UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO GABRIEL DE SOUZA VEREZA MEDEIROS

Solving the Dial-a-Ride Problem Using Iterated Local Search

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

Advisor: Prof. Dr. Luciana Salete Buriol

Porto Alegre
December 2016

ABSTRACT

The Dial-a-Ride Problem (DARP) is an $\mathcal{NP}$-hard combinatorial problem. The DARP is a variant of the Vehicle Routing Problem (VRP), focusing in the transportation of passengers. It consists in, given a set of requests for pick-up and delivery requested by users and a set of homogeneous vehicle, optimize the assignment of requests to vehicles in order to serve all requests, providing optimal routes and respecting problem constraints such as route duration, vehicle capacity, passenger ride time, departure and arrival time windows. This work presents an approach to DARP using the Iterated Local Search (ILS) metaheuristic, which has been successfully applied in solving other variants of the VRP. The goal of this work was to evaluate the suitability of ILS in the context of DARP. For that, a comparison between obtained results and the ones found in the literature was presented and indicated promising results.

Keywords: Dial-a-ride problem. Iterated Local Search. Metaheuristic. Vehicle routing.

RESUMO

O *Dial-a-Ride Problem* (DARP) é um problema combinacional $\mathcal{NP}$-difícil. O DARP é uma variante do *Vehicle Routing Problem* (VRP), focando no transporte de passageiros. O problema consiste em, dado um conjunto de requisições de coleta e entrega feitas por usuários e um conjunto de veículos idênticos, otimizar a atribuição de requisições a veículos com objetivo de atender todas as requisições, produzindo rotas ótimas e respeitando limitações impostas pelo problema, como capacidade máxima dos veículos, tempo máximo de rota de um passageiro e janelas de tempo no embarque e desembarque. Este trabalho apresenta uma abordagem para o DARP utilizando a metaheurística *Iterated Local Search* (ILS), que tem sido aplicada com sucesso na resolução de outras variantes do VRP. O objetivo desse trabalho foi avaliar a adequação do ILS ao contexto do DARP. Para isso, comparações entre resultados obtidos e resultados disponíveis na literatura foram realizadas e indicaram resultados promissores.

Palavras-chave: Dial-a-ride problem. Iterated Local Search. Metaheuristica. Roteamento de veículos.

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BKS	Best Known Solution
CPU	Central Processing Unit
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-A-Ride Problem
HFVRP	Heterogeneous Fleet Vehicle Routing Problem
ILS	Iterated Local Search
PDVRP	Pickup and Delivery Vehicle Routing Problem
RVND	Random Variable Neighborhood Decent
SD	Standard Deviation
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VRPSD	Vehicle Routing Problem with Stochastic Demands
VRPTW	Vehicle Routing Problem with Time Windows

CONTENTS

1 INTRODUCTION

Modern urban areas face several kinds of mobility problems. With an ever increasing demand for transportation, not only of goods, but also people, many studies have been conducted by the scientific community in order to optimize logistic transportation.

The problem addressed in this work is known as the *Dial-a-Ride Problem* (DARP). Briefly, the problem consists in a set of requests for pick-up and delivery from passengers that need to be served by a limited set of vehicles available. The main goal is to create a set of optimal vehicle routes that serve all requests, complying with a set of constraints. Most common constraints are similar to other vehicle routing problems and include vehicle capacity, route duration and route time. However, as the DARP considers the transport of people, not goods, several quality of service constraints exist, such as passenger ride time, time limit for pick-up and delivering, and type of vehicle used to serve a specific passenger.

Dial-a-ride problems arise, for instance, in the context of patient transportation from and to hospitals, in the transportation of people that live in rural areas and constantly need to travel to neighbor cities for work and health matters, and in the context of providing public service for the disabled and elderly people who cannot use regular public transportation systems easily.

An instance of DARP consists in information about the vehicle, restrictions about the route and details about requests. Considering the case where all vehicles are of the same kind, the following properties apply:

- Number of vehicles available;
- Maximum capacity of a vehicle;
- Maximum route duration;
- Maximum ride duration, i.e the maximum time a passenger can be in route;

Each request in the instance is described with properties:

- Pick-up location;
- Drop-off location;
- Time window for picking-up, i.e. a time window defining when the service must begin;
- Time window for dropping-off, i.e. a time window defining when the service must end;

- Service time, e.g. time needed for boarding or alighting;
- Number of passengers, either boarding or leaving the vehicle;

All vehicles can travel from every location to any other location. The time and cost related to each travel is computed as the euclidean distance between the locations. Real-world applications of DARP must consider actual road distance and traffic, which are not considered in the model at hand.

Due to the application oriented characteristic of DARP, no standardized definition of optimal route exists. Definitions include maximization of number of passengers served, minimization of user waiting time, minimization of total route time, among many others. In this work, an optimal vehicle route is defined as a route that comply with all constraints and minimizes travel costs.

It was proved by Jr, Kakivaya and Stone (1998) that the DARP belongs to the $\mathcal{NP}$-Hard class of problems, which brings several challenges when trying to model and solve the problem. Large instances of the problem come with high complexity due to the size of the search space to be considered, making exact approaches non-suitable due to high time consumption.

In this context, we propose to apply a metaheuristic known as Iterated Local Search (ILS) to the DARP. Several *Vehicle Routing Problem* (VRP) variations were solved using ILS. However, no ILS implementation was found for the DARP variation considered in this work.

It is expected that the results of this work may bring contributions to solving the presented problem and insights on the use of ILS to solve DARP and possibly other problems. By comparing the ILS with other metaheuristics used in the literature, strengths and weakness of ILS are highlighted, as well as its suitability on solving the DARP.

2 LITERATURE REVIEW

According to Cordeau et al. (2007), the DARP is a generalization of the *Vehicle Routing Problem with Pickup and Delivery* (VRPPD). Cordeau and Laporte (2007) state that, from a modelling point of view, the DARP is a generalization of various vehicle routing problems such as the *The Pickup and Delivery Vehicle Routing Problem* (PDVRP) and the *Vehicle Routing Problem With Time Windows* (VRPTW). The difference between DARP and most of other related problems is the fact that people, not goods, are transported. When transporting passengers, concerns about quality of service are introduced in the form of constraints, such as route duration, route length, customer waiting time, customer ride time and difference between actual and desired delivery times.

The DARP is presented in several variants. In the **static** version of DARP, requests are known before vehicles start their route. In the **dynamic** version, requests can be requested after routes have started. Examples of works that solved the dynamic version are described in the survey by Cordeau and Laporte (2007).

Some variants of DARP also consider the number of vehicles available. A variation with **single vehicle** was formulated and solved by Psaraftis (1980) using dynamic programming, whose objective function was the minimization of weighted sum of route completion time and customer dissatisfaction (i.e., the weighted combination of waiting time before pickup and ride time). According to Cordeau and Laporte (2007), one of the first heuristics for the **multiple-vehicle** DARP was proposed by Jaw et al. (1986).

Another variant is with regard on the type of vehicles available. In the **homogeneous** variation, all vehicles available are equal. In the **heterogeneous** variation, vehicles can have different passenger capacity or be adapted to transport a specific kind of passengers, e.g. disabled people that need to be transported with wheelchairs (PARRAGH, 2011, p. 912).

The variation considered in this work is the static homogeneous multi-vehicle DARP. Cordeau and Laporte (2003) described a tabu search heuristic for this variant. As part of the search algorithm, an eight-steps solution evaluation was developed. Cordeau (2006) used a branch-and-cut approach. Parragh, Doerner and Hartl (2010) applied the same preprocessing steps and adapted the eight-steps solution evaluation algorithm proposed by Cordeau and Laporte (2003) for their variable neighborhood search. Jorgensen, Larsen and Bergvinsdottir (2007) formulated and solved DARP using a genetic algorithm to construct clusters of users and a modified nearest neighbor procedure to create the

routes.

In this work, iterated local search (ILS) is used, which, according to Lourenço, Martin and Stützle (2003), contains several of the desirable features of a metaheuristic, such as simplicity, effectiveness, robustness, and ease of implementation. The authors also described successful usage of ILS in the Travelling Salesman Problem (TSP). ILS implementations for variations of the Vehicle Routing Problem (VRP) include Bianchi et al. (2006) for the *Vehicle Routing Problem with Stochastic Demands* (VRPSD), Subramanian et al. (2010) for the *Vehicle Routing Problem with Simultaneous Pickup and Delivery* (VRPSPD), Chen, Huang and Dong (2010) for the *Capacitated Vehicle Routing Problem* (CVRP), and Penna, Subramanian and Ochi (2013) for the *Heterogeneous Fleet Vehicle Routing Problem* (HFVRP).

2.1 Search Algorithms

This section introduces and describes generic implementations of the Iterated Local Search Algorithm and the 2-Opt Algorithm.

2.1.1 Iterated Local Search

The Iterated Local Search algorithm was described by Lourenço, Martin and Stützle (2003), and consists in, given a local optimum solution found by a local search algorithm, use this local optimum to obtain a set of solutions from where the search should continue. Obtaining the set of solutions is achieved through perturbation of the local optimum, which generates a more focused set of solutions to be considered, instead of restarting the search from a completely new solution. ILS is summarized as

> The essence of iterated local search can be given in a nut-shell: one iteratively builds a sequence of solutions generated by the embedded heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic. (LOURENÇO; MARTIN; STÜTZLE, 2003, p. 322)

The algorithm has four procedures: *(i) GenerateInitialSolution*, where an initial solution is constructed; *(ii) LocalSearch*, used to improve a solution; *(iii) Perturb*, which generates a new starting point by perturbing the solution produced by the *LocalSearch* procedure; *(iv) AcceptanceCriterion*, used to determine from which solution the search should continue. Algorithm 1 describes a generic implementation of ILS.

Algorithm 1 Iterated Local Search

```
1: procedure ITERATED LOCAL SEARCH
2:     s_0 ← GenerateInitialSolution
3:     s* ← LocalSearch(s_0)
4:     while Stopping criterion is not met do
5:         s ← Perturb(s*)
6:         s' ← LocalSearch(s)
7:         s* ← AcceptanceCriterion(s*, s', history)
8:     end while
9: end procedure
```

The perturbation step is used to escape from a current locally optimal solution. Such move is frequently done on a larger neighborhood than the one used in the local search step, or in a way that the local search procedure cannot undo in a single step. The acceptance criterion defines the next solution to be perturbed. It's a crucial step for ILS, as it defines the balance between intensification and diversification of the search. This step can optionally have some kind of history mechanism, so that previous known solutions can be used when choosing the next solution instead of a fixed set of rules. Even though the history mechanism normally leads to better results and performance, most of the usage of ILS does not implement it (LOURENÇO; MARTIN; STÜTZLE, 2003).

According to Lourenço, Martin and Stützle (2003), the effectiveness of ILS depends mainly on the local search algorithm, the perturbation and the acceptance criteria. The authors also point out that even the most naive implementations of these procedures tend to yield better results than restarting the search at a random point in the solution space.

2.1.2 Local search 2-opt

The 2-opt algorithm is a local search proposed by Croes (1958) for solving the *Travelling Salesman Problem* (TSP). The main concept of the algorithm is, given a path in graph, remove a pair of arcs and reconnect the vertexes using different arcs: if the modified route is better, according to a criteria, the best route is updated. This procedure is repeated until no further optimization is obtained. Algorithm 2 describes a generic implementation of 2-opt.

Algorithm 2 2-Opt Algorithm

```
1:  function 2-OPT(route)
2:      best ← route
3:      improved ← false
4:      do
5:          improved ← false
6:          for all vertex v_i ∈ route do
7:              for all vertex v_j ∈ route(j ≠ i − 1, j ≠ i + 1) do
8:                  newRoute ← best
9:                  Replace arcs (v_i → v_{i+1}) and (v_j → v_{j+1}) by arcs (v_i → v_j)
10:                 and (v_{i+1} → v_{j+1}) in newRoute
11:
12:                 if newRoute is better than best then
13:                     best ← newRoute
14:                     improved ← true
15:                 end if
16:             end for
17:         end for
18:     while improved ≠ false
19:     return best
20: end function
```

In Algorithm 2, the loop that checks if improvements are still happening is described from lines 4-18. At each iteration of this loop, all pairs of non-consecutive arcs are replaced by a different pair of arcs, described in lines 6-17. The algorithm stops if no replacement of arcs generate a better solution, meaning that the current best is the local optimal solution.

3 THE DIAL-A-RIDE PROBLEM

The DARP is modelled on a directed graph $G = (V, A)$ where V is the set of vertices and A is the set of arcs. For each arc (i, j) a non-negative travel cost c_{ij} and a non-negative travel time t_{ij} are considered (PARRAGH; DOERNER; HARTL, 2010). A total of n customer requests need to be served by m vehicles. Each request consists of a pair of vertexes $(i, i + n)$ where i denotes an origin location and $i + n$ denotes the corresponding destination location. At the origin of each request is assigned a demand of passengers $q_i > 0$ waiting to be transported and a corresponding negative demand $q_{i+n} = -q_i$ at the destination denotes alighting. A time window $[e_i, l_i]$ representing the earliest and the latest time that the service must begin is assigned to each request, either on the origin or the destination. In case of an outbound request, the time window is set on the destination vertex. An inbound request has a time window in the origin vertex. Each vertex has a service time, associated with loading and unloading operations, denoted by d_i. All vehicles have capacity Q and maximum route time T and leave the origin $i = 0$ (depot) and must finish its route in $i = 2n + 1$ (end depot). In order to guarantee quality of service, the ride time of a customer must not exceed L. Departure from vertex i at time D_i results in arriving at vertex j at $A_i = D_i + t_{ij}$. Beginning of service at vertex i cannot start before e_i, therefore, beginning of service is defined as $B_i = max(A_i, e_i)$. Waiting time, i.e. the time the vehicle had to wait at vertex i, is defined as $W_i = B_i - A_i$. Departure time is calculated as $D_i = B_i + d_i$. The ride time of each customer is defined as $L_i = B_{n+i} - Di$. The duration of a route performed by vehicle k is denoted by $B^k_{2n+1} - B^k_0$. The goal is to minimize cost $c(s)$ defined by $\sum_{(i,j)\in s} c_{ij}$, that are defined by arcs (i, j) present in solution s.

3.1 Mathematical formulation

Although the proposed solution doesn't make use of the mathematical formulation, it is a clear description of all constraints present in DARP. The formulation by Cordeau (2006) is described as follows.

Let P be the set of origins, $P = \{1, \ldots, n\}$, D the set of destinations $D = \{n + 1, \ldots, 2n\}$ and N the set of all requests plus the depot $N = P \cup D \cup \{0, 2n + 1\}$. Let K be the set of m vehicles.

For each arc (i, j) and each vehicle $k \in K$, let $x^k_{ij} = 1$ if vehicle k travels from

node i to j. For each node $i \in N$ and each vehicle $k \in K$, B_i^k is the time at which vehicle k begins service at i, Q_i^k is the load of vehicle k after visiting i and L_i^k be the ride time of user i on vehicle k.

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

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

$$\sum_{j\in N} x_{ij}^k - \sum_{j\in N} x_{n+i,j}^k = 0 \qquad \forall i \in P, k \in K \tag{3}$$

$$\sum_{j\in N} x_{0j}^k = 1 \qquad \forall k \in K \tag{4}$$

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

$$\sum_{i\in N} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{6}$$

$$B_j^k \geq (B_i^k + d_i + t_{ij}) x_{ij}^k \qquad \forall i, j \in N, k \in K \tag{7}$$

$$Q_j^k \geq (Q_i^k + q_j) x_{ij}^k \qquad \forall i, j \in N, k \in K \tag{8}$$

$$L_i^k = B_{n+i}^k - (B_i^k + d_i) \qquad \forall i \in P, k \in K \tag{9}$$

$$B_{2n+1}^k - B_0^k \leq T \qquad \forall k \in K \tag{10}$$

$$e_i \leq B_i^k \leq l_i \qquad \forall i \in N, k \in K \tag{11}$$

$$t_{i,n+i} \leq L_i^k \leq L \qquad \forall i \in P, k \in K \tag{12}$$

$$max\{0, q_i\} \leq Q_i^k \leq min\{Q, Q + q_i\} \qquad \forall i \in N, k \in K \tag{13}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i, j \in N, k \in K \tag{14}$$

Total routing cost is minimized in objective function (1). Constraints (2) and (3) ensure that each request is served once by the same vehicle. Routes must start and end in the depot, which is ensured by constraints (4), (5) and (6). Constraint (7) ensures that the beginning of service depends on the beginning of service of the previously visited node, plus the service time in that node, plus the time cost of travelling between nodes. The consistency of the vehicle load is ensured by constraint (8). In (9), the ride time of each user is bounded by constraint (12). Constraint (10) limits the duration of each route. Each request must be served within its time windows, which is ensured by constraint (11). The number of passengers on board must not exceed the maximum capacity of a vehicle,

ensured by constraint (13).

The objective function used in this work also considers not only route cost, but penalty values representing constraint violations, as described in Section 4.2.

4 PROPOSED APPROACH

This chapter presents details about the proposed solution. Solution representation and evaluation are described, as well as preprocessing procedures, route and ride time optimization and an ILS approach adapted to DARP.

4.1 Solution Representation

A solution s is defined as a set of m routes $s = \{r_1, ..., r_m\}$. Each route is represented as a vector of size $2n + 1$ used to indicate succession of nodes. Value i at position 0 indicates that i is the first node visited after the depot, i.e the first arc is $(0, i)$. Next arc in the route is from i to the value j at position i. As an example, consider requests represented as $(origin, destination)$ pairs $(1, 4), (2, 5), (3, 6)$. Figure 4.1 describes a route $r = \{0, 2, 1, 4, 3, 6, 5, 0\}$.

Figure 4.1: Route representation as an array of succession.

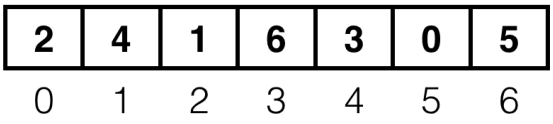

Figure 4.1 shows an indexed succession array. Value j at index i indicates that there is an arc from i to j. In the example, first arc in route is from 0 to 2; second arc is from 2 to 1; third arc is from 1 to 4; and so on until an arc returns to depot, i.e. $j = 0$.

Each route also contains an array for arrival time, beginning of service, departure time, waiting time and load. The value at index i in the arrival times array represents A_i, for the beginning of service array it denotes B_i, for the departure time represents D_i, for the waiting time array it denotes W_i and for load it represents the vehicle load L_i.

Initial expectation was that storing intermediate costs would avoid full cost recalculation every time a route changed, i.e. recalculation would only be done after the changed point, as earlier values wouldn't be affected. But due to the nature of the 8-steps evaluation scheme described in 4.2.1, all vertex in the array must be considered during the cost evaluation of a route.

4.2 Solution Evaluation

Following the work by Cordeau and Laporte (2003), load violation $q(s)$, duration violation $d(s)$, time window $w(s)$ and ride time violation $t(s)$ were penalized in the evaluation function $f(s)$. Load violation is computed as $q(s) = \sum_{i=1}^{2n}(y_i - Q)^+$, duration violation is calculated as $d(s) = \sum_{k=1}^{m}(B_{2n+1}^k - B_0^k - T)^+$, time window violation as $w(s) = \sum_{i=1}^{2n}(B_i - l_i)^+$ and ride time violation as $t(s) = \sum_{i=1}^{n}(L_i - L)^+$. Where $x^+ = \max\{0, x\}$. As an extension, order violation is also penalized. Order violation happens when the destination of a request is visited before its origin and is computed as

$$o(s) = \sum_{i=1}^{n} \begin{cases} 0 & \text{if } B_{i+n} > B_i \\ 1000 & \text{otherwise} \end{cases}$$

Let $c(s)$ denote the sum of distances travelled by each vehicle, which is the sum of costs c_{ij} associated with arcs(i, j) in each route, the evaluation function is given as

$$f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s) + \tau t(s) + \omega o(s)$$

where $\alpha, \beta, \gamma, \tau$ and ω are penalty terms for load violation, duration violation, time window violation, ride time violation and order violation, respectively. In the beginning of the search, all values are set to $\alpha = \beta = \gamma = \tau = \omega = 1$. Every time a new current best solution s^* is found, all penalty terms are either decreased or increased. If a constraint is violated, the corresponding penalty term is increased. On the other hand, if the constraint is satisfied, the penalty term is decreased. If, for example, s^* violates capacity constraint, then $\alpha = \alpha(1 + \delta)$. Similarly if the constraint is satisfied, then $\alpha = \alpha/(1 + \delta)$. The value of δ is randomly chosen in interval $[0.05, 0.10]$. According to Parragh, Doerner and Hartl (2010), using different values of δ works as a diversification mechanism.

4.2.1 Forward time slack

In order to optimize route duration and comply with ride time constraint, the beginning of service in each vertex is set according to the route evaluation scheme presented by Cordeau and Laporte (2003). It's an adaptation to DARP of the forward time slack de-

fined by Savelsbergh (1992). Forward slack time at vertex i is computed as

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

where W_p is the waiting time at vertex p, the last vertex on the route is denoted by q and P_j is the ride time of user whose destination is $j \in \{n+1, \ldots, 2n\}$, $P_j = 0$ for all other j. The cumulative waiting time until 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 ride time and P_j, represents the slack time at j. The forward slack time F_i is the minimum of all slack times between i and q.

It was also noted by Cordeau and Laporte (2003) that delaying the departure from the depot by $\sum_{0<p<q} W_p$ does not affect the arrival time at vertex q. Delaying the departure by more will increase the arrival time at q by the same amount. As a consequence, departure time from the depot should be delayed by at most $\min \left\{ F_0, \sum_{0<p<q} W_p \right\}$.

Forward slack time can also be used to reduce the ride time of a user by delaying B_i at each origin vertex. This led Cordeau and Laporte (2003) to develop an eight-steps evaluation scheme that is also used in the proposed approach. The eight-steps evaluation scheme is described as:

1. Set $D_0 = e_o$.
2. Compute A_i, W_i, B_i and D_i for each vertex i in the route.
3. Compute F_0.
4. Set $D_0 = e_o + \min \left\{ F_0, \sum_{0<p<q} W_p \right\}$.
5. Update A_i, W_i, B_i and D_i for each vertex i in the route.
6. Compute L_i for each request assigned to the route.
7. For every vertex j that is an origin.
   (a) Compute F_j.
   (b) Set $W_j = W_j + \min \left\{ F_j, \sum_{j<p<q} W_p \right\}$; $B_j = A_j + W_j$; $D_j = B_j + d_j$.
   (c) Update A_i, W_i, B_i and D_i for each vertex i that comes after j in the route.
   (d) Update ride time L_i for each request i whose destination vertex is after j.
8. Compute changes in violations of vehicle load, duration, time window and ride time constraints.

(CORDEAU; LAPORTE, 2003, p. 587)

The procedure first minimizes time window violations in steps (1) and (2). Route duration is minimized without increasing time window violations in steps (3) until (6) (CORDEAU; LAPORTE, 2003). Step (7) sequentially minimizes ride time by delaying

the beginning of service in vertexes that are origin of a request, without increasing route duration, time window or ride time violations.

4.3 Preprocessing

Before applying the ILS algorithm, the instance is preprocessed to tighten time windows when possible. Following the work by Cordeau (2006), time windows in inbound requests can be tightened by setting $e_i = max\{0, e_{i+n} - L - d_i\}$ and $l_i = min\{l_i, l_{n+i} - t_{i,n+i} - d_i\}$. Similarly, outbound requests can be tightened by setting $e_{n+i} = max\{0, e_i + d_i + t_{i,n+i}\}$ and $l_{n+i} = min\{l_{n+1}, l_i + d_i + L\}$.

4.4 Iterated Local Search Applied to DARP

Proposed approach uses ILS with some adaptations to include a stop criterion and the mechanism needed to update penalty terms as described in 4.2. Algorithm 3 describes these adaptations.

Algorithm 3 Iterated Local Search - DARP

```
1:  procedure DARP ITERATED LOCAL SEARCH
2:      s_0 ← GenerateInitialSolution
3:      s* ← LocalSearch(s_0)
4:      noImprovement ← 0
5:      while noImprovement < 2000 do
6:          s ← Perturb(s*)
7:          s' ← LocalSearch(s)
8:          s* ← AcceptanceCriterion(s*, s')
9:          if s* did improve then
10:             update penalty terms
11:             noImprovement ← 0
12:         else
13:             noImprovement ← noImprovement + 1
14:         end if
15:     end while
16: end procedure
```

Algorithm 3 is an adaptation from the generic form of ILS presented in Section 2.1.1. Main difference in the algorithm is the update of penalty terms every time a new best solution is found, described in lines 9-11. A counter variable is used to keep track

of iterations in which no better solution was found (line 13), and is reset when the best solution is updated (line 11). This counter is used as the stop criterion of ILS, according to the test made in line 5.

A detailed explanation of each component of the proposed ILS approach is given in the following Subsections.

4.4.1 Initial Solution Generation

The initial solution generation algorithm is based on the work from Parragh, Doerner and Hartl (2010). On their work, all requests were sorted by some artificial beginning of service randomly chosen in the interval $[e_i, l_i]$. All routes are then initialized using the first m requests on the list. Remaining requests are inserted sequentially, i.e origin followed by destination, at the end of a route. The route in which a request is inserted is selected using a randomly chosen minimum distance criteria. Criterion one compares the distance between the origin of the last request in the route and the origin of the request being inserted. Criterion two compares the distance between the origin of the last request in the route and the destination of the request being inserted. Criterion three compares the distance between the destination of the last request and the origin of the request being inserted. Criterion four compares the distance between the destination of the last request and the destination of the one being inserted.

In this work, instead of sorting requests by an artificial B_i randomly chosen between e_i and l_i, the median of interval $[e_i, l_i]$ is used. Using the median causes requests with a tighter time window to be served before requests with a wider time window. Consider two inbound requests with time window $R = [e_r, l_r]$ and $S = [e_s, l_s]$, and that $l_r > l_s > e_s > e_r$. Randomly choosing B_r and B_s may result in $B_r < B_s$, meaning that a request that could be served later is being given preference over a request with a less wide time window. Using the median value, the request with smaller time window is always served first, i.e $B_s < B_r$.

4.4.2 Local Search

Three different implementations of local search were experimented in this work. First implementation uses only intra-route local search. Second implementation uses both

intra-route and inter-route local search for every iteration of ILS. The third implementation uses intra-route search for most iterations, applying inter-route search every 5 iterations. All implementations are described in following subsections and results are discussed in Section 5.4.

4.4.2.1 Intra Route Search: 2-opt

Local search performed on a solution consists in applying a local search for each of its routes. The local search step is an intra-route optimization procedure, as it operates on a single route at a time.

For this implementation the 2-opt Algorithm described in Section 2.1.2 is used. Algorithm 4 describes how 2-opt was adapted to DARP.

Algorithm 4 2-opt-DARP Algorithm

```
1:  function 2-OPT(route)
2:      best ← route
3:      improved ← false
4:      do
5:          improved ← false
6:          for all vertex v_i ∈ route do
7:              for all vertex v_j ∈ route(j ≠ i − 1, j ≠ i + 1) do
8:                  newRoute ← best
9:                  Replace arcs (v_i → v_{i+1}) and (v_j → v_{j+1}) by arcs (v_i → v_j)
10:                 and (v_{i+1} → v_{j+1}) in newRoute
11:
12:                 if didImprove(best, newRoute) then
13:                     best ← newRoute
14:                 end if
15:             end for
16:         end for
17:     while improved ≠ false
18:     return best
19: end function
```

The difference between Algorithm 4 and the generic 2-opt described in Section 2.1.2 is the comparison used. The local search step uses an adaptation of the Acceptance Criterion described in Subsection 4.4.4, operating on routes instead of solutions. The motivation is the same: always prefer feasible routes over infeasible ones, i.e. ensure that if the current best route is feasible, no infeasible route is taken as best. Algorithm 5 describes such comparison.

Algorithm 5 didImprove

```
1:  function DIDIMPROVE(best, newRoute)
2:      if best is feasible then
3:          if newRoute is feasible and f(newRoute) < f(best) then
4:              return newRoute
5:          end if
6:      else
7:          if newRoute is feasible or f(newRoute) < f(best) then
8:              return newRoute
9:          end if
10:     end if
11:     return best
12: end function
```

In the context of ILS, 2-opt is applied to all routes only in its first execution. Following executions of intra-route local search are performed only in routes that changed in the perturbation steps. Intra-route search to routes that remain unchanged is not performed, as it would yield the same results found previously.

4.4.2.2 Intra and Inter Route Search

In this implementation, the intra route search described in 4.4.2.1 is combined with a *Random Variable Neighborhood Descent* (RVND) based on the work by Penna, Subramanian and Ochi (2013). Inter-route operations are used to generate sets of neighborhoods of a given solution. Each inter-route operation is exhaustively applied to a solution s, generating a set of solutions that form a neighborhood of s. The best neighbor of a randomly chosen neighborhood is compared to s. If the neighbor is better than s, then s is updated and the search restarts. If no improvement happens, the neighborhood is removed from the set. Procedure ends when there are no neighborhoods to use. Algorithm 6 describes the RVND function.

Algorithm 6 RVND

```
1:  function RVND(s)
2:      s* ←s
3:      Initialize set of neighborhoods (NL)
4:      while NL ≠ ∅ do
5:          N^(η) ← randomly chosen neighborhood ∈ NL
6:          Find the best neighbor s' of s* ∈ N^(η)
7:          if didImprove(s*, s') then
8:              s* ←2-opt(s')
9:              Update NL
10:         else
11:             Remove N^(η) from NL
12:         end if
13:     end while
14:     return s*
15: end function
```

In Algorithm 6, line 3 initializes NL with all neighborhood structures. Search is described in lines 4-13, which is performed until there are no more neighborhood structures to use. In line 5 a random neighborhood structure is selected from NL, line 6 finds the best neighbor, inside the chosen neighborhood. Line 7 uses the function described in Algorithm 5 to determine if one solution is better than the other. When a better solution is found, an intra-route search is performed using the 2-opt Algorithm, presented in Algorithm 4, and the set NL is updated to include all neighborhood structures. If no improvement happens, line 11 removes the current neighborhood structure from NL.

The set NL is composed of three neighborhood structures based on the work from Penna, Subramanian and Ochi (2013):

- *Swap(1, 1)*: One request from route $\mathbf{r_1}$ is swapped with a request from route $\mathbf{r_2}$. As each request is made of two stops (i.e, origin and destination), both stops need to be swapped. Each request is removed from the original route and inserted into the other, at the same position as the removed one. For example, consider a set of requests $\{(\mathbf{1},\mathbf{7}),(\mathbf{2},\mathbf{8}),(\mathbf{3},\mathbf{9}),(\mathbf{4},\mathbf{10}),(\mathbf{5},\mathbf{11}),(\mathbf{6},\mathbf{12})\}$ and routes $\mathbf{r_1} = \{\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{7},\mathbf{9},\mathbf{8},\mathbf{0}\}$ and $\mathbf{r_2} = \{\mathbf{0},\mathbf{4},\mathbf{6},\mathbf{5},\mathbf{10},\mathbf{11},\mathbf{12},\mathbf{0}\}$, swapping requests $(\mathbf{2},\mathbf{8})$ and $(\mathbf{5},\mathbf{11})$ would result in $\mathbf{r_1} = \{\mathbf{0},\mathbf{1},\mathbf{5},\mathbf{3},\mathbf{7},\mathbf{9},\mathbf{11},\mathbf{0}\}$ and $\mathbf{r_2} = \{\mathbf{0},\mathbf{4},\mathbf{6},\mathbf{2},\mathbf{10},\mathbf{8},\mathbf{12},\mathbf{0}\}$. Swapping request between two routes is a operation with constant cost $O(1)$. However, generating the neighborhood of a solution is done by exhaustively swapping all pairs of requests assigned to different routes. Each swap made generates a neighbor solution. The neighborhood construction has

cost $O(n^2)$.

- *Swap(2, 2)*: Similar to *Swap(1,1)*, but swaps two consecutive requests from each route instead of only one.

- *Shift*: A request is removed from route $\mathbf{r_1}$ and inserted in route $\mathbf{r_2}$. The insertion step adds the new request in the best position in $\mathbf{r_2}$, i.e, the position that results in the smallest cost according to the evaluation algorithm described in Section 4.2. This operation also generates the neighborhood by exhaustively shifting each requests to all possible routes. Each shift made generates a neighbor solution. The neighborhood construction has cost $O(n^2)$.

Two implementations were tested for the combined intra and inter-route search. First implementation performs the RVND search every iteration. Second implementation performs RVND every k iterations without improvement, using the 2-opt local search in most iterations. The motivation for not performing RVND every iteration comes from the high computing time needed by RVND. Considering time growth and cost minimization, second approach was chosen. Details on the tests performed are presented in Section 5.4. The final implementation of local search is described in Algorithm 7.

Algorithm 7 LocalSearch-DARP

```
1: function LocalSearch(s, iterationsWithoutImprovement)
2:     s* ←s
3:     if iterationsWithoutImprovement is multiple of k then
4:         s* ← RVND(s)
5:     else
6:         s* ← 2-opt(s)
7:     end if
8:     return s*
9: end function
```

After preliminary tests, the value of k was defined to $k = 5$.

4.4.3 Perturbation

Perturbation is used in ILS to generate a new starting point for the local search step. This mechanism is used to prevent the search to get stuck in locally optimal solutions. Operators *Swap(1,1)* and *Shift(1,1)* described in Subsection 4.4.2.2 are used in the

perturbation step. The operator to be used in the perturbation phase is randomly chosen. Perturb procedure is described in Algorithm 8.

Algorithm 8 Perturbation Algorithm

```
1:  procedure PERTURB(solution)
2:      r1 ← random route from solution
3:      r2 ← random route from solution, different from r1
4:      randomValue ← random value in interval [0, 1]
5:      if randomValue > 0.50 then
6:          r1Request ← random request from r1
7:          r2Request ← random request from r2
8:          swap r1Request and r2Request in r1 and r2
9:      else
10:         request ← random request from r1
11:         remove request from r1
12:         insert request in the end of r2
13:     end if
14: end procedure
```

Lines 7-9 perform the swap operator, while lines 11-13 perform the shift operator. Note that only one operator is chosen on each execution.

4.4.4 Acceptance Criterion

The acceptance criterion step is used to define from which solution the ILS should continue. This step considers the current best solution and a local-searched solution found after the perturbation step.

The search performed by ILS allows the existence of infeasible solutions in the intra-route search. In some scenarios, the total penalty of an infeasible solution s' may result in $f(s')$ being less than $f(s^*)$, even if s^* is feasible. In order to give preference to feasible solutions, feasibility check is also considered in the acceptance criterion. If the current best solution s^* is feasible, solution s' becomes the best one only if s' is feasible and $f(s') < f(s^*)$. On the other hand, if s^* is infeasible, s' becomes best if it's feasible or $f(s') < f(s^*)$. Algorithm 9 describes how the next solution to be perturbed is chosen by the acceptance criterion procedure.

Algorithm 9 Acceptance Criterion Algorithm

```
1:  function ACCEPTANCE CRITERION(s*, s')
2:      if s* is feasible then
3:          if s' is feasible and f(s') < f(s*) then
4:              return s'
5:          end if
6:      else
7:          if s' is feasible or f(s') < f(s*) then
8:              return s'
9:          end if
10:     end if
11:     return s*
12: end function
```

4.4.5 Stop condition

Stop condition is based on the number of iterations without improvement in the best solution. Arbitrary values between 1000 and 10,000 were tested. After preliminary tests, the maximum number of iterations without improvement was set to 2000.

5 EXPERIMENTAL RESULTS

This chapter presents the evaluation of the proposed approach implementation, solving several problem instances. The evaluation considers the quality of the results, i.e. how close to known best values the final solution is; algorithm progression, i.e. how much the final solution is better than the initial solution; and running times.

The choices taken regarding initial solution generation and the local search procedure are also justified by results comparison.

5.1 Environment

The approach was implemented in *Java* version 1.8.0.77. Tests were executed in a *Macbook Pro*, with a 64-bits *Intel Core i5* processor at 2.7 GHz and 8 GB memory.

5.2 Instances

A set of instances was generated by Cordeau and Laporte (2003), which has been widely used in the literature of DARP. In their work, 20 instances were randomly generated, with requests count varying between 24 and 144. Half of the requests in the instance are inbound requests and half are outbound. Every request consists of a single passenger. Service time is 10 for all origins and destinations. Locations were generated in the square $[-10, 10]^2$. The location of the depot is equal to average location of the seed points used to generated locations. The time windows values are in the interval $[0, 1440]$, and two procedures were used to define earliest and latest times. First procedure chooses an uniform random number e_i in the interval $[60, 480]$ and then an uniform random number l_i in the interval $[e_i + 15, e_i + 45]$. The second procedure chooses random numbers e_i in $[60, 480]$ and l_i in $[e_i + 30, e_i + 90]$. All instances have maximum route duration set to 480, vehicle capacity to 6, and maximum ride time to 90. Table 5.1 describes the instances generated by Cordeau and Laporte (2003).

Table 5.1: Instance Set

Instance	Requests	Vehicles	BKS
R01a	24	3	190.02
R02a	48	5	301.34
R03a	72	7	532.00
R04a	96	9	570.25
R05a	120	11	626.93
R06a	144	13	785.26
R07a	36	4	291.71
R08a	72	6	487.84
R09a	108	8	658.31
R10a	144	10	851.82
R01b	24	3	164.46
R02b	48	5	295.66
R03b	72	7	484.83
R04b	96	9	529.33
R05b	120	11	577.29
R06b	144	13	730.67
R07b	36	4	248.21
R08b	72	6	458.73
R09b	108	8	593.49
R10b	144	10	785.68

5.3 Initial Solution Evaluation

A comparison between an implementation using a randomly chosen B_i in the interval $[e_i, l_i]$ used by Parragh, Doerner and Hartl (2010) and another using $B_i = median(e_i, l_i)$ described in Subsection 4.4.1 is presented in Table 5.2. Column *Cost*[1] presents the value of total costs of solutions generated by randomly choosing B_i in the interval $[e_i, l_i]$. Column *Penalties*[1] describes only penalty values computed due to constraint violations (see Section 4.2). Column *Cost*[2] presents the value of total costs of solutions generated by using $B_i = median(e_i, l_i)$. In column gap^c, the gap between the average cost of the two approaches is presented, calculated as $gap^c = (Cost^2 - Cost^1)/Cost^1$. Column *Penalties*[2] describes only penalty values computed due to constraint violations. Column gap^p, the gap between the average penalty values of the two approaches is presented, calculated as $gap^p = (Penalties^2 - Penalties^1)/Penalties^1$. Values are an average of 20 runs. Values highlighted in bold indicate better results.

Table 5.2: Results of initial solution generation.

Instance	Random		Median			
	Cost[1]	Penalties[1]	Cost[2]	gap^c	Penalties[2]	gap^p
R1a	665.41	355.94	599.20	**-9.95%**	290.43	**-18.40%**
R2a	2484.17	1945.17	2194.13	**-11.68%**	1666.73	**-14.31%**
R3a	5632.91	4523.91	4724.66	**-16.12%**	3612.50	**-20.15%**
R4a	6211.41	4963.00	4752.85	**-23.48%**	3489.58	**-29.69%**
R5a	8256.00	6884.64	7615.26	**-7.76%**	6252.51	**-9.18%**
R6a	9460.31	7639.20	8348.76	**-11.75%**	6530.55	**-14.51%**
R7a	1804.96	1304.69	1604.00	**-11.13%**	1111.92	**-14.78%**
R8a	7338.79	6315.29	6254.42	**-14.78%**	5229.92	**-17.19%**
R9a	13948.57	12479.15	13138.00	**-5.81%**	11672.10	**-6.47%**
R10b	17384.07	15455.29	14617.43	**-15.91%**	12705.57	**-17.79%**
R1b	496.47	198.24	381.93	**-23.07%**	79.40	**-59.95%**
R2b	1469.05	925.60	1358.20	**-7.55%**	820.74	**-11.33%**
R3b	3580.07	2487.65	3030.65	**-15.35%**	1935.25	**-22.21%**
R4b	5354.97	4105.08	4380.52	**-18.20%**	3146.00	**-23.36%**
R5b	5115.33	3741.23	3284.75	**-35.79%**	1927.92	**-48.47%**
R6b	7558.81	5751.58	5757.57	**-23.83%**	3951.95	**-31.29%**
R7b	1390.31	908.52	1064.54	**-23.43%**	575.75	**-36.63%**
R8b	4057.28	3025.63	3573.97	**-11.91%**	2548.04	**-15.78%**
R9b	8304.44	6833.12	7482.72	**-9.89%**	6010.21	**-12.04%**
R10b	14473.70	12537.77	13561.91	**-6.30%**	11616.25	**-7.35%**
Average				**-15.18%**		**-21.54%**

[1]: Random B_i implementation.
[2]: B_i as median implementation.

For all instances used, the median value resulted in better cost and penalties compared to a random value, with average improvement in total cost equal to 15.18% and to penalties by 21.54%. This confirms the expectation that choosing the median value would generate better solutions.

5.4 Local Search Comparison

As described in Section 4.4.2, three implementations for local search were considered. One using only intra-route operations, another using both inter and intra-route operations on every iteration of ILS, and a third one using inter-route operations every $k = 5$ iterations. A comparison between the implementations was performed by using a few instances from the instance set and the final solution cost comparison is presented in Tables 5.3 and computing time comparison in Table 5.4. Values are average of 10 runs.

Table 5.3: Cost comparison between local search implementations.

Instance	Gap[1]	Gap[2]	Gap[3]
R01a	+12.33%	**+2.97%**	+3.57%
R02a	+22.80%	**+12.65%**	+21.40%
R03a	**+27.48%**	+27.89%	+36.91%
R07a	+15.58%	**+10.94%**	+14.25%
R01b	+17.28%	+14.61%	**+12.00%**
R02b	+18.63%	+17.29%	**+12.02%**
R07b	+21.49%	+21.60%	**+14.83%**
Average	**+19.37%**	**+15.42%**	**+16.43%**

[1]: Intra-route only.
[2]: Intra and inter-route every iteration.
[3]: Inter-route every 5 iterations.
Values are average of 10 runs.

Table 5.4: Time comparison between local search implementations.

Instance	CPU[1]	CPU[2]	CPU[3]
R01a	**0.89**	3.15	0.95
R02a	**3.94**	79.54	10.04
R03a	**9.59**	188.48	33.05
R07a	**2.04**	17.26	4.18
R01b	1.08	2.33	**0.66**
R02b	11.30	32.70	**10.61**
R07b	**3.10**	7.10	3.67

[1]: Intra-route only.
[2]: Intra and inter-route every iteration.
[3]: Inter-route every 5 iterations.
All CPU values are in minutes.

As expected, introducing inter-route operations as part of the local search step produces better results, as shown in Table 5.3. Intra-route operations optimize a route in an isolated way. In order to perform a better optimization in a solution, which is a set of routes, inter-route operations have to be performed before optimizing routes individually.

These operations, however, come with a considerable growth in time consumed. It is clear in Table 5.4 that the intra-route only implementation performs better than the combination of inter and intra-route. The difference is mainly due to the large number of times a route cost has to be recalculated as part of the *Shift(1,1)* operation described in Section 4.4.2.2, as the operator inserts a request in the best position in a route, i.e. the one that minimizes total cost.

Due to the massive time consumption from running the inter-route local search for

every iteration, and intermediate approach was taken. Running the inter-route search every 5 iterations and using intra-route search for all other iterations delivered better results considering final solution cost and CPU time.

It was expected, however, that using inter-route search more often wold generate better final solutions for all instances, as it performs a more complete search than intra-route operations. The actual cause needs further investigation, but one possible explanation is that the perturbed solution used in RVND described in Section 4.4.2.2 is not submitted to an intra-route local search before the neighborhoods generation, meaning that even the best neighbor of a perturbed solution may have higher cost than the solution found by performing an intra-route search in the perturbed solution.

5.5 Algorithm Progress and Results Quality

This experiment considers the progress of the algorithm, i.e. how better the final solution is when compared to the initial solution, and the gap between the average cost of final solutions and the best known values from the literature. Table 5.5 presents the results. The first two columns contain information about the instance tested, column *Initial* presents the cost of the solution initially generated (as described in Section 4.4.1) and its gap to the best known values. Column *Final* presents the final solution cost reported by the ILS approach proposed in this work followed by a column containing its gap to the best known values. Column *Dev from Initial* reports the deviation of the final solutions to the *Initial* solution, calculated as *Dev to Initial* $= (Initial - Final)/Initial$. Finally, column *Iterations* presents how many iterations of the ILS algorithm were needed on average to find the final solution. Values are average of 10 runs, except when marked with $^{+}$, which are average of 3 runs.

Table 5.5: Algorithm progress and results quality.

Instance	BKS	Initial	Gap[1]	Final	Gap[2]	Dev to Initial	Iterations
R01a	190.02	622.75	+227.73%	196.81	+3.57%	68.39%	4860
R02a	301.34	2135.50	+608.67%	365.82	+21.40%	82.87%	4533
R03a	532.00	4928.10	+826.33%	728.37	+36.91%	85.22%	4773
R04a$^+$	570.25	5817.19	+920.11%	751.06	+31.71%	87.08%	4223
R05a	626.93	6018.17	+859.94%	832.10	+32.73%	86.17%	2956
R07a	291.71	1494.90	+412.46%	333.28	+14.25%	77.70%	5836
R08a	487.84	5113.38	+948.17%	611.21	+25.29%	88.04%	5364
R09a$^+$	658.31	4963.64	+653.99%	811.37	+23.25%	86.65%	5541
R01b	164.46	538.37	+227.36%	184.19	+12.00%	65.78%	3613
R02b	295.66	1353.89	+357.92%	331.21	+12.02%	75.53%	5269
R03b	484.83	3432.30	+607.94%	582.65	+20.18%	83.02%	7183
R04b$^+$	529.33	4365.83	+724.78%	631.51	+19.30%	85.53%	6623
R05b$^+$	577.29	2733.30	+373.47%	664.62	+15.13%	75.68%	9151
R07b	248.21	1319.74	+431.70%	285.03	+14.83%	78.40%	4815
R08b	458.73	3981.04	+767.84%	538.92	+17.48%	86.46%	5451
R09b$^+$	593.49	12118.47	+1941.90%	716.44	+20.72%	94.08%	5639
Average			**+680.64%**		**+20.05%**	**81.66%**	**5364**

Gap[1]: Gap from initial solution to BKS.
Gap[2]: Gap from final solution to BKS.
Values are average of 10 runs.
$^+$: average of 3 runs.

From the set of 20 instances, 16 were solved. Considering the gaps from final solution to the best known values, good solutions were found for smaller instances, for example instance *R01a*, *R01b*, and *R02b*. It is also notable that the gap value increases as instances include more requests and vehicles, since the search spaces become dramatically bigger. Exploring larger search spaces require a better balance between *exploration* (how much of the search space is visited) and *exploitation* (how deep the search goes to find the local optimal value). The balance on ILS is given by the strength of the perturbation step. By making weak perturbations to current best solutions, exploration becomes difficult, causing considerable time being spent in a region of the solution space that does not contain the global optimal solution. On the other hand, too strong perturbations lead to high *exploration* and low *exploitation*, meaning that the depth of the search in a promising region of the solution space may be too superficial and better solutions are skipped.

Considering the progress from an initial solution to a final solution, deviation from final to initial varied from 65.78% to 94.08%. This is mostly due to the fact that infeasible solutions are very likely to be created by the initial solution generation step. Solution costs becomes higher due to penalties added to the cost, which did not happen for final

solutions, meaning that all final solutions found in the evaluation were feasible.

5.6 Running Times

Table 5.6 presents a comparison between computing time taken by the proposed approach and the computing time from two solvers from the literature. Column Gap[1] presents the gap between the results found and the best known values, while CPU[1] contains computing time, both values reported in the work by Cordeau and Laporte (2003)(measured on a *Pentium 4*, 2 GHz computer). Column Gap[1] presents the gap between the results found and the best known values, while column CPU[2] has the computing time, both values reported in the work by Parragh, Doerner and Hartl (2010) (executed on a *Pentium D* computer with 3.2 GHz). Last two columns present the gap and computing time from the proposed solution. Values are average of 10 runs, except for values noted with $^{+}$, which are average of 3 runs.

Table 5.6: CPU comparison.

Instance	Gap[1]	CPU[1]	Gap[2]	CPU[2]	GAP-ILS	CPU-ILS
R01a	0.41%	1.90	0.00%	8.98	3.57%	0.95
R02a	0.75%	8.06	0.15%	20.02	21.40%	10.04
R03a	1.74%	17.18	0.77%	33.21	36.91%	33.05
R04a	4.68%	28.77	1.40%	63.73	31.71%	136.52^{+}
R05a	5.77%	46.24	1.72%	154.47	32.73%	164.40
R07a	1.05%	4.39	0.87%	9.62	14.25%	4.18
R08a	1.61%	20.44	1.61%	52.54	25.29%	46.09
R09a	3.77%	50.51	2.26%	143.08	23.25%	147.49^{+}
R01b	0.32%	1.93	0.00%	12.87	12.00%	0.66
R02b	2.45%	8.29	1.19%	25.89	12.02%	10.61
R03b	3.37%	18.54	1.94%	44.32	20.18%	71.90
R04b	5.40%	31.18	2.11%	127.43	19.30%	189.60^{+}
R05b	2.48%	54.33	1.94%	273.99	15.13%	419.11^{+}
R07b	1.08%	4.23	0.00%	16.81	14.83%	3.67
R08b	3.04%	22.86	2.23%	56.96	17.48%	44.76
R09b	2.42%	51.28	2.43%	152.58	20.72%	288.59^{+}
Average	**2.52%**		**1.29%**		**20.05%**	

[1]: (CORDEAU; LAPORTE, 2003).
[2]: (PARRAGH; DOERNER; HARTL, 2010).
Values reported in minutes.
Values are average of 10 runs.
$^{+}$: average of 3 runs.

Considering running times, the proposed solution performed better than the work by Parragh, Doerner and Hartl (2010). However, the running time from the work by Cordeau and Laporte (2003) is considerably smaller for larger instances. This is possibly explained by the nature of the inter-route search performed in the proposed approach, specially the usage of the *Shift(1,1)* neighborhood structure. By inserting a request in a route in its best possible position, several cost recalculations must be performed to find which position yields the smaller cost. The 8-steps evaluation is not a trivial operation, as it iterates over the route multiple times, making it a bottleneck when executed too often. A possible second cause relates to the result quality described in Section 5.5, poor balancing between *exploration* and *exploitation* may result in considerable time spent in non promising regions of the search space.

Although in some instances the ILS approach was faster than the work by Parragh, Doerner and Hartl (2010), all gap values from ILS are greater than the ones achieved by Cordeau and Laporte (2003) and by Parragh, Doerner and Hartl (2010). This is a possible indication that the ILS was stopping prematurely, which indicates that the stop criterion used needs to be reviewed. Best gap results were found on smaller instances, namely *R01a, R07a, R01b, R02b, R05b, R07b*, and *R08b*, meaning that 2000 iterations without improvement is a reasonable value for small instances. Instance with more requests and vehicles seem to need a greater number of iterations without improvement. Perhaps an ideal solution would be to calculate the number of iterations without improvements considering the instance size.

6 CONCLUSION

In this work, several variants and approaches proposed in the literature for the *Dial-a-Ride Problem* (DARP) were initially presented. The metaheuristic known as Iterated Local Search (ILS) was presented and proposed as a way to solve the DARP, with the assumption that, due to the good results found in applying ILS to other VRP variants, the ILS would be a suitable choice to solve DARP. Finally, an implementation of ILS for DARP was described, evaluated and compared to other approaches from the literature.

Combined with ILS, several techniques and approaches were used based on works that provided good results to DARP or VRP variants. By using and adapting the same techniques and combining them with ILS, it was expected that equally good results would be found. The evaluations presented show that the objective was reached. The proposed solution delivered results with reasonable optimality, which evidences its suitability to the problem at hand.

One notable contribution from this work is the improvement made in the initial solution generation algorithm proposed byParragh, Doerner and Hartl (2010), which generated better results for all instances used in the evaluation.

It was noted, however, that some improvements could still be made to the proposed approach. The usage of a Random Variable Neighborhood Descent delivered better results than the usage of intra-route 2-opt only, but also caused a great increase in average time needed to solve instances of the problem. The growth is expected due to the nature of the procedure, but bottlenecks were noticed. One of the bottlenecks was in the *Shift(1,1)* neighborhood structure: because it adds requests in the best position in a route, several cost recalculations must be performed. The solution representation was designed to keep record of intermediate values in order to speed up cost calculation. However, due to the nature of the 8-steps evaluation, which need to iterate over the whole route to perform optimization, such improvement was not achieved. In order to improve performance, the 8-step evaluation scheme needs to be optimized, or some other alternative to solution evaluation must be created. Other neighborhood structures also need to be tested, specially the ones specific to DARP, as the ones used in this work were fairly generic and could be used in most VRP variants.

Another possible improvement is to investigate how dynamically setting the perturbation strength could lead to better results. Perhaps a stronger, or even multiple, operations could be used in the perturbation step when the ILS has been on the same optimal

solution for several iterations.

To sum up, an ILS solution for the Dial-a-Ride problem was described and implemented. The proposed solution showed good results to most instances used for testing. The knowledge generated from this research indicates that ILS is a promising metaheuristic to be used for DARP and possibly for its variants.

REFERENCES

BIANCHI, L. et al. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. **Journal of Mathematical Modelling and Algorithms**, Springer, v. 5, n. 1, p. 91–110, 2006.

CHEN, P.; HUANG, H.-k.; DONG, X.-Y. Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. **Expert Systems with Applications**, Elsevier, v. 37, n. 2, p. 1620–1627, 2010.

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.

CORDEAU, J.-F. et al. Transportation on demand. **Handbooks in operations research and management science**, Elsevier, v. 14, p. 429–466, 2007.

CROES, G. A. A method for solving traveling-salesman problems. **Operations research**, INFORMS, v. 6, n. 6, p. 791–812, 1958.

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**, Nature Publishing Group, 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.

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

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.

PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. **Journal of Heuristics**, Springer, v. 19, n. 2, p. 201–232, 2013.

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.

ROPKE, S.; CORDEAU, J.-F.; LAPORTE, G. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. **Networks**, Wiley Online Library, v. 49, n. 4, p. 258–272, 2007.

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.

SUBRAMANIAN, A. et al. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. **Computers & Operations Research**, Elsevier, v. 37, n. 11, p. 1899–1911, 2010.