470.94	**2.66**	479.76	**4.58**	9.47
R9b	592.33	670.05	13.12	679.97	14.80	31.27	626.03	**5.69**	677.15	**14.32**	15.39
R10b[b]	783.81	1061.98	35.49	1140.94	45.56	32.79	989.43	**26.23**	1.097.12	**39.97**	12.75
Avg.	**506.91**	**575.55**	**10.56**	**587.10**	**12.49**	**23.32**	**543.05**	**5.42**	**572.43**	**11.10**	**13.04**

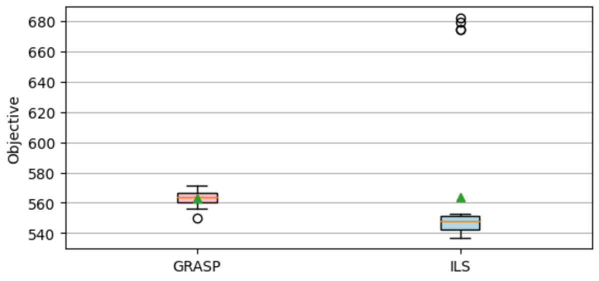
Table 6.3: Results on the DARP instances.
[a]: Single run with a time limit of 60 minutes.
[b]: Only 3 runs attempted.
[1]: Gap for the best case.
[2]: Gap for the average case.

The results showed that ILS tends to find better solutions than GRASP, but this is not always true on the average case. This situation is due to the fact that sometimes ILS generates solutions that are very distant from the remaining others, slightly distorting the final average. The boxplots in Figure 6.2 show this behavior for 30 solutions obtained by each algorithm, highlighting the stability of the GRASP when compared to the ILS.

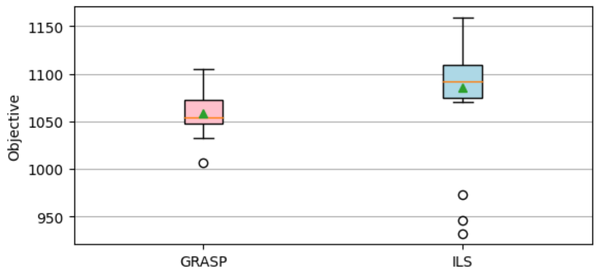
However, when it comes to computing time, ILS proves to be more efficient, especially for instances with more requests. Far beyond the choice of parameters, this can be explained by the fact that in each iteration of GRASP a starting solution is constructed, which makes the algorithm costly when this procedure is performed many times, mainly because of the great number of calls to the Eight Step Evaluation Scheme (Algorithm 6). In fact, *profiling* techniques pointed out that the total consumption of the constructive algorithm accounts for about 33.3% of the total computing time in the proposed GRASP, with the remaining 66.7% being the local search step.



(a) R7b



(b) R3a

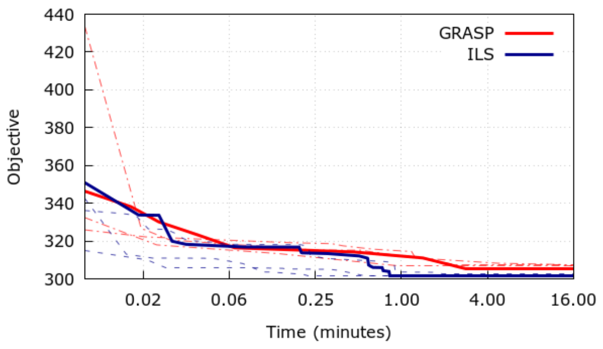


(c) R10a

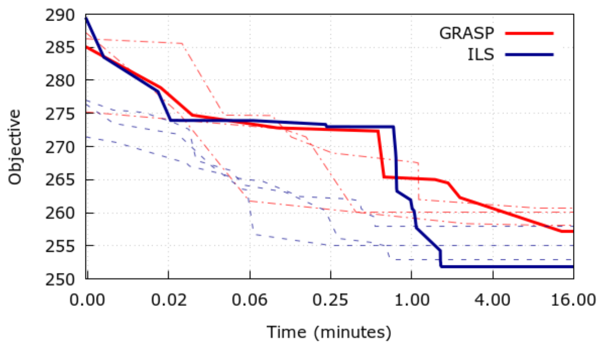
Figure 6.2: Analysis of solution quality in 30 runs of the algorithms in three different instances.

6.3.1 Convergence analysis

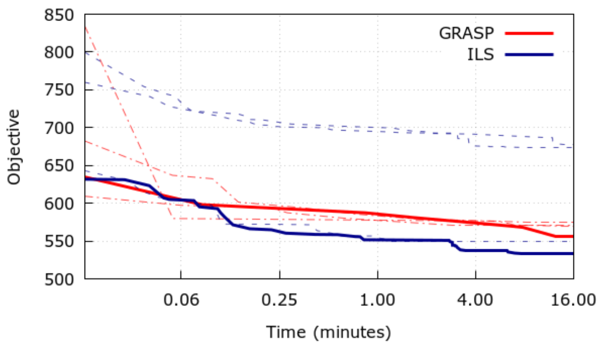
In order to evaluate which algorithm converges more quickly to the final solution, experiments were performed on six different instances, which were classified according to the number of requests: **(i)** small: [24, 72), **(ii)** medium: [72, 108) and **(iii)** large: [108, 144]. Two instances per group were chosen. The maximum running time was set to 16 minutes for small and medium instances, and 64 minutes for the large ones. A total of 4 single-threaded runs on each algorithm was attempted. Results for each instance are shown in Figure 6.3, and the continuous lines in the graphs highlight, for each approach, the executions where the best result was obtained.



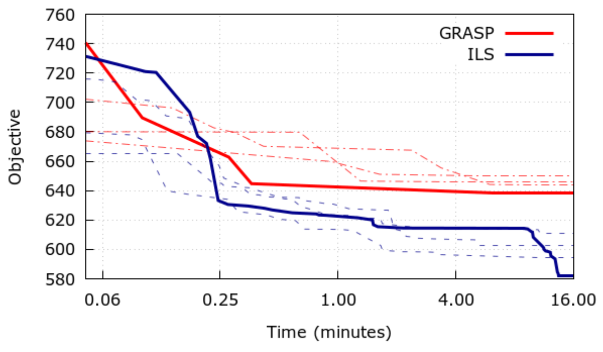
(a) Small: R2a



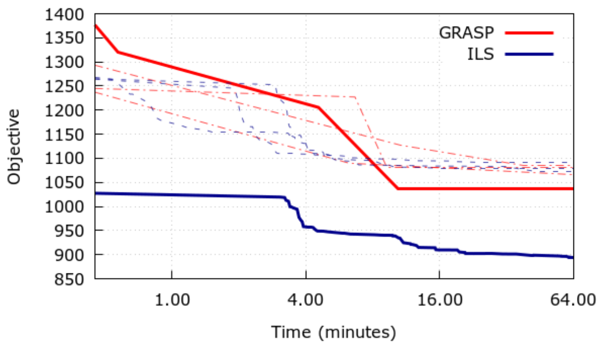
(b) Small: R7b



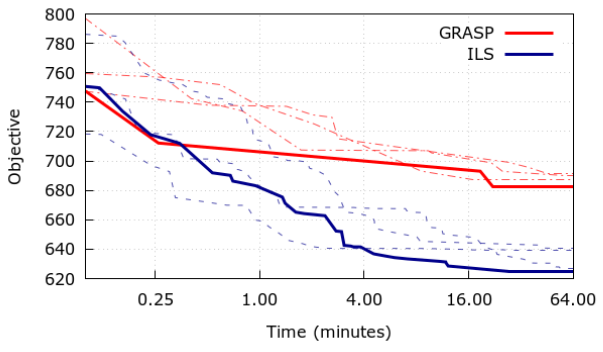
(c) Medium: R3a



(d) Medium: R4a



(e) Large: R10a



(f) Large: R9b

Figure 6.3: Convergence analysis on six instances of different sizes.

In the long run, one can note that the ILS tends to achieve better results overall. However, the GRASP seems to be a rather consistent algorithm, in terms of generating solutions with similar values. This behavior can be observed, e.g., in the instance R3a, in which the ILS outputs two curves very distant from the remaining others, while GRASP maintains a certain consistency. As a matter of fact, these observations are in line with the data presented in Table 6.3, where the proposed ILS proved, on average, to output better solutions within less computing time when compared to GRASP.

7 Conclusion

The *dial-a-ride* problem (DARP) was introduced and a literature review on the current state-of-the-art was provided, along with research opportunities in variants of the problem, such as the multi-deport heterogeneous *dial-a-ride* problem (MD-H-DARP) and the DARP with electric vehicles. The objective of the work was the development of heuristics approaches to the problem, given its $\mathcal{NP}$-hardness character. To that purpose, the mathematical modelling of the problem was presented, following the pioneer work of (CORDEAU; LAPORTE, 2003), and two algorithms based in GRASP and ILS metaheuristics were proposed. The VND procedure was embedded in both approaches in order to perform a systematic exploration in the search space, using three different neighborhood structures of great appeal in vehicle routing problems.

Extensive computational experiments were conducted, using a widely adopted set of challenging benchmark instances from the literature. Numerical results showed the potential of both metaheuristics on providing reasonable solutions. Although they did not outperform the state-of-the-art methods, their behavior could be thoroughly investigated and compared. In general, the ILS provided better results in terms of computing time and quality of solution with respect to the GRASP. Moreover, the main overhead identified in the proposed GRASP was the time demanded by the constructive algorithm, which was about 33% at the bigger picture, mainly due to the nature of the Eight Step Evaluation Scheme (Algorithm 6). In any case, improvements can be made. Other perturbation operators could be tried in the ILS, along with a long-term memory mechanism that allows the usage of previous known solutions. Besides that, one could, e.g., optimize the constructive algorithm, so as to consume less computing time in the GRASP, while adding more neighborhood structures and search strategies to improve the quality of solutions. Another possibility is to try the constructive algorithm with different values of α, so as to observe the general behavior of the algorithm within the search strategies.

In short, two different heuristics approaches were implemented, and their effectiveness were proven good. Furthermore, a constructive algorithm that aims at generating high-quality initial solutions was described and tested. All algorithms proposed can be easily extended and modified, including to other DARP variants. Code and benchmark instances are avaliable at ⟨https://github.com/diegopaiva1/darp⟩.

Bibliography

BONGIOVANNI, C.; KASPI, M.; GEROLIMINIS, N. The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological*, Elsevier, v. 122, p. 436–456, 2019.

BRAEKERS, K.; CARIS, A.; JANSSENS, G. K. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research Part B: Methodological*, Elsevier, v. 67, p. 166–186, 2014.

BREVET, D. et al. A dial-a-ride problem using private vehicles and alternative nodes. *Journal on Vehicle Routing Algorithms*, Springer, v. 2, n. 1-4, p. 89–107, 2019.

CHASSAING, M.; DUHAMEL, C.; LACOMME, P. An els-based approach with dynamic probabilities management in local search for the dial-a-ride problem. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 48, p. 119–133, 2016.

CORDEAU, J.-F. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, INFORMS, v. 54, n. 3, p. 573–586, 2006.

CORDEAU, J.-F.; LAPORTE, G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, Elsevier, v. 37, n. 6, p. 579–594, 2003.

CORDEAU, J.-F.; LAPORTE, G. The dial-a-ride problem: models and algorithms. *Annals of operations research*, Springer, v. 153, n. 1, p. 29–46, 2007.

DAGUM, L.; MENON, R. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, IEEE, v. 5, n. 1, p. 46–55, 1998.

FEIGENBAUM, E. A.; FELDMAN, J. *Computers and Thought*. USA: McGraw-Hill, Inc., 1963. ISBN 0070203709.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, Elsevier, v. 8, n. 2, p. 67–71, 1989.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995.

GLOVER, F. W.; KOCHENBERGER, G. A. *Handbook of metaheuristics*. [S.l.]: Springer Science & Business Media, 2006. v. 57.

GSCHWIND, T.; DREXL, M. Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, INFORMS, v. 53, n. 2, p. 480–491, 2019.

HO, S. C. et al. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, Elsevier, v. 111, p. 395–421, 2018.

JAW, J.-J. et al. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, Elsevier, v. 20, n. 3, p. 243–257, 1986.

JORGENSEN, R. M.; LARSEN, J.; BERGVINSDOTTIR, K. B. Solving the dial-a-ride problem using genetic algorithms. *Journal of the operational research society*, Taylor & Francis, v. 58, n. 10, p. 1321–1331, 2007.

JR, J. W. B.; KAKIVAYA, G. K. R.; STONE, J. R. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, Taylor & Francis, v. 30, n. 2, p. 91–123, 1998.

LIN, S.; KERNIGHAN, B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, INFORMS, v. 21, n. 2, p. 498–516, 1973.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 320–353.

MALHEIROS, I. et al. A hybrid algorithm for the multi-depot heterogeneous dial-a-ride problem. *Computers & Operations Research*, Elsevier, v. 129, p. 105196, 2021.

MASMOUDI, M. A. et al. A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. *Computers & operations research*, Elsevier, v. 81, p. 1–13, 2017.

MASMOUDI, M. A. et al. The dial-a-ride problem with electric vehicles and battery swapping stations. *Transportation research part E: logistics and transportation review*, Elsevier, v. 118, p. 392–420, 2018.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & operations research*, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

MUELAS, S.; LATORRE, A.; PEÑA, J.-M. A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city. *Expert Systems with Applications*, Elsevier, v. 40, n. 14, p. 5516–5531, 2013.

PAQUETTE, J.; CORDEAU, J.-F.; LAPORTE, G. Quality of service in dial-a-ride operations. *Computers & Industrial Engineering*, Elsevier, v. 56, n. 4, p. 1721–1734, 2009.

PARRAGH, S. N. Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C: Emerging Technologies*, Elsevier, v. 19, n. 5, p. 912–930, 2011.

PARRAGH, S. N.; DOERNER, K. F.; HARTL, R. F. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, Elsevier, v. 37, n. 6, p. 1129–1138, 2010.

PIMENTA, V. et al. Models and algorithms for reliability-oriented dial-a-ride with autonomous electric vehicles. *European Journal of Operational Research*, Elsevier, v. 257, n. 2, p. 601–613, 2017.

POTVIN, J.-Y.; ROUSSEAU, J.-M. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, Springer, v. 46, n. 12, p. 1433–1446, 1995.

PSARAFTIS, H. N. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, INFORMS, v. 14, n. 2, p. 130–154, 1980.

PSARAFTIS, H. N. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation science*, INFORMS, v. 17, n. 3, p. 351–357, 1983.

RODRIGUES, P. P. et al. Resolução de um caso real do problema dial-a-ride multicritério via clustering search. *Production*, SciELO Brasil, v. 24, n. 3, p. 572–582, 2014.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, Informs, v. 40, n. 4, p. 455–472, 2006.

SAVELSBERGH, M. W. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, INFORMS, v. 4, n. 2, p. 146–154, 1992.

TALBI, E.-G. *Metaheuristics: from design to implementation.* [S.l.]: John Wiley & Sons, 2009. v. 74.

TOTH, P.; VIGO, D. Heuristic algorithms for the handicapped persons transportation problem. *Transportation science*, INFORMS, v. 31, n. 1, p. 60–71, 1997.

WILSON, N. H. et al. *Scheduling algorithms for a dial-a-ride system.* [S.l.]: Massachusetts Institute of Technology. Urban Systems Laboratory, 1971.