UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS NOHRA HADDAD

AN EFFICIENT HEURISTIC FOR ONE-TO-ONE PICKUP AND DELIVERY PROBLEMS

NITERÓI

2017

UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS NOHRA HADDAD

AN EFFICIENT HEURISTIC FOR ONE-TO-ONE PICKUP AND DELIVERY PROBLEMS

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Area: Algorithms and Optimization

Advisor:
LUIZ SATORU OCHI

Co-advisor:
SIMONE DE LIMA MARTINS

NITERÓI

2017

MATHEUS NOHRA HADDAD

AN EFFICIENT HEURISTIC FOR ONE-TO-ONE PICKUP AND DELIVERY PROBLEMS

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Area: Algorithms and Optimization

Niterói

2017

“The mind, once expanded to the dimensions of larger ideas, never returns to its original size.”

Oliver Wendell Holmes

À minha família.

Agradecimentos

Aos meus pais Aloisio e Kika pela ótima educação e pelo incentivo em toda minha vida acadêmica.
Aos meus irmãos Eduardo e Thais pelo carinho e união.
Aos meus avós paternos Rafick (em sua memória) e Maria, e avós maternos Ivo e Laila pelo amor transmitido e apoio.
À minha namorada Fernanda por todo amor, amizade, apoio e paciência durante todo o doutorado.
À minha família que me incentivou e torceu pela minha vitória.
À família Teixeira Lima que me acolheu e também torceu pela minha vitória.
Aos grandes amigos formados graças à OptHouse, em especial Alan, Bino, Cris, Ed, Guerine, Heder, Igor, João, Marculino e Uéverton.
Aos inseparáveis amigos Dings, Marcelo, Samuel Evangelista e muitos outros por estarem sempre a meu lado durante o doutorado.
Aos professores que possibilitaram tal feito, em especial Fábio Protti, Marcone Souza e Yuri Abitbol graças aos seus comentários e contribuições bastante pertinentes.
Aos orientadores Luiz Satoru e Simone Martins pela confiança depositada em mim e pela responsabilidade, tentando sempre construir o melhor caminho para a elaboração da pesquisa científica.
À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de estudos de Doutorado.
To Professor Thibaut Vidal I express my sincerely gratitude for all the shared knowledge, support and help, being most of the time a truly advisor.
To Professor Richard Hart and the University of Vienna for the very good reception and collaboration during my sandwich PhD period in Vienna.

Resumo

Esta tese trata dois problemas de coleta e entrega um-para-um: o Problema de Roteamento de Veículos com Coleta e Entrega Fracionária Um-para-um (PRVCEFU) e o Problema de Roteamento de Veículos com Coleta e Entrega Premiada (PRVCEP). Ambos são problemas $\mathcal{NP}$-difíceis e importantes para várias aplicações práticas, incluindo o transporte de produtos a granel por navios e sistemas de compartilhamento de bicicletas. Três atributos desafiadores de roteamento de veículos são combinados neste estudo, coleta e entrega um-para-um, coleta e entrega fracionária e seleção de clientes, que são conhecidos por exigirem vizinhanças avançadas de buscas locais e procedimentos de exploração eficientes. Inicialmente, desenvolvemos um algoritmo heurístico baseado na metaheurística Iterated Local Search (ILS) e no método Random Variable Neighborhood Descent (RVND) para resolver o PRVCEFU. O núcleo deste algoritmo consiste em uma busca em grande vizinhança que lida eficientemente com localizações de coleta e entrega e cargas fracionárias, reduzindo o problema de encontrar a melhor combinação de inserção de um par de coleta e entrega em uma rota para um problema de caminho mínimo com recursos limitados, que é resolvido via programação dinâmica. Em seguida, os rótulos não dominados são obtidos para cada rota separada e um problema da mochila é resolvido para encontrar o melhor plano de inserção, considerando a solução completa. Experimentos computacionais em instâncias da literatura mostraram o desempenho competitivo do algoritmo proposto, encontrando novas melhores soluções em 92 de 93 casos e também superando os métodos anteriores na média. Novas instâncias para o PRVCEFU e também novos resultados são relatados, produzindo consistentemente boas soluções em várias execuções. Os bons resultados encontrados no PRVCEFU nos motivaram a adaptar este algoritmo para resolver o PRVCEP. Então, novamente, um ILS combinado com o RVND foi usado, porém agora a ideia foi permitir que o algoritmo aceitasse soluções inviáveis durante seu processo de busca. A fim de reduzir o espaço de busca de soluções inviáveis, o conceito de busca local granular também foi incorporado ao algoritmo. O algoritmo proposto teve um desempenho competitivo em instâncias da literatura, encontrando 20 resultados melhores ou iguais em 36 instâncias do que os métodos anteriores.

Palavras-chave: roteamento de veículos; coleta e entrega um-para-um; coleta e entrega fracionária; seleção de clientes; metaheurísticas; grandes vizinhanças

Abstract

This thesis deals with two one-to-one pickup and delivery problems: the Multi-vehicle one-to-one Pickup and Delivery Problem with Split Loads (MPDPSL) and the Multi-Vehicle Profitable Pickup and Delivery Problem (MVPPDP). Both are $\mathcal{NP}$-hard problems and important for various practical applications, including bulk product transportation by ships and bike-sharing systems. Three challenging vehicle routing attributes are combined in this study, one-to-one pickup and delivery, split loads, and selection of customers, which are known to require advanced local-search neighborhoods and exploration procedures. Initially, we developed a heuristic algorithm based on Iterated Local Search (ILS) and Random Variable Neighborhood Descent (RVND) to solve the MPDPSL. The core of this algorithm consists of a new large-neighborhood search that efficiently deals with pickup and delivery locations and split loads reducing the problem of finding the best insertion combination of a pickup and delivery pair into a route to a resource-constrained shortest path problem, which is solved via dynamic programming. Next, non-dominated labels are obtained for each separate route and a knapsack problem is solved to find the best insertion plan considering the whole solution. Computational experiments on benchmark instances showed the great performance of the proposed algorithm, finding new best solutions on 92 out of 93 instances and also outperforming previous methods in average. New instances for the MPDPSL and also new results are reported, producing consistently good solution on several runs. The good results found on the MPDPSL motivated us to adapt this algorithm for solving the MVPPDP. Then, again, an ILS combined with a RVND was used, but now the idea was to allow the algorithm to accept infeasible solutions during its search process. In order to reduce the search space of infeasible solutions, a granular local search concept was also incorporated into the algorithm. The proposed algorithm had a great performance on instances from the literature, finding 20 better or equal results on 36 instances than previous methods.

Keywords: vehicle routing, one-to-one pickup and delivery; split loads; customer selection; metaheuristics; large neighborhoods

List of Figures

List of Algorithms

List of Tables

Glossary

BKS	:	Best Known Solution;
COG	:	Center of Gravity;
DARP	:	Dial-a-Ride Problem;
ILS	:	Iterated Local Search;
GLS	:	Guided Local Search;
LNS	:	Large Neighborhood Search;
LTL	:	Less-than-truckload;
MPDPSL	:	Multi-Vehicle One-to-one Pickup and Delivery Problem with Split Loads;
MVPPDP	:	Multi-Vehicle Profitable Pickup And Delivery Problem;
PDP	:	One-to-one Pickup and Delivery Problem;
PDPSL	:	Pickup and Delivery Problem with Split Loads;
PTP	:	Profitable Tour Problem;
RVND	:	Random Variable Neighborhood Search;
SDVRP	:	Split Delivery Vehicle Routing Problem;
TOP	:	Team Orienteering Problem;
VND	:	Variable Neighborhood Descent;
VNS	:	Variable Neighborhood Search;
VRP	:	Vehicle Routing Problem;

Contents

Chapter 1

Introduction

The great competition that exists in the industrial sphere means that more and more industries need technological tools to aid their decision-making. With the help of these tools, the entire process from production to sale of products becomes more efficient, facilitating industrial management. It is known that in many countries, as in Brazil, the road system is considered the main form of distribution of products and services. Obtaining a reduction of expenses with the transportation of the products can enable a reduction of the price of the final product. In this way, computational methods responsible for solving vehicle routing problems are of great importance for industries.

In the transportation field, the Vehicle Routing Problem (VRP) is an important combinatorial optimization problem that has been the focus of extensive research effort since 1960 [36]. The VRP aims to find minimum-cost itineraries to service a set of geographically distributed customers with a fleet of vehicles, in such a way that each customer is visited once and the capacity of each vehicle is respected. Over the years, the classical version of the VRP has been increasingly-well solved, but as new applications are discovered, many additional constraints, objectives, and other decision subsets, called "attributes" in [68], are combined with the classical problem, leading to new challenges.

In the meantime, a vehicle routing attribute has drawn considerable research attention over the years: the "one-to-one" pickup and delivery problem, which requires each service to be performed as a pair, in such a way that each pickup precedes its associated delivery, and both visits occur in the same route (see [22], [11] and [49] for surveys on pickup and delivery problems). Once again, an essential ingredient of state-of-the-art heuristics for this problem is the efficient exploration of a variety of neighbor solutions during the search, a task which tends to be more complex when pairs of deliveries are relocated or exchanged instead of single visits.

More recently, two problems have gained prominence in the context of one-to-one pickup and delivery problems: the Multi-Vehicle One-to-one Pickup and Delivery Problem with Split Loads (MPDPSL) [23] and the Multi-Vehicle Profitable Pickup And Delivery Problem (MVPPDP)[28].

The MPDPSL aims to find itineraries for a homogeneous vehicle fleet in order to collect and deliver products to a set of geographically-distributed one-to-one pickup and delivery customers. These itineraries must take into account that a customer may be serviced by several vehicles, each one taking one part of the product. The objective in the MPDPSL is to find minimum distance routes using at most all available vehicles, respecting capacity and maximum driving distance per vehicle, as well as precedence between pickups and deliveries. The MPDPSL is $\mathcal{NP}$-hard [46] and commonly found in practice, for example in parcel delivery services or car hauling applications. Combining pickups and deliveries with split loads leads to a difficult problem, even for neighborhood-centered heuristics, due to the large variety of possible service modes for each product. As a consequence few methods are available to address this problem.

The MVPPDP is a similar problem compared to the MPDPSL. It also has a set of paired pickup and delivery customers that have to be attended by a homogeneous vehicle fleet. However, it does not allow a pair of customers to be visited by more than one vehicle. Another difference is that there is a revenue to be obtained associated with each customer if this customer is visited by any vehicle. The objective in the MVPPDP is to find the set of customers and routes that maximizes the total profit, which is given by the sum of all revenues minus the total travel costs. These routes must respect the capacity, the maximum travel time and the precedence of pickups over deliveries. The MVPPDP is also $\mathcal{NP}$-hard [28] and has practical importance, for example in tramp ship routing. It is also noteworthy that in the MVPPDP, not necessarily all pairs of customers must be visited by a vehicle. Thus, a method for solving the MVPPDP must have an efficient procedure for choosing which pairs of customers are going to be visited, by which vehicles and in which order.

Given the great difficulty of finding optimal solutions for these problems, heuristic strategies are usually used. In this way, the goal is to obtain good quality solutions in acceptable computational times. The interest in the incorporation of exact models in heuristics has grown, one reason is because the commercial software for solving these models have become more powerful. Another reason is because the exact resolution of subproblems is considered to be an acceptable procedure to include into a heuristic, as subproblems are normally easier to solve by exact methods. This combination is known as “hybrid algorithm”.

Thus, the main objective of this thesis is to propose the study of an efficient heuristic to solve one-to-one pickup and delivery problems. Initially, we designed a hybrid heuristic algorithm based on the Iterated Local Search (ILS) and on the Random Variable Neighborhood Descent (RVND) to solve the MPDPSL. The method relies on various local-search neighborhoods to cover all problem characteristics, combining changes of pickup locations, delivery locations, and load-splitting decisions. Larger neighborhoods are also considered, by reducing the problem of finding the best combination of pickup and deliveries with split loads – for one product – to a resource-constrained shortest path and solving it by dynamic programming. Extensive experiments on benchmark instances demonstrate the very good performance of the method on classic instances, outperforming previous algorithms within the same computational time. In addition, new instances and solutions are introduced for the MPDPSL.

After showing the effectiveness of the first algorithm, we investigated its performance on a similar one-to-one pickup and delivery problem and we adapted our algorithm to solve the MVPPDP. Now, the large-neighborhood proposed previously was not fully usable anymore, as it considers inclusions of split loads. As the MVPPDP is a very restrictive problem, we designed a heuristic algorithm that allows working with infeasible solutions, related to the violation of the duration constraint. The idea is to explore the infeasible search space of the MVPPDP and see if better feasible solutions can be obtained. This algorithm adopts some previously-defined neighborhoods, new neighborhoods for increasing the profits and also a repairing neighborhood that tries to get a feasible solution from an infeasible one. Working with infeasible solutions makes the search space much larger and, thus, we adopted the idea of granular local search to reduce the time spent on each neighborhood. Computational experiments were conducted on benchmark instances from the literature and the results show the great ability of the proposed method to achieve good solutions, being able to even find new best solutions within the same time limits of state-of-the-art algorithms.

1.1 Structure of the Thesis

The remainder of this thesis is structured as follows.

Chapter 2 describes the basic concepts of the methods used in this thesis. The Multi-

Vehicle One-to-one Pickup and Delivery Problem with Split Loads is studied in Chapter 3. The contributions on the Multi-Vehicle Profitable Pickup and Delivery Problem are detailed in Chapter 4. Finally, in Chapter 5, the conclusions and future works are presented.

Chapter 2

Basic Concepts

2.1 Variable Neighborhood Descent

The Variable Neighborhood Descent (VND) [43] explores the space of solutions through systematics changes of the neighborhood structures. The authors were inspired by three principles when they created this method:

- A local optimum with respect to one neighborhood structure is not necessarily so for another;
- A global optimum is a local optimum with respect to all possible neighborhood structures;
- For many problems local minima with respect to one or several neighborhoods are relatively close to each other.

According to the authors, an important and empirical observation is that a local optimum usually contains information about the global optimum [12, 33]. This information can be several variables with the same value in both. However, it is not usual to know which are these variables, which induces systematic investigations of the neighborhood structure that contains this local optimum until a better solution is found.

In the Algorithm 1 the pseudocode of the VND applied to a minimization problem is presented. The input data is an evaluation function f, a neighborhood set N, a number of different neighborhood structures r and a solution s. At each iteration of the VND a local search is performed on a neighborhood structure, starting with the first one. At the end of this local search, the best neighbor of the current solution related to this neighborhood structure is obtained. If this best neighbor is not better than the current solution, then a new local search in the next neighborhood structure is performed. If this best neighbor is better than the current solution, then the current solution becomes that best neighbor and the local search is restarted in the first neighborhood structure. The method ends when no improvement is found in any of the adopted neighborhoods and returns the best solution obtained during its execution, that is, the best solution with respect to all neighborhood structures.

Algorithm 1: Variable Neighborhood Descent

```
input  : $f(.), N(.), r, s$
output: Refined solution $s$
1  $k \leftarrow 1$ ;                          /* Current neighborhood structure */
2  while ($k \leq r$) do
3      Find the best neighbor $s' \in N^{(k)}(s)$;
4      if ($f(s') < f(s)$) then
5          $s \leftarrow s'$;
6          $k \leftarrow 1$
7      end
8      else
9          $k \leftarrow k + 1$
10     end
11 end
12 Return $s$;
```

The classic version of VND considers searches on neighborhood structures following a pre-established order. However, recent work has proven the efficiency of an approach that establishes random orders for exploring the neighborhood structures, producing a better diversification of the search. The successful use of such strategy, called Random Variable Neighborhood Descent (RVND), is reported in [59] and [60].

2.2 Iterated Local Search

Iterated local search (ILS) [40] is a global optimization method that explores the search space by successive perturbations from local optima solutions. By doing so, the intention is to prevent a search stagnation in a local optimum which is considerably worse than the global optimum. In a perturbation, the number of modified components of a solution is called the perturbation strength. The perturbations must be strong enough to conduct the search to different regions in search space, but they also need to be weak enough to avoid behaving like a random restart. It is important to notice that, according to [40], a good perturbation may turn an excellent solution in an excellent starting point for the local search phase.

Algorithm 2: ILS

```
   input  : $s$
   output: Refined solution $s$
1  $s_0 \leftarrow$ GenerateInitialSolution();
2  $s^* \leftarrow$ LocalSearch($s_0$);
3  while termination condition is not met do
4      $s' \leftarrow$ Perturbation($s^*$);
5      $s'^* \leftarrow$ LocalSearch($s'$);
6      $s^* \leftarrow$ AcceptanceCriterion($s^*$, $s'^*$,);
7  end
```

To develop an ILS algorithm, four procedures must be defined: *GenerateInitialSolution* (line 1), to provide an initial solution for the heuristic; *LocalSearch* (line 5), to explore the search space in order to find a local optimum; *Perturbation* (line 4), which modifies the current solution in a new point of the search space and *AcceptanceCriterion* (line 6), to determine which solution will be used in the next iteration.

Chapter 3

Multi-Vehicle One-to-one Pickup and Delivery Problem with Split Loads

3.1 Introduction

A classical restriction of the VRP is that each delivery is done in one block, by a single vehicle. In [26], the authors raised this restriction, allowing the total demand of a customer to be served during several visits, leading to the split delivery vehicle routing problem (SDVRP). At first, one might think that allowing split deliveries results in increased costs since more visits may be performed. Yet, this relaxation leads to a larger solution space, opening the way to lower costs. This problem variant is known to be notably more difficult to solve than the classical VRP from an exact method standpoint, and requires more advanced classes of neighborhoods to be properly solved via metaheuristics [57].

As both "pickup and deliveries" and "split deliveries" attributes lead to more complex neighborhood searches, combining them together in one vehicle routing variant poses significant methodological challenges, thus partly explaining the reduced number of methods proposed for the Multi-vehicle one-to-one Pickup and Delivery Problem with Split Loads (MPDPSL) despite its practical relevance, for example in bike-sharing systems. So far, two main articles have considered this variant. The first paper on the topic, by [47], addresses a practical application faced by a logistic company which provides outsourced services. Later on, [23] solve this problem for the transportation of bulk products by ship, where each load is already packaged in multiple containers, and mail services collect and deliver multiple packets between origin and destination pairs. This second method produces solutions of higher quality than [47] on the available instances. Note that, despite the claim of a multi-vehicle algorithm in [23], the experimental analyses of both papers are restricted to the case with a single vehicle (PDPSL). This is due to the fact that, without any global resource constraint on a route (e.g., distance or time), and since the depot does not act as a replenishment facility, it is always profitable and feasible to merge successive trips.

The aim of this Chapter is to pursue on this research line, by proposing improved heuristic resolution approaches for the MPDPSL, new larger neighborhoods with efficient exploration procedures, and experimental analyses on distance-constrained multi-vehicle benchmark instances. More precisely, we propose a hybrid heuristic called IPDS, based on iterated local search (ILS) with random variable neighborhood descent (RVND), which incorporates classical neighborhoods and shaking procedures with larger dynamic programming-based neighborhoods for joint service reinsertions and optimization of split loads. As such, the key contributions of this Chapter are:

1. a simple and efficient hybrid heuristic for the MPDPSL;
2. new dynamic programming based neighborhoods for split pickup-and-delivery problems;
3. new state-of-the-art results for single-vehicle benchmark instances; and finally,
4. new experimental analyses on new multi-vehicle benchmark instances.

3.2 Problem statement

The Multi-Vehicle One-to-one Pickup and Delivery Problem with Split Loads (MPDPSL) was defined by [23] when generalizing its single-vehicle version, the Pickup and Delivery with Split Loads (PDPSL), proposed by [47].

Consider a graph $G = (V, E)$, where $V = P \cup D \cup \{0, 2n+1\}$ includes the vertices associated to n pickup and delivery (p-d) pairs of customers as well as the vertices $\{0, 2n+1\}$, representing the initial and final depots locations. The set $P = \{1, 2, \dots, n\}$ represents the pickup customers, while the set $D = \{n+1, \dots, 2n\}$ represents the corresponding delivery customers. In this way, each service i has a pickup customer $i \in P$ and the corresponding delivery customer $(n + i) \in D$. The set of edges is defined as $E = \{(i, j) | i, j \in V^2\}$.

A homogeneous fleet with m vehicles is available. Each vehicle $k \in \{1, \dots, m\}$ has the same capacity limit C. A demand q_i is associated to each pickup and delivery customer $i \in V$. This demand is positive, $q_i > 0$, for the pickup customer $i \in P$ and negative,

$q_{n+i} = -q_i$, for the delivery customer $(n+i) \in D$. As well as previous works and without loss of generality, we also assume that $q_i \leq C$ for all services i. Each vehicle must depart and return from the depot without any load, that is, $q_0 = q_{2n+1} = 0$. When a vehicle arrives at a pickup customer, it can collect all available load or just a part of it. Moreover, when a vehicle arrives at a delivery customer, all load from this vehicle intended for this customer is delivered. A distance cost d_{ij} is associated with each edge $(i,j) \in E$, and each vehicle is constrained by a maximum travel distance T.

The objective of the MPDPSL is to design a set of up to m routes, starting and ending at the depot, with minimum travel distance, in such a way that the whole service of each pickup and delivery is satisfied, the route distance limit is respected as well as vehicle capacities, and each pickup precedes its delivery in the same route.

3.2.1 Mathematical formulation

A mathematical model for the MPDPSL is presented in [46]. This mathematical model is a nonlinear programming formulation that uses techniques from the formulation of the PDPSL with "Soft" Time Windows [54]. The following notation was used in this nonlinear programming formulation:

3.2.1.1 Parameters

$\boldsymbol{d_{ij}}$ is the amount required for delivery between pickup customer i and delivery customer j,

$\boldsymbol{c_{ij}}$ is the distance between node i and node j, assuming that costs are symmetric such that $c_{ij} = c_{ji}$.

$\boldsymbol{P}$ is the set of pickup customers

$\boldsymbol{D}$ is the set of delivery customers

$\boldsymbol{N_i}$ is the number of possible stops at a pickup customer (or delivery customer) i on a route

$\boldsymbol{K}$ is the maximum number of possible routes (dependent on the total loads to delivered)

$\boldsymbol{M}$ is a large number

3.2.1.2 Decision Variables

$\boldsymbol{x_{in_ik}}$ is the distance the vehicle has traveled on the k-th route when visiting the i-th pickup customer for the n_i-th time,

$\boldsymbol{y_{jn_jk}}$ is the distance the vehicle has traveled on the k-th route when visiting the j-th delivery customer for the n_j-th time,

$\boldsymbol{z_{ijn_in_jk}}$ is the amount picked up from pickup customer i on the n_i-th visit to be dropped off at delivery customer j on the n_j-th visit on the k-th route,

$\boldsymbol{u_{ijn_in_jk}}$ is 1 if the n_i-th visit to pickup customer i is before the n_j-th visit to pickup customer j on the k-th route; 0 otherwise,

$\boldsymbol{v_{ijn_in_jk}}$ is 1 if the n_i-th visit to pickup customer i is before the n_j-th visit to delivery customer j on the k-th route; 0 otherwise,

$\boldsymbol{w_{ijn_in_jk}}$ is 1 if the n_i-th visit to delivery customer i is before the n_j-th visit to delivery customer j on the k-th route; 0, otherwise,

$\boldsymbol{r_{in_ik}}$ is 1 if pickup customer i is visited n_i times on route k; 0 otherwise,

$\boldsymbol{t_{jn_jk}}$ is 1 if delivery customer i is visited n_i times on route k; 0 otherwise,

$\boldsymbol{y_{depot,k}}$ is the distance the vehicle has traveled when it returns to the depot on the k-th route,

$\boldsymbol{p_k}$ is 1 if route k is used; 0 otherwise.

3.2.1.3 Formulation

The nonlinear programming formulation was stated as:

$$\min \sum_{k \in K} y_{depot,k} \tag{3.1}$$

Subject to:

$$\sum_{j \in D} \sum_{n_j \in N_j} z_{ijn_in_jk} \leq 1 - \sum_{l \in P: l \neq i || n_l < n_i} \sum_{j \in D} \sum_{n_l \in N_l} \sum_{n_j \in N_j} z_{ljn_ln_jk} u_{lin_ln_ik} v_{ijn_in_jk}$$

$$i \in P, n_i \in N_i, k \in K \qquad (3.2)$$

$$c_{ij} r_{in_ik} r_{jn_jk} - M(1 - r_{in_ik}) \leq x_{in_ik} - x_{jn_jk} + M u_{ijn_in_jk} \leq M - c_{ij} r_{in_ik} r_{jn_jk}$$

$$i, j \in P, i \neq j, n_i, n_j \in N_j, k \in K \qquad (3.3)$$

$$c_{ij} r_{in_ik} t_{jn_jk} - M(1 - r_{in_ik}) \leq x_{in_ik} - y_{jn_jk} + M v_{ijn_in_jk} \leq M - c_{ij} r_{in_ik} t_{jn_jk}$$

$$i \in P, j \in D, n_i, n_j \in N_j, k \in K \qquad (3.4)$$

$$c_{ij} t_{in_ik} t_{jn_jk} - M(1 - t_{in_ik}) \leq y_{in_ik} - y_{jn_jk} + M w_{ijn_in_jk} \leq M - c_{ij} t_{in_ik} t_{jn_jk}$$

$$i, j \in D, i \neq j, n_i, n_j \in N_j, k \in K \qquad (3.5)$$

$$y_{depot,k} - y_{jn_jk} \geq c_{j,depot} p_k \qquad j \in D, n_j \in N_j, k \in K \qquad (3.6)$$

$$x_{i1k} \geq c_{i,depot} p_k \qquad i \in P, k \in K \qquad (3.7)$$

$$r_{in_ik} \leq M \sum_{l \in D} \sum_{n_l \in N_l} z_{iln_in_lk} \qquad i \in P, n_i \in N_i, k \in K \qquad (3.8)$$

$$M r_{in_ik} \geq \sum_{l \in D} \sum_{n_l \in N_l} z_{iln_in_lk} \qquad i \in P, n_i \in N_i, k \in K \qquad (3.9)$$

$$t_{jn_jk} \leq M \sum_{l \in P} \sum_{n_l \in N_l} z_{ljn_ln_jk} \qquad j \in D, n_j \in N_j, k \in K \qquad (3.10)$$

$$M t_{jn_jk} \geq \sum_{l \in P} \sum_{n_l \in N_l} z_{ljn_ln_jk} \qquad j \in D, n_j \in N_j, k \in K \qquad (3.11)$$

$$u_{ijn_in_jk} \leq M \sum_{l \in D} \sum_{n_l \in N_l} z_{iln_in_lk} \qquad i, j \in P, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.12)$$

$$u_{ijn_in_jk} \leq M \sum_{l \in D} \sum_{n_l \in N_l} z_{jln_jn_lk} \qquad i, j \in P, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.13)$$

$$u_{iin_im_ik} = r_{im_ik} \qquad i \in P, n_i, m_i \in N_i, m_i > n_i, k \in K \qquad (3.14)$$

$$v_{ijn_in_jk} \leq M \sum_{l \in D} \sum_{n_l \in N_l} z_{iln_in_lk} \qquad i \in P, j \in D, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.15)$$

$$v_{ijn_in_jk} \leq M \sum_{l \in P} \sum_{n_l \in N_l} z_{ljn_ln_jk} \qquad i \in P, j \in D, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.16)$$

$$w_{ijn_in_jk} \leq M \sum_{l \in P} \sum_{n_l \in N_l} z_{lin_ln_ik} \qquad i, j \in D, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.17)$$

$$w_{ijn_in_jk} \leq M \sum_{l\in P} \sum_{n_l\in N_l} z_{ljn_ln_jk} \qquad i,j \in D, n_i \in N_i, n_j \in N_j, k \in K \tag{3.18}$$

$$w_{jjn_jm_jk} = t_{jm_jk} \qquad j \in D, n_j, m_j \in N_j, m_j > n_j, k \in K \tag{3.19}$$

$$\sum_{i\in P}\sum_{j\in D}\sum_{n_i\in N_i}\sum_{n_j\in N_j}\sum_{k\in K} z_{ijn_in_jk} = d_{ij} \tag{3.20}$$

$$x_{in_ik} \leq Mr_{in_ik} \qquad i \in P, n_i \in N_i, k \in K \tag{3.21}$$

$$x_{in_ik} \geq r_{in_ik} \qquad i \in P, n_i \in N_i, k \in K \tag{3.22}$$

$$y_{jn_jk} \leq Mt_{jn_jk} \qquad j \in D, n_j \in N_j, k \in K \tag{3.23}$$

$$y_{jn_jk} \geq t_{jn_jk} \qquad j \in D, n_j \in N_j, k \in K \tag{3.24}$$

$$\sum_{i\in P}\sum_{n_i\in N_i} x_{in_ik} \leq Mp_k \qquad k \in K \tag{3.25}$$

$$\sum_{j\in D}\sum_{n_j\in N_j} y_{jn_jk} \leq Mp_k \qquad k \in K \tag{3.26}$$

$$\sum_{i\in P}\sum_{j\in D}\sum_{n_i\in N_i}\sum_{n_j\in N_j} z_{ijn_in_jk} \leq Mp_k \qquad k \in K \tag{3.27}$$

$$x_{in_ik} + \epsilon \leq x_{in_{i+1}k} + M(1 - r_{in_{i+1}k}) \qquad i \in P, n_{i+1} \in N_i, k \in K \tag{3.28}$$

$$y_{jn_jk} + \epsilon \leq y_{jn_{j+1}k} + M(1 - t_{jn_{j+1}k}) \qquad j \in D, n_{j+1} \in N_j, k \in K \tag{3.29}$$

$$r_{in_ik} \geq r_{in_{i+1}k} \qquad i \in P, n_i \in N_i, k \in K \tag{3.30}$$

$$t_{jn_jk} \geq t_{jn_{j+1}k} \qquad j \in D, n_j \in N_j, k \in K \tag{3.31}$$

$$z_{ijn_in_jk} \leq v_{ijn_in_jk} \qquad i \in P, j \in D, n_i, n_j \in N_j, k \in K \tag{3.32}$$

$$p_{k+1} \leq p_k \qquad k \in K \tag{3.33}$$

$$x_{in_ik} \geq 0 \qquad \forall i \in P, n_i \in N_i, k \in K \tag{3.34}$$

$$y_{jn_jk} \geq 0 \qquad \forall j \in D, n_j \in N_j, k \in K \tag{3.35}$$

$$z_{ijn_in_jk} \geq 0 \qquad \forall i \in P, j \in D, n_i \in N_i, n_j \in N_j, k \in K \tag{3.36}$$

$$y_{depot,k} \geq 0 \qquad \forall k \in K \tag{3.37}$$

$$u_{ijn_in_jk} \in \{0,1\} \qquad \forall i,j \in P, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.38)$$

$$v_{ijn_in_jk}, e_{ijn_in_jk} \in \{0,1\} \qquad \forall i \in P, j \in D, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.39)$$

$$w_{ijn_in_jk} \in \{0,1\} \qquad \forall i,j \in D, n_i \in N_i, n_j \in N_j, k \in K \qquad (3.40)$$

$$r_{in_ik} \in \{0,1\} \qquad \forall i \in P, n_i \in N_i, k \in K \qquad (3.41)$$

$$t_{jn_jk} \in \{0,1\} \qquad \forall j \in D, n_j \in N_j, k \in K \qquad (3.42)$$

$$p_k \in \{0,1\} \qquad \forall k \in K \qquad (3.43)$$

The objective (3.1) is to minimize the total distance traveled by all vehicles, when they arrive at the final depot. The capacity of each vehicle is controlled by constraints (3.2). Constraints (3.3-3.5) define the order of the pickup and delivery customers and force to correctly calculate the distances required to travel. Constraints (3.6-3.7) state that if a vehicle is used, then it must leave from the initial depot and finishes its travel at the final depot. Constraints (3.8-3.9) ensure that a pickup customer is only visited if a load is picked up and vice versa. The same idea is also valid for a delivery customer (3.10-3.11). Constraints (3.12-3.19) give conditions for the precedence variables, if a costumer is not visited, then they are set to zero. Constraints (3.20) ensure that all loads must be picked up. Constraints (3.21-3.24) define that if no distance was traveled to reach a customer, then this customer is not visited and vice versa. Constraints (3.25 - 3.26) prohibit customers on a route from being visited if the route is not used. Constraints (3.27) impede that loads are delivered by a route that does not exist. Constraints (3.28 - 3.31) prevent customers from being visited out of order, based on n_i. Constraints (3.32) prohibit a load to be delivered if the delivery customer is visited before the corresponding pickup customer. Constraints (3.33) maintain the order of the routes.

3.2.2 Illustrated examples

Figure 3.1 illustrates a possible solution for an instance with seven p-d pairs, served by two vehicles. In this example, the set of pickup customers is $P = \{1,2,3,4,5,6,7\}$ and the set of corresponding delivery customers is $D = \{8,9,10,11,12,13,14\}$. The initial and final depots are in the same location, represented by 0. The capacity of the vehicle is $C = 100$ and the maximum travel distance is $T = 2100$. In this solution *Route* $1 = \{0,5,3,10,6,13,7,12,14,1,8,4,11,0\}$ and *Route* $2 = \{0,5,2,9,12,0\}$. The distance for *Route* 1 is 1708.58 and the distance for *Route* 2 is 652.97, thus the total

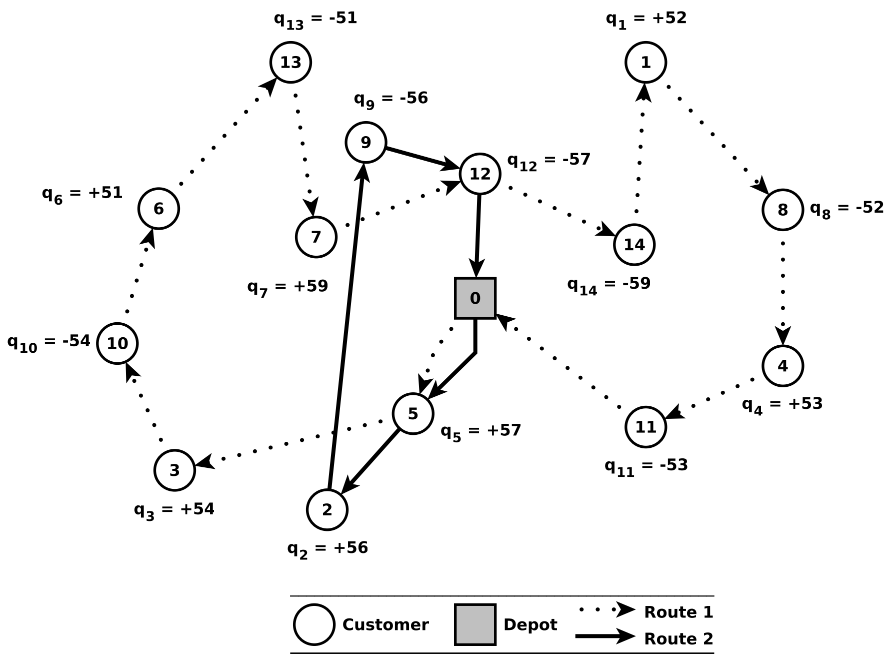

Figure 3.1: Example of a possible solution in MPDPSL

distance is 2361.55.

Analysing the p-d pair from customer 5 to customer 12, we observe that $q_5 = +57$ units must be collected at customer 5 and this load must be delivered at customer 12 (that is, $q_{12} = -57$). This service is fulfilled partly by vehicle 1 (16 units), and partly by vehicle 2 (41 units).

We adopt for the MPDPSL the same definition of a split load as in the literature about the SDVRP [8]. A split load occurs when a load is partitioned into more than the minimum number of splits that are necessary to fulfil all demand. Therefore, if a demand is 24 and the vehicle capacity is 10, the minimum number of trips required to fulfil such a demand is 3. If this demand is supplied in 4 or more trips, then the service is considered as being split. Note that the MPDPSL is notably different from the SDVRP as more than one split load can occur in the same route in the optimal solution. Due to this characteristic, the size of a solution can increase significantly. In some cases this growth can even be exponential, as illustrated in Figure 3.2.

In this example, there are only two pairs of customers and one vehicle with capacity $C = 100$ and distance limit $T = \infty$. Customer 1 requires 99 units of load, while customer 2 needs 100 units of load. The distance between customer 2 and customer 4 is very small ($d_{24} = d_{42} = \epsilon$). Thus, the optimal solution for this vehicle is, firstly, to visit customer

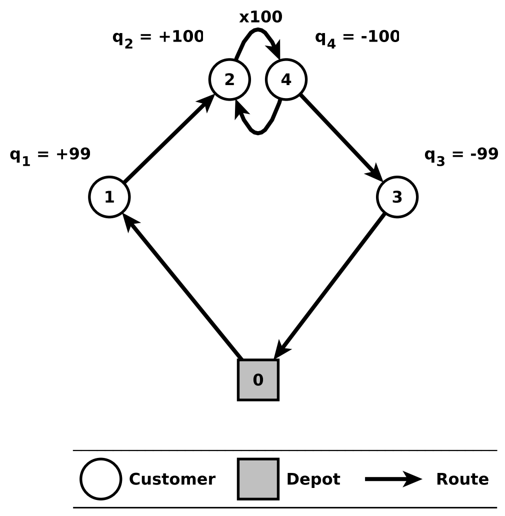

Figure 3.2: In the MPDPSL, there is no bound on the number of multiple splits in a single route. An example.

1 and collect 99 units of load. Then it would go to customer 2 and collect 1 unit of load, delivering this load at customer 4 and repeating this operation until the demand is fulfilled, that is, more 99 times collecting 1 unit at customer 2 and delivering at customer 4. Finally, the vehicle would deliver the remaining 99 units at customer 3 and go back to the depot. The optimal solution performs 202 visits instead of 4. This is of course an extreme case of the classical MPDPSL, as in practical applications a base service time would be counted (e.g., as part of the travel distance), hence increasing d_{24}. Nevertheless, a good heuristic should be able to find multiple split loads, as these situations can naturally occur.

3.3 Related literature

3.3.1 One-to-one Pickup and Delivery Problem (PDP)

In [22], the authors reported the developments in the are of exact and heuristic algorithms for one-to-one PDPs.

A tabu search heuristic was presented in [21] for the multi-vehicle Dial-a-Ride Problem (DARP), in which there may be a time window, for each pair of customers on their desired departure or arrival time. The objective of this problem is to find a set of minimum cost vehicle routes capable of accommodating all requests, respecting vehicle capacity, route duration and a maximum travel time. The authors proposed a procedure for neighbourhood evaluation, which adjusts the time for visiting the customers on the routes to minimize distance and ride times. Results are reported on randomly generated and real-life instances.

In [10] a two-stage heuristic is presented for the PDP with time windows. The first stage applies a Simulated Annealing to minimize the number of vehicles and the second stage applies Large Neighborhood Search (LNS) [55] to minimize the total distance. The computational results showed that the algorithm has improved 10 (17%) of the 56 best published solutions to the Solomon benchmarks and also matching or improving the best solutions in 46 problems (82%).

An adaptive large neighbourhood search heuristic was proposed in [53] for the same problem (PDP with Time Windows). This heuristic is an extension of the previously proposed LNS and contains some sub-heuristics that are used based on their historic performance. Tests were performed on more than 350 instances with up to 500 requests and the algorithm was able to improve more than 50% of the best known solutions in the literature.

3.3.2 Split Delivery Vehicle Routing Problem (SDVRP)

The first papers dealing with the Split Delivery Vehicle Routing Problem (SDVRP) were [26, 27]. The authors showed how advantageous the VRP is when allowing split deliveries, influencing not only on the total distance traveled, but also on the reduction of the number of vehicles used. A two-stage heuristic was developed to solve SDVRP. The first stage builds a solution for the VRP using the Clarke & Wright Saving's algorithm [19]. In the second stage local searches are performed on this initial solution using two moves proposed by the authors. The first move, called *k-split interchange*, duplicates the visit to a customer on different routes that have sufficient residual capacity to meet the customer's demand. The second move, named *route addition*, removes all existing split deliveries of a customer and creates a new route that will only visit this customer with its full demand. In both phases swap-based (*swap*) and edge-removal (*2-opt*) moves, both inter-route and intra-route, were also used. Furthermore, 540 instances were created, with up to 150 customers, which served to perform the tests.

In [6] the authors developed an algorithm based on Tabu Search to solve the SDVRP. This algorithm consists of three phases and uses the strategy *Route First - Cluster Second.* Initially, in the first phase, a solution is constructed using the GENIUS algorithm [29] to create the routes. In the second phase, insertion moves are performed, removing a customer that is served on one or more routes and reinserting it in another route. When reinserting the customer, the residual capacity of the vehicle is taken into account. The cheapest insertion criterion is used for each reinsertion. In the third and last phase, the GENIUS algorithm is again applied in an attempt to improve the resulting solution. It is also noted that if there are *k-split* cycles, they are removed in the third phase. The authors performed computational experiments with 49 instances with up to 199 clients, such instances were created by the authors themselves.

A memetic algorithm with population management was proposed by [13]. The algorithm consists of a Genetic Algorithm combined with a local search procedure and uses a distance measure to control population diversity. The local search is performed using three types of moves. The first one is based on the *k-split interchange* [26, 27], with a procedure that seeks for the best way to split the delivery of a customer. The other two moves are based on exchanges of pairs or trios of customers, considering the possibility of splitting the delivery of these customers. The algorithm was tested on the instances of [6] and was able to overcome the Tabu Search, from the same work, in many cases.

The SDVRP which imposes the use of a minimum number of vehicles was addressed in [15]. The authors proposed to use the Scatter Search meta-heuristic to address this problem. Two heuristic procedures are designed to generate the initial solution, one named Big Tour and the other based on the Clarke & Wright Savings algorithm. Both were adapted to incorporate split deliveries into the solution. The algorithm also uses local searches based on classical-adapted VRP moves to improve solutions. The authors also tested their algorithm on the instances of [6] and concluded that the results can be compared to the best known results so far.

Again, the SDVRP with a minimum number of vehicles was also addressed in [2, 3] and the authors introduced a constructive procedure that is based on an angle control measure. The built solution passes through local searches using the Variable Neighborhood Descent. The moves used in these searches are the classical of the VRP and some specific to the SDVRP. Later, in [1], the authors used some of these ideas and opted for a Tabu Search with vocabulary building to solve the same problem. This vocabulary building is a population-based approach that from a set of solutions can discover good attributes from these solutions that can generate new quality solutions. This approach was able to greatly improve the last two algorithms developed by the same authors.

A study on metaheuristics based on local searches was performed in [25]. The idea was to adapt four classic VRP moves to deal with split deliveries. The following metaheuristics were tested: Simulated Annealing, Threshold Accepting, Record-to-Record, Attribute-based Local Search Beam and Attribute-based Hill Climber. The latter was the one that presented the best results when compared to results found in the literature.

In [56] a multi-start heuristic based on the Iterated Local Search was proposed. The algorithm uses the Random Variable Neighborhood Descent in the local search phase of ILS. Classical neighborhood structures of the VRP are explored, as well as specific structures for the SDVRP. The initial solution is generated using two greedy criteria, cheapest insertion and nearest insertion, and two insertion strategies, parallel and sequential. Perturbations are characterized by the use of the *Multiple-k-split* mechanism that applies the *k-split interchange* method to 5, 6 or 7 customers, such amount is selected randomly. Computational tests were performed with 324 instances found in the literature and the algorithm was able to match or outperform the best results in the literature on 90% of the cases.

3.3.3 Pickup and Delivery Problem with Split Loads (PDPSL)

In the literature, the classic pickup and delivery problem (PDP) is widely represented. Still [41] was the first work on a problem related to the PDPSL, with the difference that it considers one commodity with simultaneous pickups and deliveries instead of one-to-one requests. The objective consists of first minimizing the fleet size and then the distance. The paper proposed a mixed integer programming (MIP) formulation for this problem and a route construction heuristic, which firstly determines the minimum number of vehicles required and then builds routes based on a cheapest insertion criterion. An additional MIP formulation and an extension of this heuristic, using parallel clustering, were later proposed in [42].

The benefits of allowing split loads in the one-to-one PDP was evaluated in [47], hence defining the PDPSL. The objective of the PDPSL is to find a single route with minimum cost, fulfilling the required demand. A heuristic based on simulated annealing and tabu search was developed and random large-scale instances were created. The authors observed that the benefits of split loads are closely linked to three characteristics of the instances: the load size, the cost associated with the pickup or delivery and the percentage of loads which have pickup and delivery locations in common. They also showed, for a given set of origins and destinations, that the greatest benefits are observed when the load size is greater than half the capacity of the vehicle. A variant of the problem addressed in [47] can be found in [62], with additional time-window constraints. This work describes an algorithm that inserts shipments into vehicles using multiple-insertion heuristics for static and real-time test cases.

In [48], an empirical analysis of the heuristic presented in [47] was performed. The authors noted that when demands are between 51% and 60% of the capacity of the vehicle, up to 30% transportation costs can be saved. The potential savings due to split loads also depends on the percentage of loads to be collected or delivered in a common location, and the average distance from an origin to a destination relative to the distance from origin to origin and destination to destination.

For the first time the PDPSL with multiple vehicles and distance constraints was considered in [23], hence formally defining the MPDPSL. The authors developed a heuristic based on tabu search and simulated annealing. The initial solution is built using a variant of the savings algorithm by [19], and then improved by local searches based on swap and insert/split neighborhoods. The simulated annealing is then used in combination with a tabu list to control move acceptance. Experiments were conducted on the instances from [47], as well as adapted instances from [53]. However, since no distance limits are associated to the vehicles, it is always optimal to merge the routes into a single one, such that these instances cannot be viewed as multi-vehicle test cases.

3.4 Methodology

As noted previously, few heuristics have been designed for the MPDPSL, and these methods were only evaluated on benchmark instances that require the use of a single vehicle. This section aims to study the MPDPSL, providing not only an efficient algorithm for it, with new large neighborhoods, but also performing experiments with distance constraints.

3.4.1 General structure of the method

The proposed algorithm for the MPDPSL combines together an ILS with random variable neighborhood descent. This is done by replacing the classical local search of the ILS by a two step approach: first, a RVND which finds a local optimum with respect to several simple enumerative neighborhoods, and second, an improvement procedure which explores a large neighborhood via dynamic programming. The pseudo code of this iterated local search for the pickup and delivery problem with split loads (that we refer as IPDS), is shown in Algorithm 3.

Algorithm 3: IPDS

```
input : $T_{\text{MAX}}, p_{\text{MAX}}$
1  $s \leftarrow$ greedyInitialSolution();
2  $s \leftarrow$ RVND($s$);
3  $s \leftarrow$ RCSP_insertion($s$);
4  while $time \leq T_{\text{MAX}}$ do
5      $s' \leftarrow$ Perturbation($s$,$p_{\text{MAX}}$);
6      $s' \leftarrow$ RVND($s'$);
7      $s' \leftarrow$ RCSP_insertion($s'$);
8      if $f(s') < f(s)$ then
9          $s \leftarrow s'$ ;
10     end
11 end
12 Return $s$ ;
```

The IPDS algorithm receives as input the time limit T_{MAX} for executing the algorithm and the number p_{MAX}, which is the maximum limit of perturbations. At the beginning of the execution, a solution s is built via a greedy constructive heuristic (line 1). This solution is improved by means of the RVND (line 2) and the new large neighborhood search (line 3), that we call *RCSP insertion*. Then, iteratively, to escape from local optima, a perturbation is applied on the current solution s, generating a new solution s' (line 5), which is improved by the RVND and *RCSP insertion* (lines 6–7). The best solution s is always stored (lines 8–10) during the course of the method. This process is iterated until a termination criterion, here a time limit (line 4), and the best found solution is returned.

We now detail the components of the algorithm: the construction of the initial solution, the local and large neighborhood improvement procedures, and finally the perturbation operator. With the exception of the large neighborhood operator, all procedures are relatively simple and classic, hence leading to a well-performing algorithm which can be easily reproduced.

3.4.2 Initial solution

The initial solution s is produced by a greedy constructive heuristic. Iteratively, this heuristic computes for each pickup customer i its best insertion position, with minimum increase of distance. The pickup customer i with the shortest distance increase is inserted at each iteration, and the corresponding delivery $(n+i)$ is added in its best position after i. Note that the method seeks to insert either full deliveries or full truckloads $(\min\{q_i, C\})$, resulting in an initial solution without split loads. If no feasible insertion is found because of the distance constraints, then a new route is created.

After the construction, the initial solution is improved by the two-phase local search, including the RVND and the *RCSP insertion* procedures, which allows to introduce split loads in the solution.

3.4.3 Local search procedures

We first recall the concept of *block* [16], which is needed to describe some neighborhoods. A block B_i is defined as a sequence of consecutive visits that starts at a pickup customer i and ends at the corresponding delivery customer $(n+i)$. A block B_i is a simple block if there is no customer between i and $(n+i)$. B_i is a compound block when there is at least one block $B_j \in B_i$ such that $\Pi(i) < \Pi(j) < \Pi(n+j) < \Pi(n+i)$, where $\Pi(i)$ is the position of the customer i in the route. It is noteworthy that a compound block cannot contain a pickup customer without its corresponding delivery customer and vice versa.

3.4.4 Random Neighborhood Variable Descent

As in the random neighborhood variable descent of [59] and [60], there is no predefined order for the neighborhoods, that is, before every execution of the local search, a new neighborhood order is randomly chosen. Each neighborhood is defined relatively to one type of move, which can be applied on different p-d pairs and routes. Each neighborhood is efficiently pre-evaluated exhaustively, considering the moves in random order of p-d pairs, and applying the first improving move. After each improvement, the search restarts from the first neighborhood structure. Otherwise, the search continues on the next neighborhood structure and finishes when all the neighborhoods have been examined without success. We now define the intra-route and inter-route neighborhoods used by the IPDS algorithm, as well as a post-optimization procedure derived from an optimality condition of the MPDPSL.

3.4.4.1 Intra-route neighborhood structures:

$\boldsymbol{N^{(1)}}$ – *PairSwap* considers two pairs of customers $(i, n+i)$ and $(j, n+j)$ and swaps the pickup customer i with the pickup customer j, as well as the delivery customer $(n+i)$ with the delivery customer $(n+j)$.

$\boldsymbol{N^{(2)}}$ – *PairShift* considers a pair of customers $(i, n+i)$ and relocates the pickup customer i in a position of the interval $[\Pi(i) - \Delta, \Pi(i) + \Delta]$ and the delivery customer $(n+i)$ in a position of the interval $[\Pi(i) + 1, \Pi(i) + \Delta]$. During preliminary experiments, we observed that $\Delta = 5$ led to a good trade-off between move exploration exhaustiveness and CPU time.

$\boldsymbol{N^{(3)}}$ – *PickShift* relocates a pickup customer i in another position before the delivery customer $(n+i)$.

$\boldsymbol{N^{(4)}}$ – *DelShift* relocates a delivery customer $(n+i)$ in another position after the pickup customer i.

$\boldsymbol{N^{(5)}}$ – *BlockSwap* swaps a block B_i with another block B_j.

$\boldsymbol{N^{(6)}}$ – *BlockShift* relocates a block B_i in another position.

3.4.4.2 Inter-route neighborhood structures:

$\boldsymbol{N^{(7)}}$ – *InterPairSwap* selects a pair of customers $(i, n+i)$ from a route k_1 and another pair $(j, n+j)$ from a route k_2 and swaps the pickup customer i with the pickup customer j. The delivery customer $(n+i)$ is swapped with the delivery customer $(n+j)$.

$\boldsymbol{N^{(8)}}$ – *InterPairShift* takes a pair of customers $(i, n+i)$ from a route k_1 and transfer this pair to a route k_2. After defining $\Pi(i)$ in k_2, the delivery customer is inserted in a position of the interval $[\Pi(i) + 1, \Pi(i) + \Delta]$.

$\boldsymbol{N^{(9)}}$ – *InterBlockSwap* selects a block B_i from a route k_1 and another block B_j from a route k_2 and swaps them.

$\boldsymbol{N^{(10)}}$ – *InterBlockShift* transfers a block B_i from a route k_1 to a route k_2.

Finally, we rely on the following theorem to perform a solution post-optimization after each local search:

Theorem 1 ([23]) *If the distance matrix satisfies the triangle inequality, then there exists an optimal solution of the MPDPSL such that between each visit to a pickup customer i and its corresponding delivery $(n+i)$ no other pickups or deliveries of this same p-d pair occur.*

As such, we scan the solution and search for visits to the same p-d pair $(i, n+i)$ appearing in the order $i \rightarrow i \rightarrow n+i \rightarrow n+i$ in a route (with possible visits to other customers in-between). If this situation occurs, the two visits can be merged as one single visit while maintaining feasibility and improving the total distance. There are 2×2 possibilities for insertion of the merged p-d in place of the previous services, and the best one in terms of distance is chosen.

3.4.5 A new large neighborhood search for the MPDPSL

The improvement procedure of the previous section relies on the enumeration of many possible moves to produce improved solutions. However, we know that MPDPSL solutions can include an arbitrarily large number of visits to the same p-d pair (as illustrated in Figure 3.2). Enumerating all possible combinations of splits and placements of visits would take an exponential time. For this reason, previous methods adopted strategies which limit the number of split loads [47, 23]. To address this issue, we propose a larger (exponential-size) neighborhood, which seeks to optimize the split loads and can be efficiently explored via dynamic programming.

In the proposed neighborhood search, called *RCSP insertion*, the problem of finding the best relocation combination of pickup and delivery pairs, with possible split loads, is addressed as a resource-constrained shortest path problem (RCSP) followed by a knapsack problem. This optimization is conducted once for each p-d pair, considering the pairs in random order and applying the first improving move. For each pair x, the method works as follows.

– Remove all occurrences of the service x from all routes.

– Phase 1: For each route $\sigma = (\sigma_1, \ldots, \sigma_{n(\sigma)})$ with $n(\sigma)$ visits, evaluate the possible insertions of the p-d pair x via dynamic programming, and characterize all non-dominated trade-offs $\mathcal{S}_\sigma = \bigcup_i \{(s^d_{\sigma i}, s^q_{\sigma i})\}$ between the detour distance $s^d_{\sigma i}$ and the delivered load quantity $s^q_{\sigma i}$.

– Phase 2: Based on the known labels for each route, find the best combination of insertions in all routes in order to fulfill the total demand q_x. This selection can be performed by solving a knapsack problem.

3.4.5.1 Phase 1: Evaluation of non-dominated insertions for each route.

This problem is assimilated to a resource-constrained shortest path problem in an acyclic directed graph $H = (V', A)$, illustrated in Figure 3.3. The set V' is divided into two groups of nodes: $V' = V_{\text{ROUTE}} \cup V_{\text{INSERT}}$, where $V_{\text{ROUTE}} = \{v_1, \ldots, v_{n(\sigma)}\}$ contains one node per visit in the route σ, and the set $V_{\text{INSERT}} = \{v_1^{\text{P}}, v_1^{\text{D}}, \ldots, v_{n(\sigma)-1}^{\text{P}}, v_{n(\sigma)-1}^{\text{D}}\}$ contains a copy of the pickup vertex v_i^{P} and a copy of the delivery vertex v_i^{D} between each successive pair of vertices (v_i, v_{i+1}) in V_{ROUTE}, for a total of $3 \times n(\sigma) - 2$ nodes in V'.

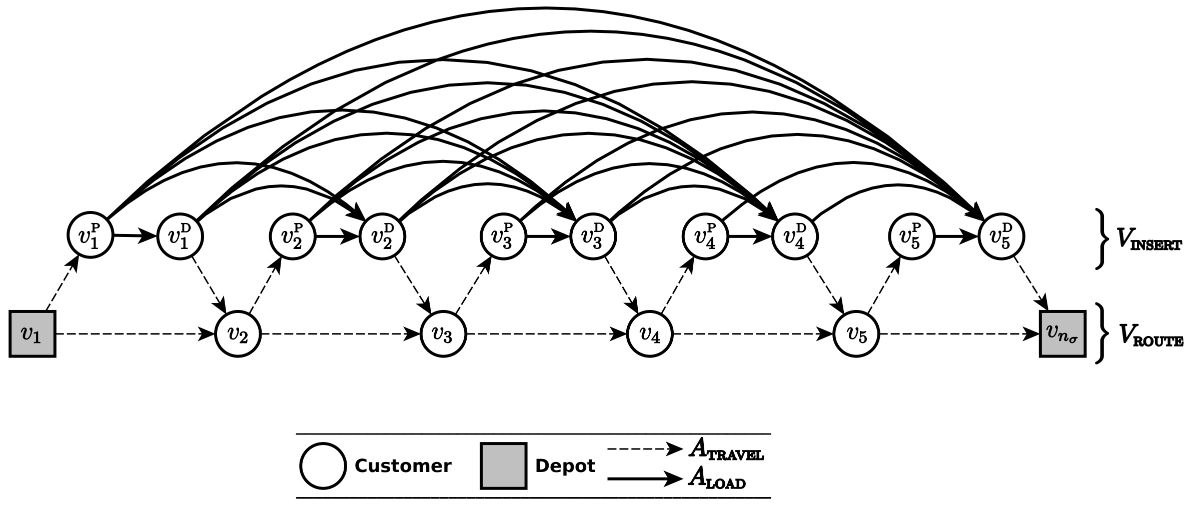

Figure 3.3: Auxiliary graph H for a route containing $n(\sigma) = 6$ visits

The arc set A is also divided into two sets $A = A_{\text{TRAVEL}} \cup A_{\text{LOAD}}$. Each arc is characterized by a detour distance δ_a^{DIST} and a delivered load δ_a^{LOAD}. The arcs in A_{TRAVEL} (dashed arrows in Figure 3.3) can either connect successive visits in V_{ROUTE}, or connect a visit v_i with its candidate pickup v_i^{P}, or connect a candidate delivery v_i^{D} with the next visit v_{i+1}. Each such arc a represents a pure vehicle relocation without any load destined for customer x, and thus $\delta_a^{\text{LOAD}} = 0$. On the other hand, the arcs in A_{LOAD} (solid arrows on the figure) correspond to trips which carry some load of x. The following cases can be distinguished.

- **Direct arc:** $a = (v_i^{\text{P}}, v_i^{\text{D}})$. This arc corresponds to a direct travel between the pickup and delivery locations of x, characterized by $\delta_{(v_i^{\text{P}}, v_j^{\text{D}})}^{\text{DIST}} = d_{x,n+x}$ and $\delta_{(v_i^{\text{P}}, v_j^{\text{D}})}^{\text{LOAD}} = C - \sum_{k=1}^{i} q_{\sigma_k}$.
- **Indirect pickup–delivery arc:** $a = (v_i^{\text{P}}, v_j^{\text{D}})$ with $i < j$. This arc corresponds to a detour starting at the pickup location of x, visiting the locations $(\sigma_{i+1}, \sigma_{i+2}, \ldots, \sigma_j)$,

and ending at the delivery location of x. As such,

$$\begin{cases} \delta^{\text{DIST}}_{(v_i^{\text{P}},v_j^{\text{D}})} = d_{x,\sigma_{i+1}} + d_{\sigma_j,x}, \text{ and} \\ \delta^{\text{LOAD}}_{(v_i^{\text{P}},v_j^{\text{D}})} = C - \max_{l\in\{i,\ldots,j\}} \sum_{k=1}^{l} q_{\sigma_k}. \end{cases}$$

– **Indirect delivery-delivery arc:** $a = (v_i^{\text{D}}, v_j^{\text{D}})$ with $i < j$. This arc corresponds to a travel starting at the delivery location of x, returning to the pickup location of x, visiting the locations $(\sigma_{i+1}, \sigma_{i+2}, \ldots, \sigma_j)$, and ending at the delivery location x. As such,

$$\begin{cases} \delta^{\text{DIST}}_{(v_i^{\text{P}},v_j^{\text{D}})} = d_{n+x,x} + d_{x,\sigma_{i+1}} + d_{\sigma_j,x}, \text{ and} \\ \delta^{\text{LOAD}}_{(v_i^{\text{P}},v_j^{\text{D}})} = C - \max_{l\in\{i,\ldots,j\}} \sum_{k=1}^{l} q_{\sigma_k}. \end{cases}$$

After the construction of the graph H, the resource constrained shortest path problem is solved by a simple labeling algorithm, which computes for each vertex v a set of labels $S_v = \{s_{vi} \mid i \in \{1, \ldots, |S_v|\}\}$, where each label $s_{vi} = (s^{\text{DIST}}_{vi}, s^{\text{LOAD}}_{vi})$ is characterized by its detour distance and its total delivery quantity. Starting at the depot with $\mathcal{S}_{v_1} = \{(0,0)\}$, the labels are iteratively propagated in topological order of the nodes as follows:

$$\begin{aligned} &\textbf{for } v \in (v_1^{\text{P}}, v_1^{\text{D}}, v_2, v_2^{\text{P}}, v_2^{\text{D}}, \ldots, v_{n(\sigma)}), \\ &\mathcal{S}_v = \bigcup_{w|(w,v)\in A} \bigcup_{s_{wi}\in\mathcal{S}_w} \{(s^{\text{DIST}}_{wi} + \delta^{\text{DIST}}_{(w,v)}, s^{\text{LOAD}}_{wi} + \delta^{\text{LOAD}}_{(w,v)})\}. \end{aligned} \tag{3.44}$$

At each step of the construction, non-dominated labels are eliminated from the set S_v. A label s_{vi} is dominated by a label s_{vj} if $s^{\text{DIST}}_{vi} \geq s^{\text{DIST}}_{vj}$, and $\min\{s^{\text{LOAD}}_{vi}, q_x\} \leq \min\{s^{\text{LOAD}}_{vj}, q_x\}$. Moreover, a completion bound is used to eliminate additional labels. Indeed, any label s_{v_ij} of a node $v_i \in V_{\text{ROUTE}}$ which fulfills all needed demand, i.e., such that $s^{\text{LOAD}}_{v_ij} \geq q_x$, leads to a distance bound of $s^{\text{DIST}}_{v_ij}$. Any label which exceeds the best known distance bound can be pruned from the search.

At the end of the labeling procedures, and for each route σ, the set of non-dominated labels $S(\sigma) = S_{v_{n(\sigma)}}$ is stored. For single-vehicle problem instances, the best insertion of visits to the service pair x corresponds to the only non-dominated label $s \in S(\sigma)$ such that $s^{\text{LOAD}} \geq q_x$. In the case with multiple vehicles, the best re-insertion of visits for the pair x can involve multiple routes. As described in the following paragraph, the best combination of insertions can be found by solving a knapsack problem based on the labels $S(\sigma)$ for each route σ.

3.4.5.2 Phase 2: Combination of insertions in multiple vehicles.

In the presence of multiple vehicles, the algorithm searches for a good combination of insertions in different routes in order to cover the total demand. This problem can be formulated as a knapsack problem with an additional conflict constraint that limits the selection to one label at most in each route. For each label $s_{\sigma j} \in S(\sigma)$, for each route σ, we associate a binary decision variable $y_{\sigma j}$ which equals 1 if and only if the label $s_{\sigma j}$ is selected. The resulting problem is formulated in Equations (3.45–3.48).

$$\min \sum_{\sigma \in \mathcal{R}} \sum_{s_{\sigma j} \in S(\sigma)} s_{\sigma j}^{\text{DIST}} y_{\sigma j} \tag{3.45}$$

$$\sum_{\sigma \in \mathcal{R}} \sum_{s_{\sigma j} \in S(\sigma)} s_{\sigma j}^{\text{LOAD}} y_{\sigma j} \geq q_x \tag{3.46}$$

$$\sum_{s_{\sigma j} \in S(\sigma)} y_{\sigma j} \leq 1 \qquad \sigma \in \mathcal{R} \tag{3.47}$$

$$y_{\sigma j} \in \{0, 1\} \qquad \sigma \in \mathcal{R},\ s_{\sigma j} \in S(\sigma) \tag{3.48}$$

To solve this problem, we tested different exact techniques, either based on dynamic programming or integer programming. In our experiments, these methods led to a significant computational-time overhead. We thus opted for a heuristic resolution (as in [13] for the SDVRP), using a simple greedy heuristic which iteratively selects the label $s_{\sigma j}$ with maximum ratio $s_{\sigma j}^{\text{LOAD}} / s_{\sigma j}^{\text{DETOUR}}$, and matches in most cases the optimal result. The associated new MPDPSL solution is accepted if it improves the distance of the current solution.

3.4.6 Perturbation operator

The last component of the IPDS algorithm, the perturbation operator, is designed to escape from the local minima of the previous neighborhood improvement procedures. It consists of relocating n_{PERT} random p-d pairs from their original routes to new random positions, inserting both pickup and deliveries consecutively. The number of pairs n_{PERT} to be relocated, which determines the strength of the perturbation, is randomly selected in $\{1, 2, ..., p_{\text{MAX}}\}$ with uniform distribution. As such, p_{MAX} is a method parameter which establishes a maximum limit on the impact of the perturbation.

3.5 Computational results

Our computational experiments have been conducted on the two existing sets of PDPSL instances from previous literature, as well as new MPDPSL instances. The first set originates from [47], and the second from [23]. All these instances were generated in a way that each load occupies 51% to 60% of the capacity of the vehicle. In this setting, the savings related to split loads tend to be the greatest [47].

The first set of instances was randomly generated by [47]. It contains three subsets of 15 instances each. These subsets have 75, 100 and 125 pickup and delivery pairs. In each instance, the pickups can occur in five different locations, and each subset has a different number of delivery locations: 15, 20 and 25 delivery locations, respectively. The second set of instances was randomly generated in [23], adapting the set of instances from [53], leading to four subsets of 12 instances each, with 50, 100, 250 and 500 pickup and delivery pairs. In this set, every visit location is randomly generated.

The IPDS algorithm was developed in C++ using OptFrame [20], a computational framework for the development of efficient heuristic algorithms for combinatorial optimization problems. Each test was executed on a single core of a Intel Core 2 Quad 2.4 GHz, 4 GB of RAM and in Ubuntu 14.04. It is important to highlight that the computer used is very similar to the one used in [23]. Furthermore, the IPDS uses only two main parameters: the strength of the perturbation operator p_{MAX}, which has been set to 3 (as this value provided a robust and good parameter setting in our preliminary experiments) and the stopping criterion T_{MAX}, which has been set for each instance group to the CPU time of the current state-of-the-art method of [23].

Depending on the instance set, previous authors have either reported results on a single run, or best results over multiple runs. Both measures tend to be influenced by the variance of the performance of an algorithm. We thus opted to report the average solution quality over several runs, which is a better estimate of the average behavior of an algorithm. For each instance, we obtain a value for the solution using IPDS, z_{IPDS}, and compute (Eq. 3.49) the percentage gap relative to the value of the best known solution (BKS), z_{BKS}, collected for each instance from [47] and [23].

$$Gap = 100 \times (z_{\text{IPDS}} - z_{\text{BKS}})/z_{\text{BKS}} \tag{3.49}$$

The objective in the MPDPSL is to minimize the total travel costs, hence if Eq. (3.49) returns a positive gap, it means that this solution has a greater value compared to the best known solution, therefore it is a worst solution in terms of quality. Moreover, if a negative value is returned, then the solution obtained has a lower value than the best known solution, thus a new best known solution is found.

3.5.1 Performance comparisons on PDPSL instances

3.5.1.1 Instances from [47]

Both [47] and [23] present the solution quality of one run per instance. To provide a reliable estimate of performance, we repeated our experiments 20 times with different random seeds, and report the average solutions on each instance. The best results are also indicated to establish bounds for future research. We adopted the same time limits as [47] and [23]: 25.50 minutes per run for each instance with 75 pairs, 56.20 minutes for each instance with 100 pairs, and 95.90 minutes for each instance with 125 pairs. Tables 3.1, 3.2 and 3.3 display the results on these instances. For each instance, the result of the best method is highlighted in boldface.

Table 3.1: Results for the PDPSL with 75 pairs – Instances from [47]
Time limit set to 25.5 minutes per run

		Nowak et al.		**Şahin et al.**		**IPDS**			
Instance	**BKS**	**1-Run**	**Gap(%)**	**1-Run**	**Gap(%)**	**Avg-20**	**Gap(%)**	**Best-20**	**Gap(%)**
75_1A	3796.32	3830.12	0.89	3894.34	2.58	**3786.30**	**-0.26**	3727.32	-1.82
75_1B	3808.16	3857.12	1.29	3842.88	0.91	**3764.84**	**-1.14**	3683.20	-3.28
75_1C	3790.03	3810.50	0.54	3790.03	0.00	**3767.15**	**-0.60**	3686.44	-2.73
75_1D	3799.32	3799.32	0.00	3862.23	1.66	**3755.47**	**-1.15**	3707.17	-2.43
75_1E	3788.37	3868.96	2.13	3820.87	0.86	**3781.96**	**-0.17**	3719.38	-1.82
75_2A	3161.69	3313.48	4.80	3177.98	0.52	**3104.55**	**-1.81**	3044.34	-3.71
75_2B	3169.92	3296.36	3.99	3179.00	0.29	**3102.96**	**-2.11**	3069.97	-3.15
75_2C	3121.97	3203.25	2.60	3121.97	0.00	**3085.99**	**-1.15**	3040.38	-2.61
75_2D	3117.69	3266.42	4.77	3117.69	0.00	**3074.07**	**-1.40**	3010.40	-3.44
75_2E	3148.70	3332.59	5.84	3168.66	0.63	**3109.04**	**-1.26**	3075.34	-2.33
75_3A	3897.12	4058.37	4.14	3910.04	0.33	**3892.26**	**-0.12**	3817.39	-2.05
75_3B	3868.75	4172.42	7.85	3868.75	0.00	**3860.70**	**-0.21**	3771.77	-2.51
75_3C	3858.71	4090.65	6.01	3900.38	1.08	**3866.62**	**0.20**	3787.46	-1.85
75_3D	3845.05	4110.39	6.90	3888.20	1.12	**3850.45**	**0.14**	3762.61	-2.14
75_3E	3893.36	4052.23	4.08	3908.01	0.38	**3826.36**	**-1.72**	3733.67	-4.10
Avg			3.72		0.69		**-0.85**		-2.66
CPU		Xeon 2.4 GHz 2 GB		Intel Core 2 Quad 2.4 GHz 4 GB		Intel Core 2 Quad 2.4 GHz 4 GB			

From these experiments, IPDS appears to produce solutions of higher quality than the methods of Nowak and Şahin, as it was able to find better average results on all 45 instances. The best performance appears to be achieved for larger data sets. Considering

Table 3.2: Results for the PDPSL with 100 pairs – Instances from [47]
Time limit set to 56.2 minutes per run

		Nowak et al.		Şahin et al.		IPDS			
Instance	**BKS**	**1-Run**	**Gap(%)**	**1-Run**	**Gap(%)**	**Avg-20**	**Gap(%)**	**Best-20**	**Gap(%)**
100_1A	4992.59	5073.40	1.62	4992.59	0.00	**4886.80**	**-2.12**	4823.76	-3.38
100_1B	5036.55	5036.55	0.00	5042.30	0.11	**4921.42**	**-2.29**	4861.49	-3.48
100_1C	5015.09	5029.38	0.29	5015.09	0.00	**4922.82**	**-1.84**	4813.72	-4.02
100_1D	4996.08	5012.97	0.34	4996.08	0.00	**4922.44**	**-1.47**	4831.00	-3.30
100_1E	5015.26	5130.15	2.29	5015.26	0.00	**4896.29**	**-2.37**	4792.94	-4.43
100_2A	4204.28	4450.06	5.85	4258.49	1.29	**4169.72**	**-0.82**	4096.25	-2.57
100_2B	4306.73	4484.47	4.13	4306.73	0.00	**4225.95**	**-1.88**	4156.74	-3.48
100_2C	4215.07	4473.39	6.13	4259.09	1.04	**4201.15**	**-0.33**	4134.60	-1.91
100_2D	4244.77	4424.57	4.24	4267.37	0.53	**4194.76**	**-1.18**	4089.79	-3.65
100_2E	4228.82	4559.26	7.81	4228.82	0.00	**4200.25**	**-0.68**	4132.97	-2.27
100_3A	5126.71	5294.37	3.27	5126.71	0.00	**4987.04**	**-2.72**	4934.62	-3.75
100_3B	5084.70	5371.74	5.65	5161.29	1.51	**5042.60**	**-0.83**	4974.61	-2.17
100_3C	5075.45	5216.80	2.78	5098.71	0.46	**5004.95**	**-1.39**	4938.02	-2.71
100_3D	5106.32	5467.79	7.08	5106.32	0.00	**5010.16**	**-1.88**	4941.18	-3.23
100_3E	5076.14	5572.47	9.78	5076.14	0.00	**5029.86**	**-0.91**	4884.24	-3.78
Avg			4.08		0.33		**-1.51**		-3.21
CPU		Xeon 2.4 GHz 2 GB		Intel Core 2 Quad 2.4 GHz 4 GB		Intel Core 2 Quad 2.4 GHz 4 GB			

Table 3.3: Results for the PDPSL with 125 pairs – Instances from [47]
Time limit set to 95.9 minutes per run

		Nowak et al.		Şahin et al.		IPDS			
Instance	**BKS**	**1-Run**	**Gap(%)**	**1-Run**	**Gap(%)**	**Avg-20**	**Gap(%)**	**Best-20**	**Gap(%)**
125_1A	5950.44	6020.05	1.17	6002.15	0.87	**5762.68**	**-3.16**	5682.79	-4.50
125_1B	5938.94	5938.94	0.00	5998.06	1.00	**5785.19**	**-2.59**	5678.06	-4.39
125_1C	5933.69	5977.69	0.74	5933.69	0.00	**5758.32**	**-2.96**	5625.24	-5.20
125_1D	6060.85	6138.94	1.29	6083.59	0.38	**5802.33**	**-4.27**	5701.05	-5.94
125_1E	5906.34	6024.26	2.00	5906.34	0.00	**5755.05**	**-2.56**	5660.45	-4.16
125_2A	5396.85	5717.54	5.94	5444.23	0.88	**5262.65**	**-2.49**	5183.40	-3.96
125_2B	5456.91	5745.38	5.29	5460.81	0.07	**5313.86**	**-2.62**	5209.02	-4.54
125_2C	5412.81	5667.26	4.70	5412.81	0.00	**5289.23**	**-2.28**	5145.39	-4.94
125_2D	5475.40	5778.58	5.54	5494.71	0.35	**5321.00**	**-2.82**	5234.01	-4.41
125_2E	5419.02	5780.01	6.66	5419.02	0.00	**5281.44**	**-2.54**	5191.63	-4.20
125_3A	6237.20	6934.05	11.17	6252.24	0.24	**6128.28**	**-1.75**	6050.78	-2.99
125_3B	6300.04	6918.16	9.81	6300.04	0.00	**6152.84**	**-2.34**	6057.74	-3.85
125_3C	6324.66	6607.30	4.47	6332.93	0.13	**6129.45**	**-3.09**	6024.87	-4.74
125_3D	6317.05	7239.79	14.61	6359.16	0.67	**6166.94**	**-2.38**	6040.13	-4.38
125_3E	6257.16	6776.37	8.30	6277.38	0.32	**6137.54**	**-1.91**	6057.75	-3.19
Avg			5.45		0.33		**-2.65**		-4.36
CPU		Xeon 2.4 GHz 2 GB		Intel Core 2 Quad 2.4 GHz 4 GB		Intel Core 2 Quad 2.4 GHz 4 GB			

the average gap for each set of instances, we observe negative gaps on every set (-0.85%, -1.51% and -2.65%), which represent an average improvement over the best known solutions in the literature. Considering the best results out of 20 runs, we observe a significant improvement of the previous BKS, finding 45 new best solutions with 3.41% of improvement, in average.

To validate the statistical significance of these results, we conducted a Friedman test comparing the solution values for each instance. This test led to a value $p < 2.2 \times 10^{-16}$, which indicates a significant difference of performance. We also performed pairwise Wilcoxon tests to locate these differences which, as reported in Table 3.4, support the existence of significant differences between all three methods: IPDS is significantly better than Şahin's algorithm, which is in turn significantly better than Nowak's algorithm.

Table 3.4: Results of pairwise Wilcoxon tests – Instances from [47]

Algorithms	**p-value**
Şahin–Nowak	2.12×10^{-10}
IPDS–Nowak	5.68×10^{-14}
IPDS–Şahin	5.68×10^{-14}

3.5.1.2 Instances from [23]

A second set of instances was introduced in [23] and the authors also presented the best solutions found after 20 runs for each instance with 50, 100 and 250 p-d pairs. For the instances with 500 pairs, the authors presented the best solutions over 5 runs. As indicated by the authors in a private communication, the reported time values in [23] correspond to the average time of one run. These values also depend on the specific instance. As such, we have defined for each group of instances a termination criterion T_{MAX} which is smaller or equal to the average CPU time of [23]: 5 seconds for the instances with 50 service pairs, 40 seconds for the instances with 100 pairs, 5 minutes for the instances with 250 pairs, and 1 hour for the instances with 500 pairs. Tables 3.5–3.8 display the results of these experiments. In these tables, the solution quality of the best method is highlighted in boldface.

Since [23] measure the best solution quality over several restarts (20 or 5), the comparison is established with the best solution of IPDS over the same number of runs. When analyzing the tables, we observe that IPDS produces best solutions of higher quality (better than the BKS) on 47 instances out of 48. The significance of these improvements is again confirmed by a pairwise Wilcoxon test with a value $p = 2.35 \times 10^{-13}$. The magnitude

Table 3.5: Results for the PDPSL with 50 pairs – Instances from [23]
Time limit set to 5 seconds per run

	Şahin et al.		**IPDS**			
Instance	**Time(s)**	**Best-20**	**Best-20**	**Gap(%)**	**Avg-20**	**Gap(%)**
50A	4.5	16791.20	**15481.36**	**-7.80**	15913.01	-5.23
50B	4.3	17115.50	**15422.03**	**-9.89**	15814.13	-7.60
50C	4.5	14956.00	**14131.43**	**-5.51**	14591.86	-2.43
50D	4.3	16290.00	**14947.06**	**-8.24**	15345.82	-5.80
50E	7.6	11397.50	**9517.49**	**-16.49**	9895.06	-13.18
50F	7.4	9532.59	**8429.16**	**-11.58**	8927.89	-6.34
50G	6.2	9665.06	**8820.07**	**-8.74**	9175.39	-5.07
50H	11.2	9199.58	**7608.63**	**-17.29**	7930.69	-13.79
50I	5.8	14469.40	**12864.70**	**-11.09**	13235.73	-8.53
50J	6.5	13200.20	**11891.39**	**-9.92**	12131.98	-8.09
50K	2.3	12759.30	**12337.42**	**-3.31**	12594.40	-1.29
50L	4.4	14867.80	**13426.21**	**-9.70**	13973.27	-6.02
Avg	5.7			**-9.96**		-6.95

Table 3.6: Results for the PDPSL with 100 pairs – Instances from [23]
Time limit set to 40 seconds per run

	Şahin et al.		**IPDS**			
Instance	**Time(s)**	**Best-20**	**Best-20**	**Gap(%)**	**Avg-20**	**Gap(%)**
100A	25.1	27301.2	**25398.19**	**-6.97**	26268.75	-3.78
100B	19.4	27090.1	**25027.88**	**-7.61**	26020.76	-3.95
100C	34.0	27221.3	**25319.76**	**-6.99**	25833.95	-5.10
100D	19.0	28574.7	**26110.15**	**-8.62**	27177.34	-4.89
100E	74.7	15320	**13498.17**	**-11.89**	14022.84	-8.47
100F	95.8	17574.2	**13548.03**	**-22.91**	13919.54	-20.80
100G	50.1	14888.4	**14508.21**	**-2.55**	15062.04	1.17
100H	57.9	16259.7	**14445.99**	**-11.15**	15021.09	-7.62
100I	32.4	24994.4	**22603.21**	**-9.57**	23292.98	-6.81
100J	37.5	23025.5	**21284.65**	**-7.56**	21843.80	-5.13
100K	30.4	24509	**22435.89**	**-8.46**	23248.32	-5.14
100L	49.3	23994.7	**20705.86**	**-13.71**	21400.98	-10.81
Avg	43.8			**-9.83**		-6.78

Table 3.7: Results for the PDPSL with 250 pairs – Instances from [23]
Time limit set to 5 minutes per run

	Şahin et al.		**IPDS**			
Instance	**Time(s)**	**Best-20**	**Best-20**	**Gap(%)**	**Avg-20**	**Gap(%)**
250A	287.3	58847.6	**56857.45**	**-3.38**	58821.37	-0.04
250B	253.0	57559.1	**55871.66**	**-2.93**	57637.00	0.14
250C	299.5	57495.9	**56483.36**	**-1.76**	58107.41	1.06
250D	356.1	59396.7	**57368.20**	**-3.42**	59438.78	0.07
250E	3174.6	31736.8	**28327.20**	**-10.74**	29454.09	-7.19
250F	1123.1	27596	**24820.19**	**-10.06**	25562.37	-7.37
250G	1089.3	29421.8	**26552.49**	**-9.75**	27549.93	-6.36
250H	939.5	31911.5	**27326.55**	**-14.37**	28656.36	-10.20
250I	468.2	50154.8	**48124.77**	**-4.05**	50165.75	0.02
250J	448.8	53636.2	**51119.88**	**-4.69**	52868.55	-1.43
250K	537.5	50084.4	**46946.17**	**-6.27**	49128.23	-1.91
250L	392.3	54393.4	**52580.00**	**-3.33**	55067.03	1.24
Avg	780.8			**-6.23**		-2.66

Table 3.8: Results for the PDPSL with 500 pairs – Instances from [23]
Time limit set to 1 hour per run

	Şahin et al.		**IPDS**			
Instance	**Time(s)**	**Best-5**	**Best-5**	**Gap(%)**	**Avg-5**	**Gap(%)**
500A	2124.6	106674	**105536.28**	**-1.07**	107176.11	0.47
500B	2374.2	110881	**107657.18**	**-2.91**	109636.75	-1.12
500C	1985.8	109181	**107676.77**	**-1.38**	109991.16	0.74
500D	2247.0	109746	**104432.98**	**-4.84**	107220.16	-2.30
500E	10860.4	63068.4	**62322.13**	**-1.18**	63411.06	0.54
500F	10815.8	68829.7	**62951.49**	**-8.54**	64701.06	-6.00
500G	11101.8	70038.8	**67147.93**	**-4.13**	69658.51	-0.54
500H	16763.8	60568.5	**60489.14**	**-0.13**	62636.44	3.41
500I	5075.4	93178.2	94264.57	1.17	97404.34	4.54
500J	4698.0	96984.8	**94512.62**	**-2.55**	97141.73	0.16
500K	4539.6	97429.5	**96717.63**	**-0.73**	98134.65	0.72
500L	5996.2	98102.7	**95634.88**	**-2.52**	97539.59	-0.57
Avg	6548.6			**-2.40**		0.00

of these improvements is also larger than previously, with an improvement of 7.11% in average (comparing best solutions together), which seems to indicate that these instances with a wider diversity of possible pickup and delivery locations are more difficult to solve, and remain challenging for future works.

3.5.2 Sensitivity analysis on the components of the method

In order to examine the relative role of each component of the proposed algorithm, we started from the standard version of the algorithm and generated some alternative configurations by removing, in turn, a different neighborhood:

Base – The standard configuration, with all local-search neighborhoods and the *RCSP insertion*;

$\boldsymbol{WN_1}$ – Base configuration without the *PairSwap* neighborhood;

$\boldsymbol{WN_2}$ – Base configuration without the *PairShift* neighborhood;

$\boldsymbol{WN_{34}}$ – Base configuration without the *PickShift* and *DelShift* neighborhoods;

$\boldsymbol{WN_5}$ – Base configuration without the *BlockSwap* neighborhood;

$\boldsymbol{WN_6}$ – Base configuration without the *BlockShift* neighborhood;

$\boldsymbol{WR}$ – Base configuration, but without the *RCSP insertion* neighborhood. We note that the removal of the *RCSP insertion* neighborhood forces the algorithm to work on a classic pickup and delivery problem, without possible split moves.

The resulting algorithms have been all tested on the instances of [47], performing five runs for each of the 45 data sets, and using the same termination criterion as in Section 3.5.1. Table 3.9 displays, for each variant of the algorithm, the average gap for each set of instances (Gap-75, Gap-100 and Gap-125) as well as the average gap overall (Avg).

Table 3.9: Results for each configuration of the IPDS – Instances from [47]

Configuration	**Gap-75 (%)**	**Gap-100 (%)**	**Gap-125 (%)**	**Avg (%)**
Base	**-1.07**	**-1.82**	-2.57	**-1.82**
$\boldsymbol{WN_1}$	-0.25	-0.93	-1.37	-0.85
$\boldsymbol{WN_2}$	-0.98	-1.58	-2.52	-1.70
$\boldsymbol{WN_{34}}$	-0.75	-1.55	-2.46	-1.59
$\boldsymbol{WN_5}$	-0.98	-1.74	**-2.67**	-1.80
$\boldsymbol{WN_6}$	-0.80	-1.45	-1.95	-1.40
$\boldsymbol{WR}$	48.84	48.93	47.18	48.32

In this table, we observe that the Base configuration leads to the best overall average gap (-1.82%), as well as the best average gaps on the 75-pairs and 100-pairs instances. Still, the best average gap on the 125-pairs instances is attributed to the WN_5 variant, without the *BlockSwap* neighborhood. This effect is possibly due to the variance of the solution quality of the algorithm on this relatively small sample of 15 instances, but it also demonstrates that some neighborhoods have a much larger impact than others. In decreasing order of importance, the most important neighborhood is the proposed *RCSP insertion*, followed by the *PairSwap* neighborhood, the *BlockShift*, *PickShift* and *DelShift* neighborhoods, and then the others. The *RCSP insertion*, in our context, is essential since it manages the optimization of the split loads.

The gaps obtained by all algorithm variants (on all runs) can also be better observed by means of box plots, as in Figure 3.4. In these box plots, represented without the results of WR so as to enhance readability, we can observe the general superiority of the Base configuration, producing approximately 50% of solutions with gaps below -2%. The removal of *PairSwap* (WN_1) has a large negative impact on the final solutions, followed by the removal of *BlockShift* (WN_6), the removal of *PickShift* and *DelShift* (WN_{34}), the removal of *PairShift* (WN_2) and the removal of *BlockSwap* (WN_5).

To validate these observations, we performed a Friedman test based on the gap values of each algorithm. The test led to a value $p < 2.2 \times 10^{-16}$, demonstrating significant statistical differences. Then, we performed paired-sample Wilcoxon tests to compare the Base algorithm with all other algorithms. The results of these tests are reported in Table 3.10.

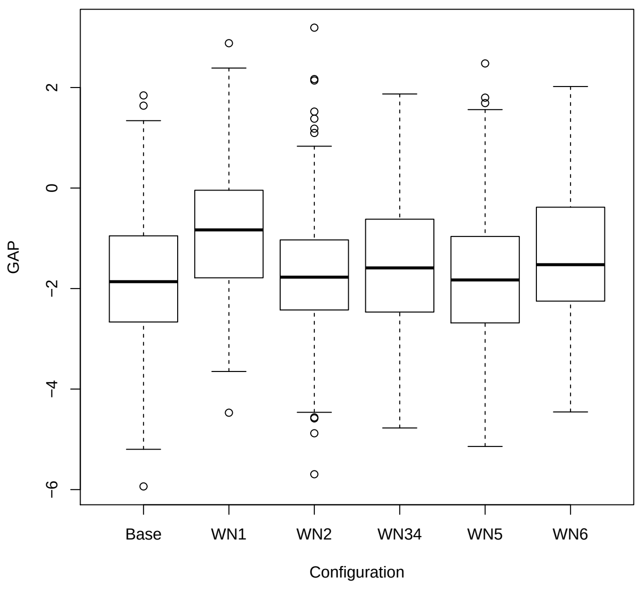

Figure 3.4: Box plot showing the gaps for each configuration of the IPDS

Table 3.10: Results from paired-sample Wilcoxon tests with the Base algorithm

Algorithms	**p-value**
Base – $\boldsymbol{WN_1}$	3.26×10^{-16}
Base – $\boldsymbol{WN_2}$	0.18
Base – $\boldsymbol{WN_{34}}$	0.01
Base – $\boldsymbol{WN_5}$	0.78
Base – $\boldsymbol{WN_6}$	5.05×10^{-05}
Base – $\boldsymbol{WR}$	$< 2.2 \times 10^{-16}$

These results confirm, with high confidence, the hypotheses that the Base algorithm produces results of significantly (better) quality than the WN_1, WN_{34}, WN_6 and WR configurations, with p-values which are always smaller than a threshold of 0.05. This highlights the importance of the associated neighborhoods which were deactivated in these configurations. A pairwise Wilcoxon test between the Base configuration and WN_2 and WN_5 led to p-values of 0.18 and 0.78, such that the significance of the difference of performance is not established in these cases. We can still reasonably conjecture that the associated neighborhoods (*PairShift* and *BlockSwap*) also have some impact, which would be better visible with additional runs and/or test instances. Besides, the CPU time consumption dedicated to these neighborhoods is very minor, hence our choice to maintain them in the Base algorithm.

3.5.3 Results with multiple vehicles and distance constraints

As discussed in Section 3.2, the absence of distance constraints in the classical PDPSL instances leads to solutions with a single vehicle, since merging routes together is always feasible and profitable. To investigate MPDPSL test cases, we extended the 45 existing instances from [47] with a distance constraint, set to $T = 1000$, and measured the performance of IPDS algorithm on the resulting data sets. The algorithm was executed 20 times with different seeds for each instance, and the time limits used in these tests are the same as in Section 3.5.1. Since no results from previous literature are available on these instances, the percentage gap was measured relatively to the best solution found in the 20 executions. Tables 3.11, 3.12 and 3.13 display the results of these experiments: the average solution in the 20 executions (Avg-20), the average gap (Gap), the best solution in the 20 executions (Best-20), and the standard deviation related to the gaps (Std Dev) for each instance.

These results aim to provide a useful base for comparisons with future algorithms. They also reflect the difficulty of the problems, since small standard deviations and small gaps related to the best solution of all runs are usually good indications of performance. From these experiments, it appears that the average gaps and their standard deviations remain moderate, hence illustrating the good performance of the method: 2.36% gap (and standard deviation of $\sigma = 1.35\%$) for the 75-pairs set, 2.03% gap ($\sigma = 1.17\%$) for the 100-pairs set and 2.10% gap ($\sigma = 1.10\%$) for the 125-pairs set. These values are slightly higher than for the single-vehicle experiments, with 1.86% gap ($\sigma = 1.08\%$) for the 75-pairs set, 1.75% gap ($\sigma = 0.97\%$) for the 100-pairs set and 1.79% gap ($\sigma = 0.97\%$) for

Table 3.11: Results for the MPDPSL with 75 pairs – Instances from [47]
Time limit set to 25.50 mins per run

Instance	Best-20	Avg-20	Gap (%)	Std Dev (%)
75_1A	3782.93	3865.12	2.17	1.66
75_1B	3747.57	3836.27	2.37	1.62
75_1C	3793.62	3864.07	1.86	1.17
75_1D	3765.05	3835.80	1.88	1.15
75_1E	3757.36	3835.99	2.09	1.44
75_2A	3097.22	3200.18	3.32	1.23
75_2B	3123.89	3181.35	1.84	1.04
75_2C	3110.82	3174.16	2.04	1.19
75_2D	3097.16	3163.17	2.13	1.45
75_2E	3118.38	3192.46	2.38	1.59
75_3A	3866.08	3981.00	2.97	1.32
75_3B	3855.72	3976.12	3.12	1.61
75_3C	3886.66	3959.61	1.88	1.07
75_3D	3870.89	3944.61	1.90	1.16
75_3E	3828.10	3958.88	3.42	1.55
Avg			2.36	1.35

Table 3.12: Results for the MPDPSL with 100 pairs – Instances from [47]
Time limit set to 56.20 mins per run

Instance	Best-20	Avg-20	Gap (%)	Std Dev (%)
100_1A	4920.25	4993.39	1.49	1.02
100_1B	4940.53	5029.61	1.80	1.08
100_1C	4903.04	5010.18	2.19	1.17
100_1D	4928.88	5012.28	1.69	0.86
100_1E	4869.26	4977.90	2.23	1.05
100_2A	4212.57	4270.83	1.38	0.88
100_2B	4226.56	4304.39	1.84	1.13
100_2C	4213.68	4281.58	1.61	1.06
100_2D	4217.27	4305.97	2.10	1.27
100_2E	4186.25	4275.55	2.13	1.16
100_3A	4982.29	5131.18	2.99	1.31
100_3B	5057.84	5189.52	2.60	1.38
100_3C	5031.36	5137.47	2.11	1.38
100_3D	5049.84	5152.46	2.03	1.35
100_3E	5029.86	5144.86	2.29	1.46
Avg			2.03	1.17

Table 3.13: Results for the MPDPSL with 125 pairs – Instances from [47]
Time limit set to 95.90 mins per run

Instance	Best-20	Avg-20	Gap (%)	Std Dev (%)
125_1A	5794.79	5888.54	1.62	0.96
125_1B	5880.96	5966.92	1.46	0.91
125_1C	5738.87	5881.25	2.48	1.22
125_1D	5738.32	5945.99	3.62	1.49
125_1E	5822.20	5915.08	1.60	0.89
125_2A	5310.49	5419.77	2.06	0.87
125_2B	5361.25	5476.77	2.15	1.11
125_2C	5357.90	5417.37	1.11	0.66
125_2D	5331.69	5458.60	2.38	0.98
125_2E	5339.33	5443.11	1.94	1.18
125_3A	6177.11	6283.43	1.72	1.22
125_3B	6205.87	6343.14	2.21	1.17
125_3C	6230.02	6312.84	1.33	0.92
125_3D	6181.96	6351.26	2.74	1.45
125_3E	6128.42	6313.40	3.02	1.52
Avg			2.10	1.10

the 125-pairs set. As such, MPDPSL instances seem more challenging and would deserve further attention in the coming years.

3.6 Conclusions

In this work, we have considered the multi-vehicle one-to-one pickup and delivery problem with split loads (MPDPSL). Because of the combination of the "split deliveries" and "pickups and deliveries" vehicle routing attributes, dealing with this problem via local-search based heuristics is a challenging task. The sequencing and split deliveries decision subsets are very interdependent, such that various neighborhoods must be designed to jointly modify some of these decisions. Moreover, MPDPSL solutions themselves can involve an arbitrary large number of split loads.

To address this challenge, we proposed a conceptually simple ILS, based on classic neighborhoods for pickup-and-delivery problems. Yet, to efficiently manage the split deliveries, we introduced an additional exponential-sized neighborhood, which iteratively optimizes the pickup-and-delivery locations and splits for each service, and can be efficiently explored via pseudo-polynomial (dynamic programming-based) algorithms. The performance of the proposed method has been validated through extensive computational experiments. For the existing single-vehicle problem instances, this method outperforms previous algorithms in similar computational time, and finds new best known solutions for 92 out of 93 instances. We also proposed new multi-vehicle problem instances and solutions for future comparisons.

Overall, this work takes place in a general research line which aims at progressing towards an intelligent search and exploration of larger neighborhoods via efficient dynamic-programming techniques, in comparison with the brute-force enumeration of simpler neighborhoods. In this regard, the MPDPSL can be considered a very challenging problem.

Chapter 4

Multi-Vehicle Profitable Pickup And Delivery Problem

4.1 Introduction

The objective of the Vehicle Routing Problem (VRP) is to find the minimum total route cost for each vehicle so that all customers are served. There are several variations of the VRP, where this constraint of visiting all customers is relaxed. In [18], the authors proposed the Team Orienteering Problem (TOP), in which the objective is to find a subset of points (customers) for the team (vehicles) to visit and also find a route for each member of the team so that the time limit per member is respected and the total collected profit (associated with each point) is maximized. Later, in [5], the authors defined the Profitable Tour Problem (PTP), changing the objective to the maximization of profit minus the minimization of the total route cost. The combination of TOP, PTP and the one-to-one Pickup and Delivery Problem (PDP) inspired the authors of [28] to define the Multi-Vehicle Profitable Pickup and Delivery Problem (MVPPDP).

The MVPPDP is a multi-level optimization problem that, firstly, requires to select a subset of customers for the vehicles to visit. Secondly, these customers must be assigned to each vehicle and finally, a route must be defined for each vehicle. These routes must be established in order to maximize the total profit, which is given by the sum of all profits, obtained from serving each customer, minus the travel costs. This problem, for example, is very relevant for maritime transportation, in particular the tramp ship routing problems [45, 71], where cargo owners announce their transportation requests in a spot market and the shipping company decides which request to attend, considering profits, capacity and travel time.

As the MVPPDP is a “one-to-one pickup and delivery“ problem, it leads to large neighborhoods as the method for solving it must work with pairs of customers. In addition, an algorithm for solving MVPPDP must have an efficient customer selection procedure in order to obtain good solutions. These reasons can, partly, explain the existence of only two methods proposed for the MVPPDP, a variable neighborhood search and a guided local search, both presented in [28]. The authors claimed to obtain good results by using them on generated benchmark instances for the MVPPDP.

This chapter presents an efficient heuristic resolution for the MVPPDP, relying on large-neighborhood search with efficient exploration procedures. The MVPPDP is a very restrictive problem and the previous methods only accept feasible solutions during their search. This work aims to investigate the performance of an algorithm that allows infeasible solutions, related to the duration constraint, during its search procedure. By allowing infeasible solutions, the neighborhoods become larger and, thus, it can contribute to find better solutions, but also it can increase the computational time. To counterbalance this effect, the concept of granular local search [63] is applied to reduce this computational time on large instances.

A heuristic called IPPD based on the iterated local search (ILS) and the random variable neighborhood descent (RVND) is proposed. This algorithm explores classical neighborhoods for one-to-one pickup and delivery problems and also explores specific neighborhoods for selecting customers to include or exclude from the solution. Moreover, this algorithm performs shaking procedures in order to escape from local optima. Experimental analyses on benchmark instances are also done with the objective of investigating the effectiveness of the proposed methodology.

The main contributions of the work presented in this chapter are:

1. a simple and efficient heuristic for the MVPPDP;
2. a new strategy for dealing with infeasible solutions for the MVPPDP;
3. granular local searches for one-to-one pickup-and-delivery problems;
4. new results for benchmark instances.

4.2 Problem statement

The multi-vehicle profitable pickup and delivery problem was formulated by [28], combining characteristics from the team orienteering problem, the profitable tour problem and the pickup and delivery problem.

The MVPPDP is very similar to the MPDPSL. It contains a homogeneous fleet of capacitated vehicles, initial and final depots and also one-to-one pickup and delivery customers or requests. We assumed that the capacity of each vehicle is always larger than the demand of each request. This characteristic is called less-than-truckload (LTL) and is also assumed in the MPDPSL and in [28]. Moreover, each route of the MVPPDP must start and end at the depot, respect the distance limit, the vehicle capacity and also the precedence of each pickup over its delivery in the same route.

However, there are some differences between the MVPPDP and the MPDPSL. In the MVPPDP, when a vehicle arrives at a pickup customer, it must collect all available load and when it arrives at the corresponding delivery customer, all load must be delivered. Another difference from the MPDPSL is that each request i has an associated revenue r_i to be obtained if this request is attended. In addition, a solution for the MVPPDP may not necessarily contains all requests.

The objective of the MVPPDP is to find a set of routes that maximizes the total profit. The total profit is obtained by the sum of all revenues collected minus the sum of all travel costs.

4.2.1 Mathematical formulation

In [28], a mathematical model for the MVPPDP is presented. This model is based on [52] and [49] and uses the following notations:

4.2.1.1 Parameters

$\boldsymbol{n}$ number of pair of customers (there are n pickup customers and n delivery customers)

$\boldsymbol{m}$ number of vehicles

$\boldsymbol{P}$ set of pickup customers, $P = \{1, \dots, n\}$

$\boldsymbol{D}$ set of delivery customers, $D = \{n + 1, \dots, 2n\}$

$\boldsymbol{V}$ set of all customers, the initial depot 0 and the final depot $2n+1$, $V = P \cup D \cup \{0, 2n+1\}$

$\boldsymbol{K}$ set of available vehicles, $K = \{1, \ldots, m\}$

$\boldsymbol{r_i}$ revenue to be obtained when visiting the pair of customers i

$\boldsymbol{q_i}$ demand at customer i, $q_i > 0$ for pickup customers and $q_i < 0$ for delivery customers

$\boldsymbol{d_i}$ service duration at customer i

$\boldsymbol{c_{ij}}$ transportation cost when travelling from i to j

$\boldsymbol{t_{ij}}$ travel time between customer i and customer j

$\boldsymbol{C}$ loading capacity of a vehicle

$\boldsymbol{T}$ maximum tour time of a vehicle

4.2.1.2 Decision Variables

$\boldsymbol{x_{ijk}}$ binary decision variable equals to one if and only if arc (i, j) is used by vehicle k

$\boldsymbol{Q_{ik}}$ decision variable that contains the amount of load in vehicle k after visiting the i-th customer

$\boldsymbol{B_{ik}}$ decision variable representing the beginning of service time of vehicle k at customer i

4.2.1.3 Formulation

The mathematical model follows:

$$\max \quad \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} (r_i - c_{ij}) x_{ijk} \tag{4.1}$$

$$\sum_{i \in V} \sum_{k \in K} x_{ijk} \leq 1 \qquad \forall j \in V \tag{4.2}$$

$$\sum_{j \in V} \sum_{k \in K} x_{ijk} \leq 1 \qquad \forall i \in V \tag{4.3}$$

$$x_{i0k} = 0 \qquad \forall i \in V, \forall k \in K \tag{4.4}$$

$$x_{2n+1,jk} = 0 \qquad \forall j \in V, \forall k \in K \tag{4.5}$$

$$\sum_{i \in V}(x_{ijk} - x_{jik}) = 0 \qquad \forall j \in V \setminus \{0, 2n+1\}, \forall k \in K \tag{4.6}$$

$$\sum_{j \in V}(x_{ijk} - x_{n+i,jk}) = 0 \qquad \forall i \in P, \forall k \in K \tag{4.7}$$

$$\sum_{j \in V} x_{0jk} = \sum_{i \in V} x_{i,2n+1,k} = 1 \qquad \forall k \in K \tag{4.8}$$

$$(x_{ijk} = 1) \Rightarrow Q_{jk} = Q_{ik} + q_j \qquad \forall i \in V, \forall j \in V \setminus \{0\}, \forall k \in K \tag{4.9}$$

$$Q_{ik} \leq C \qquad \forall i \in V, \forall k \in K \tag{4.10}$$

$$Q_{0k} = 0 \qquad \forall k \in K \tag{4.11}$$

$$B_{ik} \leq B_{n+i,k} \qquad \forall i \in P, \forall k \in K \tag{4.12}$$

$$B_{jk} \geq x_{ijk}(B_{ik} + d_i + t_{ij}) \qquad \forall i \in V, \forall j \in V, \forall k \in K \tag{4.13}$$

$$B_{2n+1,k} \leq T \qquad \forall k \in K \tag{4.14}$$

$$B_{0k} = 0 \qquad \forall k \in K \tag{4.15}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i \in V, \forall j \in V, \forall k \in K \tag{4.16}$$

$$Q_{ik}, B_{ik} \geq 0 \qquad \forall i \in V, \forall k \in K \tag{4.17}$$

The objective function (4.1) maximizes the total profit, which is the total revenues collected minus the total travel costs. Each customer must be visited at most once and this is ensured by constraints (4.2) and (4.3). The vehicles must not enter into the initial depot and must not leave the final depot, this is defined by (4.4) and (4.5). The flow conservation is described by constraints (4.6). Each pickup and delivery pair must be attended by the same vehicle, as stated by (4.7). Each vehicle must start the route at the initial depot and finish at the final depot (4.8). The vehicle load is controlled by constraints (4.9), which can be transformed into linear constraints by using the big M formulation. The amount of load of a vehicle must be at most C (4.10). Vehicles leave the initial depot empty (4.11). Each pickup customer must be visited before the corresponding delivery customer (4.12). Constraints (4.13) define that the earliest beginning of service at customer j is given by the beginning of service at customer i plus the service time at i

and the travel time between i and j. The total tour length of each vehicle is bounded by constraints (4.14). Each vehicle starts at the initial depot at time 0 (4.15).

4.2.2 Illustrated example

Figure 4.1 illustrates a possible solution for an instance with ten p-d pairs and two available vehicles with capacity $C = 50$ and maximum travel distance $T = 5000$. In this instance, the set of pickup customers is $P = \{1, 2, \ldots, 10\}$ and the set of corresponding delivery customers is $D = \{11, 12, \ldots, 20\}$. The initial and final depots are at the same location, represented by 0. In this solution, *Route* $1 = \{0, 10, 20, 8, 7, 18, 4, 14, 17, 0\}$ and *Route* $2 = \{0, 9, 2, 3, 1, 11, 13, 12, 19, 0\}$. It is noteworthy that both pair of customers $(5, 15)$ and $(6, 16)$ are not supplied by any vehicle. The total distance for *Route* 1 is 4792.23 with the total revenue collected of 19101 and the total distance for *Route* 2 is 4892.25 with the total revenue collected of 27053, thus the total profit is calculated by $(19101 + 27053) - (4792.23 + 4892.25) = 36469.52$.

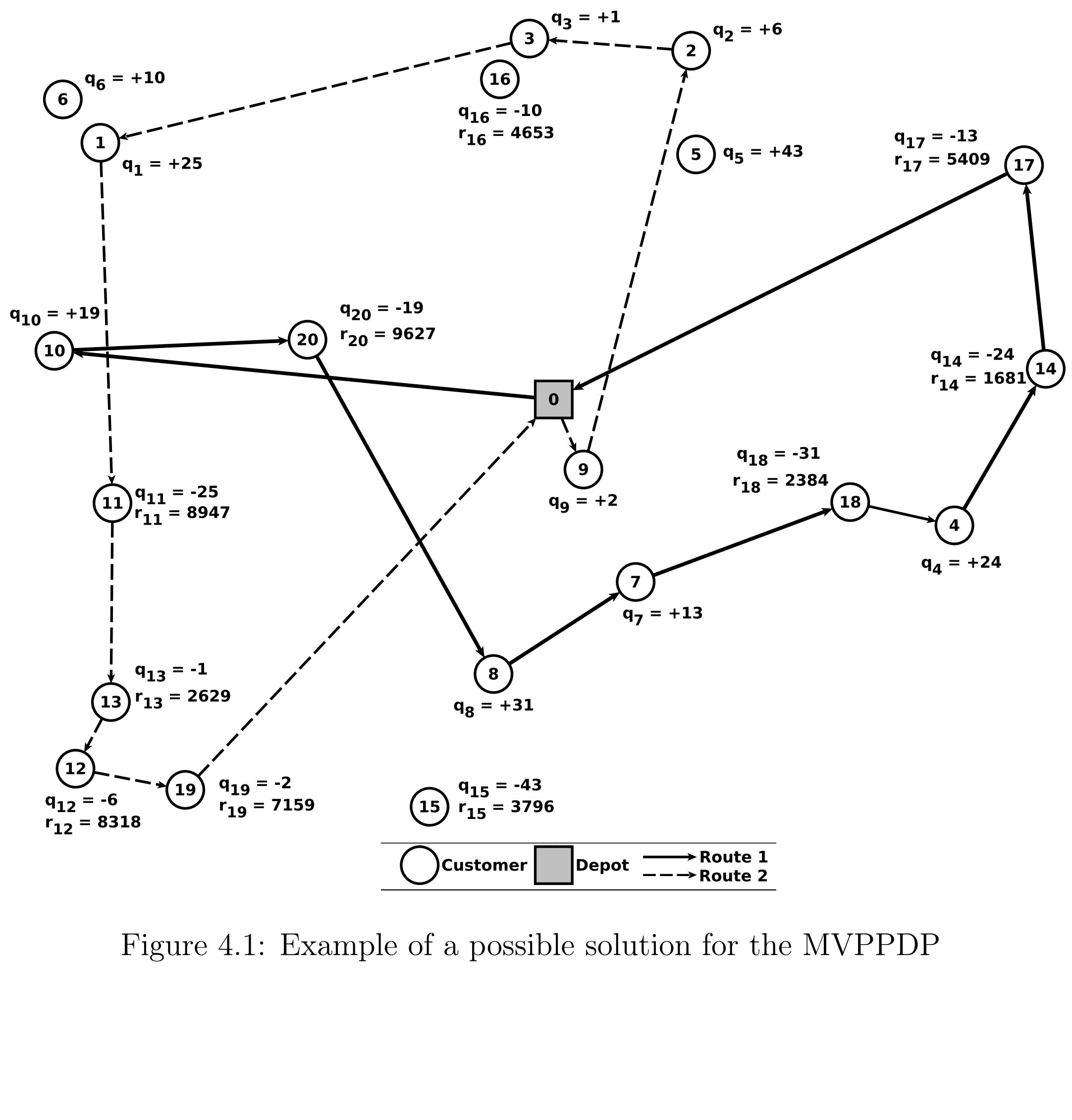

Figure 4.1: Example of a possible solution for the MVPPDP

4.3 Related literature

4.3.1 Team Orienteering Problem (TOP)

The team orienteering problem defined in [18] was proven to be in the $\mathcal{NP}$-hard class by [37] and [14]. As a consequence, many heuristics have been developed to this problem in the literature.

In [34], a LP-based granular variable neighborhood search is proposed to solve the TOP with hard Time Window constraints. The procedure uses the concept of granular local search and explores granular neighborhoods instead of complete neighborhoods to improve the efficiency of the algorithm. Granularity is based on dual information provided by the optimal solution of the LP problem. Computational results showed that the algorithm could improve 25 best known solutions.

A simulated annealing is developed in [39] and [38] for the TOP with Time Windows. The first work proposes two versions of the simulated annealing, a fast version, where the objective is to find a solution quickly and another version that concentrates on finding a better quality solution. The second work combines the simulated annealing with a multi-start hill climbing strategy.

A tabu search is proposed in [61] for the TOP. The tabu search uses an adaptive memory procedure that alternates between small an large neighborhood during the execution of the algorithm. Random and greedy strategies were used to build a solution and the algorithm also deals with infeasible solutions during its process.

Two variants of a generalized tabu search algorithm and a variable neighborhood search algorithm are used by [7] to solve the TOP. The variable neighborhood search presented better results than the two variants of the tabu search. The proposed algorithms were capable of improving 128 benchmark instances.

In [32] an ant-colony optimization algorithm is proposed to solve the TOP. Four strategies are proposed to construct the candidate solutions: sequential, deterministic-concurrent and random-concurrent, and simultaneous methods. The computational results showed that the sequential method can obtain the best solution within less than one minute for each instance.

A particle swarm optimization-based memetic algorithm is developed in [24] for solving the TOP. The method contains optimal splitting techniques and genetic crossover operators.

An iterative framework incorporating three components is developed by [30]. The first two components are a local search procedure and a simulated annealing, while the third component recombines routes to identify high quality results.

Finally, heuristics based on guided local search [65], iterated local search [66] and path relinking [58] were able to find very good solutions for the TOP in relatively short amount of time.

4.3.2 Profitable Tour Problem (PTP)

In contrast with the several works on the TOP, there are fewer studies on the profitable tour problem [9]. An approximation algorithm for the asymmetric PTP is presented by [44].

Large neighborhoods for the TOP as well as for the PTP are studied in [70]. The procedure works with standard VRP neighborhoods that works with all customers. Then, a *Select* algorithm based on resource constrained shortest paths is used to find optimal subsequence for the visits. This large neighborhood strategy was tested in a local-improvement method, an iterated local search, and a hybrid genetic search. These new neighborhoods contributed to achieve new 52 new best solutions.

Three heuristics and one exact procedures for the capacitated TOP and the capacitated PTP were proposed in [5] and [4]. The computational experiments show that the heuristic procedures often find the optimal solution for instances that can be solved to optimality by a branch-and-price algorithm.

In [31], a branch-and-cut algorithm for the capacitated PTP is presented. Valid inequalities are presented and a new family of inequalities are also presented for the capacitated PTP, called rounded multistar inequalities. The computational results indicate that the algorithm is competitive with dynamic programming algorithms.

A rich variant of the PTP is studied in [35] and a variable neighborhood search embedded with an adaptive large neighborhood search is developed.

4.3.3 Multi-Vehicle Profitable Pickup and Delivery Problem (MVPPDP)

The multi-vehicle profitable pickup and delivery problem was studied for the first time in [28]. The authors developed a heuristic based on a variable neighborhood search that performs the local searches using the variable neighborhood descent. The initial solution is built using a greedy construction heuristic based on cheapest insertion ratio (*revenue/insertion_cost*), calculated for each request and then, inserting into the partial solution the request with the highest insertion ratio. The algorithm performs local searches for minimizing the total distance travelled (*InterPairShift*, *2-opt*, *InterPairSwap*, *PairSwap*, *PairShift*, *PickShift* and *DelShift*) and for including requests that do not belong to the current solution (*Insert*, *Replace* and *GravityCenterExchange*). The shaking procedures are chosen randomly and they either remove a single request from a route or remove 10% to 40% of the route and then insert new requests based on cheapest insertion. The authors tested two VNDs, the first one executes the local searches in a sequential order (GVNSseq) and the second is based on a self-adaptive strategy that chooses the best order based on solution improvements during the execution of the algorithm (GVNSsa). The authors also developed a guided local search (GLS) based on the algorithm from [65]. Computational experiments were conducted on 36 generated instances that contain from 20 to 1000 customers. The experiments show that both variants of VNS outperform GLS in solution quality, but GLS had an advantage in terms of average runtime.

4.4 Methodology

Only two heuristics, one based on the variable neighborhood search (VNS) and the other based on the guided local search (GLS), have been developed specifically for the MVPPDP. Both algorithms explore only the feasible search space of solutions for the MVPPDP, in terms of the total tour time constraint. Because of this constraint, the search space for the MVPPDP becomes very reduced, therefore highly efficient strategies are required to reach good solutions and also to avoid getting stuck in local minima. Therefore, this work aims to investigate the exploration of a larger search space for the MVPPDP, providing an efficient algorithm for the MVPPDP that deals with solutions containing routes violating the duration constraint. The motivation of adopting this relaxation is to help the algorithm to escape from local optima, possibly reaching better solutions. In addition, as the neighborhoods become larger, the concept of granular local search is also adopted to prune many unpromising moves during the search with the aim to reduce the computational time.

4.4.1 Evaluation of a solution

A solution s for the MVPPDP is evaluated using the evaluation function $f(s)$ showed in Eq. (4.18), which is responsible for calculating the total profit. In order to obtain the total profit, the evaluation function $f(s)$ subtracts the total revenue gained by s, obtained by $p(s)$, from the total travel costs of s, given by $c(s)$. Let $d(s)$ be the total tour time exceeded by the routes and β a coefficient factor used to penalize the violation of the duration constraint.

$$f(s) = p(s) - c(s) - \beta d(s) \tag{4.18}$$

We adopted a dynamic strategy for updating the coefficient factor β during the execution of the proposed algorithm. This strategy widely used for similar problems [21, 50, 67, 51, 69]. The coefficient factor β starts with the value $\textsc{max}_{i\in V}\{r_i\}$, which represents the maximum revenue of all requests. After each group of $u_{\textsc{max}}$ iterations without improvement of the proposed algorithm, the value of β can be increased or decreased, depending on the performance of the algorithm in previous iterations. Let $n_{\textsc{infeasible}}$ be the number of all infeasible solutions s, that is if $d(s) > 0$, found by the algorithm in the last iterations and $n_{\textsc{feasible}}$ the number of all feasible solutions s, that is $d(s) = 0$, obtained in the last iterations. If more infeasible solutions were produced in the last $u_{\textsc{max}}$ iterations without any improvement, then the new value is updated to $\beta = \beta(1 + \delta)$. On the other hand, if more feasible solutions were generated then $\beta = \beta/(1 + \delta)$. The value of δ is randomly chosen in the interval $\{0.05 \ldots 0.1\}$ using an uniform distribution probability.

4.4.2 General structure of the method

The proposed algorithm for the MVPPDP combines an iterated local search with a random variable neighborhood descent, which performs, randomly, the local searches of ILS with the objective of finding a local optimum with respect to several neighborhoods. The pseudo code of the Iterated local search for the Profitable Pickup and Delivery problem (named IPPD) is shown in Algorithm 4.

The IPPD algorithm receives as input three parameters: the time limit $T_{\textsc{max}}$ for executing the algorithm, the maximum limit of perturbations $p_{\textsc{max}}$ and the maximum number of iterations without improvement $u_{\textsc{max}}$. Firstly, the value of β is defined to be the maxi-

Algorithm 4: IPPD

```
input  : $T_{\text{MAX}}, p_{\text{MAX}}, u_{\text{MAX}}$
1  $\beta \leftarrow \max_{i \in V}\{r_i\}$;
2  $s \leftarrow$ greedyInitialSolution();
3  $s \leftarrow$ RVND($s$);
4  $u \leftarrow 0$;
5  $n_{\text{INFEASIBLE}} \leftarrow 0$;
6  $n_{\text{FEASIBLE}} \leftarrow 0$;
7  while $time \leq T_{\text{MAX}}$ do
8      $s' \leftarrow$Perturbation($s$,$p_{\text{MAX}}$);
9      $s' \leftarrow$RVND($s'$);
10     if isFeasible($s'$) then $n_{\text{FEASIBLE}} \leftarrow n_{\text{FEASIBLE}} + 1$ ;
11     else $n_{\text{INFEASIBLE}} \leftarrow n_{\text{INFEASIBLE}} + 1$ ;
12     if $f(s') < f(s)$ and isFeasible($s'$) then $s \leftarrow s'$ ;
13     else $u \leftarrow u + 1$ ;
14     if $u == u_{\text{MAX}}$ then
15         $\delta \leftarrow$ rand(0.05, 0.1);
16         if $n_{\text{FEASIBLE}} > n_{\text{INFEASIBLE}}$ then $\beta \leftarrow \beta/(1+\delta)$ ;
17         else $\beta \leftarrow \beta(1+\delta)$ ;
18         $u \leftarrow 0$;
19         $n_{\text{INFEASIBLE}} \leftarrow 0$;
20         $n_{\text{FEASIBLE}} \leftarrow 0$;
21     end
22 end
23 Return $s$ ;
```

mum revenue of all services (line 1). Then, a solution s is built using a greedy constructive heuristic (line 2) and this solution is improved using the RVND (line 3). Next, (lines 4 – 6), the variable that stores the current number of iterations without improvement, u, is initialized to zero, as well as the variables $n_{\text{INFEASIBLE}}$ and n_{FEASIBLE}, responsible for counting the number of infeasible and feasible solutions, respectively. Inside the iterative loop of the algorithm, firstly, a perturbation to escape from local optima is applied on the current solution s, generating a new solution s' (line 8). This new solution s' is improved by the RVND (line 9). If s' is feasible, after been improved by the RVND, then n_{FEASIBLE} is updated, otherwise $n_{\text{INFEASIBLE}}$ must be updated (lines 10 – 11). The best solution s is updated if s' has a better evaluation function value and also s' must be feasible, if not, then u is incremented (lines 12 – 13). In the end of the process (lines 14 – 21), if the algorithm has not found improvement on u_{MAX} iterations, then the value of β is decreased if n_{FEASIBLE} is greater than $n_{\text{INFEASIBLE}}$ (line 16) or increased if n_{FEASIBLE} is less than or equal to $n_{\text{INFEASIBLE}}$ (line 17). After updating β, variables u, $n_{\text{INFEASIBLE}}$ and n_{FEASIBLE} are set to zero again (lines 18 – 20). This iterative loop continues until a termination criterion is reached, defined here by a time limit (line 7). In the end of the execution the best solution found is returned (line 23).

The following components of the algorithm are detailed next: the construction of the initial solution, the local search procedures, and the perturbation operator.

4.4.3 Initial solution

The initial solution s is produced by a greedy constructive heuristic, inspired by [18, 17] and also used in [28]. Initially m seed p-d customers are included into each route, which are the farthest away from the depot. Then, to fill up the routes, this heuristic iteratively computes for each pickup customer i its best insertion ratio, which is the maximum revenue divided by the minimum increase of distance. The pickup customer i with the maximum insertion ratio is inserted at each iteration, together with its corresponding delivery $(n + i)$, which is included into its best position after i. If no more customers can be included into the solution, because of the distance constraints, then a new solution is created and the procedure finishes.

This procedure seeks to consider a long route firstly, and then iteratively includes closer customers. In [18], the authors stated that the idea of using this strategy is to quickly build an initial solution and then rely on the improvement phase of the algorithm for finding better solutions.

After the construction of this solution, the RVND local search procedures try to improve it by allowing to rearrange the selected customers and possibly introduces new customers into the solution.

4.4.4 Local search procedures

The concept of center of gravity (COG) for a route k, also used in [64, 65, 28], must be defined here to understand how the *GravityCenterExchange* neighborhood is investigated. The COG is based on the Cartesian coordinates (x_i, y_i) of all customers included into the solution weighted by their corresponding revenues r_i. Eqs. (4.19) and (4.20) show how to calculate the Cartesian coordinates of the center of gravity $(x_{\text{COG}}, y_{\text{COG}})$.

$$x_{\text{COG}} = \frac{\sum\limits_{i \in k} x_i r_i}{\sum\limits_{i \in k} r_i} \tag{4.19}$$

$$y_{\text{COG}} = \frac{\sum\limits_{i \in k} y_i r_i}{\sum\limits_{i \in k} r_i} \tag{4.20}$$

By knowing the center of gravity, then the appropriateness A_i can be calculated using Eq. (4.21), where $c_{i,\text{COG}}$ is the distance cost between the customer i and the COG.

$$A_i = \frac{r_i}{c_{i,\text{COG}}} \tag{4.21}$$

Like in [28], we consider that an appropriateness for a pickup and delivery pair is given by the sum $A_i + A_{n+i}$. The appropriateness is used in profit-increasing neighborhoods for inserting new requests into the solution. Requests with maximum appropriateness are considered first to be included into the solution.

4.4.4.1 Random Neighborhood Variable Descent

The random neighborhood variable descent adopted in IPPD works just like the RVND described in Chapter 3 (Section 3.4.4), without predefined order for exploring the neighborhoods. However new profit-increasing neighborhoods are incorporated into the RVND in order to better explore the search space of the MVPPDP. Moreover, in IPPD, each neighborhood is efficiently pre-evaluated exhaustively, considering moves in random order of p-d pairs and routes and applying always the best improving move.

As the IPPD deals with infeasible solutions during the its search, the search space becomes much bigger, thus the time needed to explore efficiently the search space grows proportionally. To avoid spending much time on each neighborhood and to seek improvements quickly, the IPPD follows [63] and adopts a similar idea of the granular local search. In the IPPD, a set of nearest neighborhoods for each pickup customer is initially defined and, before each move is applied, it is verified if this move involves at least one customer belonging to the set of nearest neighborhoods for this pickup customer. If this is the case, then the move is applied, if not, then this move is rejected. In IPPD, the size of this set for each pickup customer was set to $\{50, 100, 150, 200, 250, 300\}$ and the value that provided better results was 250.

All intra-route and inter-route neighborhoods used by IPPD are also used in IPDS (3.4.4.1 and 3.4.4.2). The IPPD also explores profit-increasing neighborhoods and a repairing neighborhood. The repairing neighborhood structure is used with the intention of obtaining a feasible solution by removing pairs from the solution.

4.4.4.2 Intra-route neighborhood structures:

$\boldsymbol{N^{(1)}}$ – *PairSwap* considers two pairs of customers $(i, n+i)$ and $(j, n+j)$ and swaps the pickup customer i with the pickup customer j, as well as the delivery customer $(n+i)$ with the delivery customer $(n+j)$.

$\boldsymbol{N^{(2)}}$ – *PairShift* considers a pair of customers $(i, n+i)$ and relocates the pickup customer i in another position of the route and then finds another position to insert the corresponding delivery customer $(n+i)$, after the pickup customer.

$\boldsymbol{N^{(3)}}$ – *PickShift* relocates a pickup customer i in another position before the delivery customer $(n+i)$.

$\boldsymbol{N^{(4)}}$ – *DelShift* relocates a delivery customer $(n+i)$ in another position after the pickup customer i.

$\boldsymbol{N^{(5)}}$ – *BlockSwap* swaps a block B_i with another block B_j.

$\boldsymbol{N^{(6)}}$ – *BlockShift* relocates a block B_i in another position.

4.4.4.3 Inter-route neighborhood structures:

$\boldsymbol{N}^{(7)}$ – *InterPairSwap* selects a pair of customers $(i, n+i)$ from a route k_1 and another pair $(j, n+j)$ from a route k_2 and swaps the pickup customer i with the pickup customer j. The delivery customer $(n+i)$ is swapped with the delivery customer $(n+j)$.

$\boldsymbol{N}^{(8)}$ – *InterPairShift* takes a pair of customers $(i, n+i)$ from a route k_1 and transfer this pair to a route k_2. After defining the position to insert the pickup customer i in k_2, the delivery customer $(n+i)$ is inserted in a following position.

$\boldsymbol{N}^{(9)}$ – *InterBlockSwap* selects a block B_i from a route k_1 and another block B_j from a route k_2 and swaps them.

$\boldsymbol{N}^{(10)}$ – *InterBlockShift* transfers a block B_i from a route k_1 to a route k_2.

4.4.4.4 Profit-increasing neighborhood structures:

$\boldsymbol{N}^{(11)}$ – *Insert* takes a pair of customers $(i, n+i)$ not included in the solution and insert this pair on a route k. The pickup customer i is inserted on a position of k and the delivery customer $(n+i)$ is inserted on a following position of k. The pairs are analysed by following a descending order of appropriateness.

$\boldsymbol{N}^{(12)}$ – *Replace* takes a pair of customers $(i, n+i)$ not included in the solution and a pair of customers $(j, n+j)$ that belongs to the solution and swaps them.

$\boldsymbol{N}^{(13)}$ – *GravityCenterExchange* removes the farthest pair $(i, n+i)$ from the center of gravity from a route k and inserts non-included pairs into k as long as the duration constraints are met. New requests are inserted by considering a descending order of appropriateness.

4.4.4.5 Repairing neighborhood structure:

$\boldsymbol{N}^{(14)}$ – *Remove* takes a pair of customers $(i, n+i)$ and removes it from the solution.

4.4.5 Perturbation operator

The perturbation operator of the IPPD algorithm is based on the *Remove* neighborhood and consists in removing n_{PERT} random p-d pairs from the solution. The number of pairs

n_{PERT} to be removed is randomly selected in $\{1, 2, ..., p_{\text{MAX}}\}$ using an uniform distribution probability. Thus, p_{MAX} is a parameter of IPPD that limits the number of perturbation moves.

4.5 Computational results

The set of instances from [28] was used in order to perform the computational experiments of the IPPD algorithm. This set contains 36 instances divided in six subsets of six instances each. These subsets have 10, 25, 50, 125, 250 and 500 pickup and delivery pairs, thus corresponding to 20, 50, 100, 250, 500 and 1000 customers. They are subdivided into groups called: small (20 and 50 customers), medium (100 and 250 customers) and large (500 and 1000 customers). The customer locations were randomly generated on a bi-dimensional plane (x, y), in which both x and y are in the interval $[-1000, 1000]$ and both depots are located at $(0, 0)$. Each pair has an integer demand between $[1, 50]$. The revenues were generated using three strategies: *i)* equal for all pairs (F); *ii)* proportional to the demands (P); and *iii)* randomly distributed (R). The distance constraint can be tight (S) or relaxed (L) and the vehicle numbers vary from 2 to 8.

The IPPD algorithm was also developed in C++ using OptFrame [20]. Each test was performed on a single core of a Intel Core i7 3.4 GHz, 16 GB of RAM using Ubuntu 14.04. It is noteworthy that the computer used is similar to the one used in [28]. Moreover, the IPPD uses three main parameters: the strength of the perturbation operator p_{MAX}, which has been set to $\max_{k \in K} |k|$, representing the maximum route size of the current solution (this value is a solution-dependent parameter and provided good solutions during our preliminary experiments), u_{MAX}, the maximum number of iterations without improvement to update the coefficient factor β, which has been set to 5 iterations at most and finally, the stopping criterion T_{MAX}, which has been set for each group of instances to the same CPU time of the current state-of-the-art methods from [28]: 1 second per run for each instance with 20 and 50 customers, 10 seconds for each instance with 100 and 250 customers, and 100 seconds for each instance with 500 and 1000 customers.

Previous authors have reported results over 5 runs, thus we also report the results on 5 runs of the algorithm. For each instance, we obtain a solution using IPPD, z_{IPPD}. The percentage gap relative to the best known solution (BKS), z_{BKS} is computed according to Eq. (4.22). All best known solutions were collected from [28] after 20 hours executing their algorithms on each instance.

$$Gap = 100 \times (z_{\text{IPPD}} - z_{\text{BKS}})/z_{\text{IPPD}} \tag{4.22}$$

The objective in MVPPDP is to maximize the total profit, thus if Eq. (4.22) returns a negative gap, it means that this solution has a lower value compared to the best known solution, therefore worst quality. On the other hand, if a positive value is returned, it means that the solution obtained has a greater value than the best known solution, hence a new best known solution is found.

4.5.1 Sensitivity analysis on the components of the method

First of all, an analysis on the relevance of each component of the IPPD was done. We started with the standard version of the algorithm, containing all neighborhoods, and set alternative configurations by removing different neighborhoods on each test. These configurations are listed below:

Base – The standard configuration, with all local-search neighborhoods;

$\boldsymbol{WN_1}$ – Base configuration without the *PairSwap* and *InterPairSwap* neighborhoods;

$\boldsymbol{WN_2}$ – Base configuration without the *PairShift* and *InterPairShift* neighborhoods;

$\boldsymbol{WN_{34}}$ – Base configuration without the *PickShift* and *DelShift* neighborhoods;

$\boldsymbol{WN_5}$ – Base configuration without the *BlockSwap* and *InterBlockSwap* neighborhoods;

$\boldsymbol{WN_6}$ – Base configuration without the *BlockShift* and *InterBlockShift* neighborhoods;

$\boldsymbol{WN_{56}}$ – Base configuration without the *BlockShift*, *InterBlockShift*, *BlockSwap* and *InterBlockSwap* neighborhoods;

$\boldsymbol{WN_{11}}$ – Base configuration without the *Insert* neighborhood;

$\boldsymbol{WN_{12}}$ – Base configuration without the *Replace* neighborhood;

$\boldsymbol{WN_{13}}$ – Base configuration without the *GravityCenterExchange* neighborhood;

$\boldsymbol{WN_{14}}$ – Base configuration without the *Remove* neighborhood;

$\boldsymbol{W_I}$ – Base configuration without dealing with infeasible solutions;

$\boldsymbol{W_G}$ – Base configuration without granular local searches.

The algorithms generated by each configuration were tested on instances from [28], performing five runs for each of the 36 instances, and using the same stopping criteria defined in Section 4.5. Table 4.1 shows, for each variant of the algorithm, the average gap for each set of instances (Gap-(20,50), Gap-(100,250), Gap(500,1000)) as well as the total average gap (Avg). These gaps are calculated using the obtained solution on each algorithm and the BKS obtained in [28] after 20 hours running their algorithms for each instance. All gaps are negative because the stopping criteria is much smaller than 20 hours and the algorithms did not have enough time to reach a better solution than the BKS or even greater.

Table 4.1: Results for each configuration of the IPPD

Configuration	Gap-(20,50) (%)	Gap-(100,250) (%)	Gap-(500,1000) (%)	Avg (%)
Base	-0.57	-4.21	-5.79	-3.52
$\boldsymbol{WN_1}$	-0.47	-4.33	-6.26	-3.69
$\boldsymbol{WN_2}$	-1.38	-5.43	-8.10	-4.97
$\boldsymbol{WN_{34}}$	-0.47	-5.03	-6.95	-4.15
$\boldsymbol{WN_5}$	-0.46	-4.46	-5.70	-3.54
$\boldsymbol{WN_6}$	**-0.34**	-4.12	**-5.35**	-3.27
$\boldsymbol{WN_{56}}$	-0.43	**-3.58**	-5.75	**-3.25**
$\boldsymbol{WN_{11}}$	-23.49	-129.73	-50.66	-67.96
$\boldsymbol{WN_{12}}$	-0.87	-7.06	-7.71	-5.21
$\boldsymbol{WN_{13}}$	-0.63	-4.19	-5.70	-3.51
$\boldsymbol{WN_{14}}$	-0.50	-5.24	-6.34	-4.03
$\boldsymbol{W_I}$	-1.58	-9.02	-5.53	-5.38
$\boldsymbol{W_G}$	-0.57	-4.04	-6.77	-3.79

In Table 4.1, it is clearly observed that the most relevant neighborhood is the *Insert* neighborhood, because removing this neighborhood (WN_{11}) results in the worst gaps of all configurations. In contrast, the best average gap (-3.25%) is obtained by removing all block-based neighborhoods. These neighborhoods do not seem to contribute to the quality of the final solution, at least considering the time limits imposed. Another interesting fact is that working only with feasible solutions seems to hamper the algorithm for reaching better quality solutions, as we can see in gaps obtained by W_I. Moreover, the use of granular local searches suggests to contribute for a quickly production of good solutions.

The gaps obtained, on all runs, by all algorithms are better observed by the box plots of Figure 4.2. The box plots were generated without gaps from W_{11} for a better visualisation. By these box plots we can also see the importance of *PairShift*, *InterPairShift* and *Replace*, as without using them, the algorithm got worse values of gaps. We can also see a slightly better performance of the IPPD without using block-based neighborhoods (WN_{56}) over the rest of the algorithms.

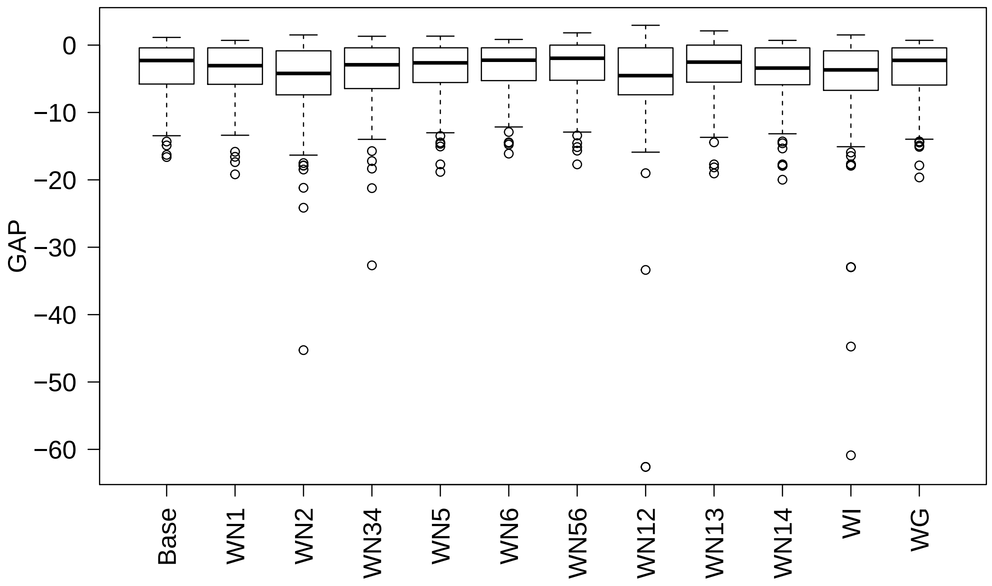

Figure 4.2: Box plots showing the gaps for each configuration of the IPPD

In order to verify previous analyses, a Friedman test based on the gap values of each algorithm was performed. The test returned a value $p < 2.2 \times 10^{-16}$, confirming significant statistical differences. Next, paired-sample Wilcoxon tests were performed to compare the Base algorithm with each configuration. The results of these tests are reported in Table 4.2.

Table 4.2: Results from paired-sample Wilcoxon tests with the Base algorithm

Algorithms	**p-value**
Base – $\boldsymbol{WN_1}$	0.78
Base – $\boldsymbol{WN_2}$	**0.00**
Base – $\boldsymbol{WN_{34}}$	0.50
Base – $\boldsymbol{WN_5}$	0.95
Base – $\boldsymbol{WN_6}$	0.64
Base – $\boldsymbol{WN_{56}}$	0.33
Base – $\boldsymbol{WN_{11}}$	$\mathbf{2.2 \times 10^{-16}}$
Base – $\boldsymbol{WN_{12}}$	**0.01**
Base – $\boldsymbol{WN_{13}}$	0.65
Base – $\boldsymbol{WN_{14}}$	0.48
Base – $\boldsymbol{W_I}$	**0.02**
Base – $\boldsymbol{W_G}$	0.80

From these results we can statistically confirm the importance of using the *Insert*, *PairShift*, *InterPairShift*, *Replace* and also our proposed strategy for dealing with infeasible solutions as the value of p was less than threshold of 0.05. Notably, without using these configurations the algorithm does not generate good solutions. For other configurations, when comparing with the Base configuration the values of p were not below the threshold, thus, we cannot attest the difference of performance. Nevertheless, because of the small time limits and the slightly better performance, we decided to keep the Base configuration, but removing all block-based neighborhoods to compare these results with the state-of-art results.

4.5.2 Performance comparisons on MVPPDP instances

Tables 4.3, 4.4 and 4.5 display the Gap$_{BKS}$ and the Gap$_{Avg}$ obtained after 5 runs for each algorithm on each instance from [28]. The Gap$_{BKS}$ represents the relative percentage deviation (Eq. 4.22) given by the best solution obtained on each algorithm (5 runs) and the BKS obtained in [28] after 20 hours running their algorithms for each instance. The Gap$_{Avg}$ is the relative percentage deviation calculated with the average solutions on 5 runs and the BKS. The algorithms GVNSseq, GVNSsa and GLS were all implemented in [28] and the IPPD corresponds to our WN_{56} configuration, that is, without exploring block-based neighborhoods. For each instance, the best result considering all methods is highlighted in boldface.

Table 4.3: Results for the MVPPDP with 20 and 50 customers on 5 runs for each instance. Time limit set to 1 second per run.

		GVNSseq		**GVNSsa**		**GLS**		**IPPD**	
Inst	**Req**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**
1FS	**20**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
2FL	**20**	-3.38	-3.38	-3.38	-3.38	-3.15	-3.15	**-0.41**	**-0.41**
3PS	**20**	**-0.55**	**-0.55**	**-0.55**	**-0.55**	**-0.55**	**-0.55**	**-0.55**	**-0.55**
4PL	**20**	0.00	0.00	0.00	0.00	-1.14	-1.14	**0.70**	**0.70**
5RS	**20**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-0.06	**0.00**	**0.00**
6RL	**20**	-2.63	-2.63	-2.63	-2.63	-2.43	-2.43	**0.04**	**0.04**
7FS	**50**	**-0.60**	**-0.60**	**-0.60**	**-0.60**	-7.83	-7.83	**-0.60**	**-0.60**
8FL	**50**	**0.00**	-0.37	**0.00**	-0.66	-1.24	-3.89	**0.00**	**0.00**
9PS	**50**	-4.27	-5.69	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
10PL	**50**	**0.00**	**-0.77**	**0.00**	-3.76	-5.31	-7.69	**0.00**	-2.32
11RS	**50**	**0.00**	**0.00**	**0.00**	**0.00**	-0.52	-0.89	**0.00**	**0.00**
12RL	**50**	**0.00**	**-0.03**	**0.00**	-2.01	-4.14	-5.22	-1.95	-1.95
	Avg	-0.95	-1.17	-0.60	-1.13	-2.19	-2.74	**-0.23**	**-0.43**

Looking at these tables, it is evident the good performance of the IPPD algorithm on small and medium-sized instances, as the average gaps are greater than gaps from the other algorithms: -0.43% for the small set and -3.58% for the medium set. For the large set, the best performance observed is for the GVNS with the self-adaptive VND (GVNSsa). It seems that the IPPD did not have enough time to explore the search space

Table 4.4: Results for the MVPPDP with 100 and 250 customers on 5 runs for each instance. Time limit set to 10 seconds per run.

		GVNSseq		**GVNSsa**		**GLS**		**IPPD**	
Inst	**Req**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**
13FS	**100**	**-5.02**	**-5.22**	**-5.02**	-5.46	-6.75	-6.98	-5.83	-5.98
14FL	**100**	**0.00**	**-0.12**	-0.18	-4.22	-0.89	-3.04	**0.00**	-3.17
15PS	**100**	-0.95	-3.46	-4.20	-7.22	-1.21	-3.80	**-0.39**	**-2.99**
16PL	**100**	-0.61	-2.59	-3.15	-3.41	-4.83	-7.60	**0.60**	**0.02**
17RS	**100**	-5.99	-5.99	-5.99	-5.99	-3.68	-6.05	**0.00**	**-5.02**
18RL	**100**	-0.28	-1.10	-1.44	-2.07	-2.84	-5.01	**0.00**	**-0.60**
19FS	**250**	**-4.37**	**-5.15**	-4.45	-5.89	-7.96	-12.15	-4.54	-6.37
20FL	**250**	**-1.40**	**-4.35**	-2.98	-4.84	-5.10	-5.57	-4.26	-5.49
21PS	**250**	-3.84	-4.94	-2.77	-4.42	-5.82	-9.91	**-1.01**	**-3.59**
22PL	**250**	-3.30	-4.35	-2.40	-3.32	-7.11	-9.42	**-1.09**	**-3.12**
23RS	**250**	-2.04	-3.15	**-1.11**	**-2.03**	-5.11	-7.69	-1.40	-3.57
24RL	**250**	-3.30	-4.06	-3.22	-5.07	-6.88	-9.14	**-1.79**	**-3.14**
	Avg	-2.59	-3.71	-3.08	-4.50	-4.85	-7.20	**-1.64**	**-3.58**

Table 4.5: Results for the MVPPDP with 500 and 1000 customers on 5 runs for each instance. Time limit set to 100 seconds per run.

		GVNSseq		**GVNSsa**		**GLS**		**IPPD**	
Inst	**Req**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**	**Gap$_{BKS}$**	**Gap$_{Avg}$**
25FS	**500**	**-2.46**	**-3.86**	-3.23	-4.22	-8.83	-11.69	-4.04	-7.32
26FL	**500**	-3.64	-4.23	**-2.67**	**-3.92**	-3.39	-6.00	-4.10	-7.08
27PS	**500**	-1.02	-2.52	-1.37	-2.99	-9.19	-11.99	**0.72**	**-1.40**
28PL	**500**	-2.54	-2.99	-2.74	-3.33	-7.52	-8.71	**1.82**	**-0.50**
29RS	**500**	**0.00**	-2.74	-1.95	**-2.65**	-12.04	-14.38	-6.29	-8.14
30RL	**500**	-1.35	**-1.95**	-0.98	-2.06	-6.73	-9.72	**-1.29**	-2.07
31FS	**1000**	**-7.25**	-9.96	-8.48	**-9.27**	-8.31	-11.02	-11.80	-14.09
32FL	**1000**	-5.39	-6.50	**-3.80**	**-5.26**	-4.07	-6.52	-10.12	-11.39
33PS	**1000**	-5.16	-6.13	-4.31	-5.18	-6.83	-7.90	**1.47**	**-1.49**
34PL	**1000**	-3.12	-4.85	**-2.55**	**-3.79**	-4.92	-6.23	-3.26	-4.63
35RS	**1000**	-4.83	-5.54	**-3.59**	**-4.63**	-9.25	-12.23	-4.90	-6.53
36RL	**1000**	-3.20	-4.12	**-3.06**	**-4.03**	-5.56	-6.83	-2.90	-4.29
	Avg	-3.33	-4.62	**-3.23**	**-4.28**	-7.22	-9.44	-3.72	-5.75

of feasible and infeasible solutions, even adopting the concept of granular local search. Still, it could achieve better solutions, on average, on two instances of 500 customers and one of 1000 customers. By working with infeasible solutions, the IPPD was able to reach new better solutions for the instances **16PL**, **4PL**, **6RL**, **27PS**, **28PL** and **33PS**, because the gaps found are greater than zero. To sum up, the IPPD got greater or equal average gaps on 20 instances, the GVNSsa obtained 14 better or equal average gaps, the GVNSseq found 13 greater or equal average gaps and the GLS was not able to find better gaps, only the same values for just 3 instances.

Once more, the statistical significance of these results is investigated by performing a Friedman test to compare the average gap values obtained for each algorithm on each instance. The Friedman test returned a value $p < 4.5 \times 10^{-8}$, which means that exists a significant difference of performance. Pairwise Wilcoxon tests were also performed to locate these differences, and the results are presented in Table 4.6. These tests confirmed that IPPD is significantly better than the GLS, but it does not differ from the GVNSseq and the GVNSsa, probably because of the performance presented on large instances. Nevertheless, because of its good performance on small and medium-size instances, the IPPD still can be considered an excellent choice for solving the MVPPDP, as it can achieve very good solutions by exploring larger and infeasible neighborhoods.

Table 4.6: Results of pairwise Wilcoxon tests

Algorithms	**p-value**
IPPD–GLS	0.00
IPPD–GVNSseq	0.60
IPPD–GVNSsa	0.41

4.6 Conclusions

The study of the multi-vehicle profitable pickup and delivery problem has been conducted in this work. A solution for the MVPPDP is not so trivial to find, because firstly, a good selection of a subset of customer requests to attend is needed, next an efficient decision of which vehicle will attend which request and finally what is the best route for each vehicle, considering a maximum travel time, the capacity of the vehicle and the precedence of a pickup over the delivery for each request. These characteristics lead the search space of the MVPPDP to have many basins of attractions. Thus, an efficient method for solving the MVPPDP must have good techniques to avoid getting trapped in these basins.

We designed an algorithm with the objective of escaping from these basins of attractions, by allowing it to work with infeasible solutions. This algorithm, called IPPD, is based on the iterated local search and applies, successively, local searches using the random variable neighborhood descent. The local searches explore neighborhoods for optimizing the order of visits for each vehicle and also for increasing the total profits. In order to reduce the computational time for all neighborhoods, because the search space is increased when dealing with infeasible solutions, the idea of granular local search was applied in the IPPD.

The proposed algorithm was tested on benchmark instances and statistical analyses proved the very good performance of working with infeasible solutions. Furthermore, the results obtained were compared to three algorithms from the literature and, the IPPD produced the best results on small and medium-size instances within the same time limits. It was even able to find new best solutions on 6 instances. However, probably because of the time limit, it was not the best algorithm for the large instances, but its performance was not so far from the other two algorithms, because no statistical difference between them was found. Thus, the IPPD has potential to be adapted to quickly produce better solutions for the large set of instances of the MVPPDP.

Chapter 5

Conclusions and Future Work

This thesis proposed an efficient heuristic for one-to-one pickup and delivery problems (PDP), in which the objective is to find a set of minimum cost routes, which start and end at a depot, for a fleet of vehicles to service pickup and delivery pairs of customers, subject to some constraints. The one-to-one PDPs are $\mathcal{NP}$-hard, thus larger instances may require prohibitive processing times for their exact solution. This fact motivates the development of heuristics for solving these problems. However, building a heuristic that deals with one-to-one pickup and delivery pairs is not an easy task, since it must be very well designed to be capable of working with pairs of customers efficiently, as the search space becomes larger than working with only one customer at a time.

Therefore, this thesis focused on developing highly-skilled techniques to deal with two one-to-one PDPs: the Multi-Vehicle One-to-one Pickup and Delivery Problem with Split Loads (MPDPSL) [23] and the Multi-Vehicle Profitable Pickup And Delivery Problem (MVPPDP)[28].

The MPDPSL relaxes the classical constraint, obtained from the Vehicle Routing Problem (VRP), of restricting a customer to be serviced by at most one vehicle, thus, a pair of customers in the MPDPSL can be attended by more than one vehicle. The MPDPSL is also defined by a duration constraint, which may force the solution to contain more than one vehicle for serving all customers. The objective of the MPDPSL is to find a set of routes using at most all available fleet of vehicles that minimizes the total distance, respecting the capacity, maximum tour time and also precedence for pickups on deliveries.

In Chapter 3, the MPDPSL is properly defined and we propose an algorithm based on Iterated Local Search (ILS) and the Random Variable Neighborhood Search (RVND) to solve the MPDPSL. The core of the algorithm, called IPDS, consists of a new large neighborhood search, which reduces the problem of finding the best insertion combination of a pickup and delivery pair into a route to a resource-constrained shortest path problem, and solves it via dynamic programming. Once the non-dominated labels are known for each separate route, a knapsack problem is solved to find the best insertion plan in the whole solution. Our computational experiments on two set of instances of the single-vehicle version from the literature demonstrate the very good performance of the algorithm. On the first set of instances, created by [47], the IPDS algorithm was executed 20 times and it was able to find better average results on all 45 instances within the same time limits of other two algorithms from the literature, which were executed only once per instance. The IPDS was even capable of improving the best known solutions for all 45 instances, with an average improvement of 3.41%. On the second set of instances, created by [23], the IPDS algorithm was again executed 20 times, limited by the same computational times of [23], for small and medium instances and 5 times for the largest set of instances containing 500 pairs of costumers. The best found solutions of IPDS were compared with the best found solutions of the state-of-the-art from [23] and, again, the IPDS proved its effectiveness, improving 47 instances from a total of 48. Sensitivity analysis on the components of the IPDS showed the importance of each neighborhood incorporated on it. Finally, additional results on new proposed instances for the MPDPSL are also reported, producing consistently good solutions over 20 runs.

Due to the great performance obtained in solving the IPDS, we decided to address another one-to-one PDP, in fact the MVPPDP, and inspect the performance of an adaptation of the IPDS for solving this problem. Differing from the MPDPSL, the MVPPDP restricts a pair of customers to be visited by at most one vehicle, but it also imposes a duration constraint for each vehicle. Each pair of customers in the MVPPDP has a revenue to be obtained if this pair is attended. The objective of the MVPPDP is to find a set of routes using at most all available fleet of vehicles that will attend at most all customers that maximizes the total profit, respecting the capacity, maximum tour time and precedence for pickups on deliveries as well. The total profit is given by the total revenue obtained minus the total travel costs.

Chapter 4 defines the MVPPDP and describes a heuristic algorithm, named IPPD, based on the ILS and the RVND for solving the MVPPDP. The main contribution of this Chapter is to present the importance of allowing infeasible solutions during the search phase of a heuristic algorithm to solve the MVPPDP. A dynamic strategy was adopted for updating a coefficient factor that penalizes the violation of the duration constraint. To reduce the search space of infeasible solutions, a granular local search was also incorporated into the algorithm. We performed computational experiments on benchmark instances from the literature [28]. First of all, a sensitivity analysis on each component of the IPPD was done. This analysis showed the power of the proposed techniques for solving the MVPPDP, but some neighborhoods that use the concept of blocks were removed from the standard version of the algorithm, because they were not contributing to the final solution, considering the time limits. Then, the results of this adapted algorithm were compared to three algorithms of the state-of-the-art of the MVPPDP [28]: two algorithms based on the Variable Neighborhood Search, one with a pre-defined order of the local searches from the VND (GVNSseq) and the other with a self-adaptive order of theses local searches from the VND (GVNSsa), and a Guided Local Search (GLS) algorithm. The gaps for all algorithms were calculated using the best known solution obtained in [28] after 20 hours of execution for each instance of their algorithms. The IPPD algorithm had a great performance on the small and medium-sized instances, with average gaps of -0.43% for the small set and -3.58% for the medium set. For the large set, even applying granular local searches, the IPPD did not has the best performance, but also not the worst. It was even capable of improving the best known solution for 3 large instances limited by 100 seconds, instead of 20 hours. Overall, the IPPD was able to find new best solutions for 6 instances out of 36 instances and this is really impressive, considering that the time limits are in seconds. To sum up, the IPPD got greater or equal average gaps on 20 instances, the GVNSsa obtained 14 better or equal average gaps, the GVNSseq found 13 greater or equal average gaps and the GLS was not able to find better gaps, only the same values for just 3 instances.

It is noteworthy that both problems (MPDPSL and MVPPDP) are one-to-one pickup and delivery problems, have common constraints (capacity, precedence and duration), deal with increasing size of their solutions during the search and are very restrictive problems, but during this work we realized that even with these many similar characteristics, both problems need very specific techniques to be efficiently solved. As future works, we intend to keep following this research line, developing specific techniques to solve these problems, specially new techniques for adapting our algorithm to quickly solve the MVPPDP, and also test it with longer time limits. We want also to continue generalizing neighborhoods and their exploration methods as well as extending the methodology to a wider range of not only one-to-one PDPs, but also other difficult vehicle routing variants. Another possible research direction is the "non-standard" hybridization heuristic-exact, involving dynamic programming rather than mathematical programming.

References

[1] Aleman, R. E., Hill, R. R. A Tabu Search with Vocabulary Building Approach for the Vehicle Routing Problem with Split Demands. *Int. J. Metaheuristics 1*, 1 (maio de 2010), 55–80.

[2] Aleman, R. E., Zhang, X., Hill, R. R. A ring-based diversification scheme for routing problems. *International Journal of Mathematics in Operational Research 1*, 1 (2009), 163–190.

[3] Aleman, R. E., Zhang, X., Hill, R. R. An Adaptive Memory Algorithm for the Split Delivery Vehicle Routing Problem. *Journal of Heuristics 16*, 3 (junho de 2010), 441–473.

[4] Archetti, C., Bianchessi, N., Speranza, M. G. Optimal solutions for routing problems with profits. *Discrete Applied Mathematics 161*, 4 (2013), 547–557.

[5] Archetti, C., Feillet, D., Hertz, A., Speranza, M. G. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society 60*, 6 (2009), 831–842.

[6] Archetti, C., Hertz, A., Speranza, M. G. A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem. *Transportation Science 40*, 1 (2006), 64–73.

[7] Archetti, C., Hertz, A., Speranza, M. G. Metaheuristics for the team orienteering problem. *Journal of Heuristics 13*, 1 (2007), 49–76.

[8] Archetti, C., Savelsbergh, M. W. P., Speranza, M. G. Worst-Case Analysis for Split Delivery Vehicle Routing Problems. *Transportation Science 40*, 2 (maio de 2006), 226–234.

[9] Archetti, C., Speranza, M. G., Vigo, D. Vehicle routing problems with profits. *Vehicle Routing: Problems, Methods, and Applications 18* (2014), 273.

[10] Bent, R., Hentenryck, P. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research 33*, 4 (4 2006), 875–893.

[11] Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., Laporte, G. Static pickup and delivery problems: a classification scheme and survey. *Top 15*, 1 (2007), 1–31.

[12] Boese, K. D. *Cost versus distance in the traveling salesman problem.* UCLA Computer Science Department, 1995.

[13] Boudia, M., Prins, C., Reghioui, M. An Effective Memetic Algorithm with Population Management for the Split Delivery Vehicle Routing Problem. In *Hybrid Metaheuristics* (2007), T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, Eds., vol. 4771 of *Lecture Notes in Computer Science*, Springer, p. 16–30.

[14] Boussier, S., Feillet, D., Gendreau, M. An exact algorithm for team orienteering problems. *4OR: A Quarterly Journal of Operations Research 5*, 3 (2007), 211–230.

[15] Campos, V., Corberán, A., Mota, E. A scatter search algorithm for the split delivery vehicle routing problem. In *Advances in computational intelligence in transport, logistics, and supply chain management.* Springer, 2008, p. 137–152.

[16] Cassani, L., Righini, G. Heuristic algorithms for the tsp with rear-loading. In *35th Annual Conference of the Italian Operations Research Society (AIRO XXXV), Lecce, Italy* (2004).

[17] Chao, I.-M., Golden, B. L., Wasil, E. A. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research 88*, 3 (1996), 475 – 489.

[18] Chao, I.-M., Golden, B. L., Wasil, E. A. The team orienteering problem. *European Journal of Operational Research 88*, 3 (1996), 464 – 474.

[19] Clarke, G., Wright, J. W. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research 12*, 4 (julho de 1964), 568–581.

[20] Coelho, I. M., Munhoz, P. L. A., Haddad, M. N., Coelho, V. N., Silva, M. M., Souza, M. J. F., Ochi, L. S. A computational framework for combinatorial optimization problems. In *VII ALIO/EURO Workshop on Applied Combinatorial Optimization* (Porto, 2011), p. 51–54.

[21] Cordeau, J.-F., Laporte, G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological 37*, 6 (2003), 579–594.

[22] Cordeau, J.-F., Laporte, G., Ropke, S. *Recent Models and Algorithms for One-to-One Pickup and Delivery Problems.* Springer US, Boston, MA, 2008, p. 327–357.

[23] Şahin, M., Çavuşlar, G., Öncan, T., Şahin, G., Tüzün Aksu, D. An efficient heuristic for the Multi-vehicle One-to-one Pickup and Delivery Problem with Split Loads. *Transportation Research Part C: Emerging Technologies 27* (2013), 169–188.

[24] Dang, D.-C., Guibadj, R. N., Moukrim, A. A pso-based memetic algorithm for the team orienteering problem. In *European Conference on the Applications of Evolutionary Computation* (2011), Springer, p. 471–480.

[25] Derigs, U., Li, B., Vogel, U. Local search-based metaheuristics for the split delivery vehicle routing problem. *The Journal of the Operational Research Society 61*, 9 (setembro de 2010), 1356–1364.

[26] Dror, M., Trudeau, P. Savings by Split Delivery Routing. *Transportation Science 23*, 2 (maio de 1989), 141–145.

[27] Dror, M., Trudeau, P. Split delivery routing. *Naval Research Logistics (NRL) 37*, 3 (1990), 383–402.

[28] Gansterer, M., Küçüktepe, M., Hartl, R. F. The multi-vehicle profitable pickup and delivery problem. *OR Spectrum 39*, 1 (2017), 303–319.

[29] Gendreau, M., Hertz, A., Laporte, G. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research 40*, 6 (novembro de 1992), 1086–1094.

[30] Hu, Q., Lim, A. An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research 232*, 2 (2014), 276–286.

[31] Jepsen, M. K., Petersen, B., Spoorendonk, S., Pisinger, D. A branch-and-cut algorithm for the capacitated profitable tour problem. *Discrete Optimization 14* (2014), 78–96.

[32] Ke, L., Archetti, C., Feng, Z. Ants can solve the team orienteering problem. *Computers & Industrial Engineering 54*, 3 (2008), 648–665.

[33] Kubiak, M. Distance measures and fitness-distance analysis for the capacitated vehicle routing problem. In *Metaheuristics*. Springer, 2007, p. 345–364.

[34] Labadie, N., Mansini, R., Melechovskỳ, J., Calvo, R. W. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research 220*, 1 (2012), 15–27.

[35] Lahyani, R., Khemakhem, M., Semet, F. Heuristics for rich profitable tour problems. In *Modeling, Simulation and Applied Optimization (ICMSAO), 2013 5th International Conference on* (2013), IEEE, p. 1–3.

[36] Laporte, G. Fifty years of vehicle routing. *Transportation Science 43*, 4 (2009), 408–416.

[37] Laporte, G., Martello, S. The selective travelling salesman problem. *Discrete applied mathematics 26*, 2-3 (1990), 193–207.

[38] Lin, S.-W. Solving the team orienteering problem using effective multi-start simulated annealing. *Applied Soft Computing 13*, 2 (2013), 1064–1073.

[39] Lin, S.-W., Vincent, F. Y. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research 217*, 1 (2012), 94–107.

[40] Lourenço, H. R., Martin, O., Stützle, T. Iterated Local Search. In *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds., vol. 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2003, p. 321–353.

[41] Mitra, S. An Algorithm for the generalized Vehicle Routing Problem with Backhauling. *APJOR 22*, 2 (2005), 153–170.

[42] Mitra, S. A Parallel Clustering Technique for the Vehicle Routing Problem with Split Deliveries and Pickups. *The Journal of the Operational Research Society 59*, 11 (2008), 1532–1546.

[43] Mladenović, N., Hansen, P. Variable neighborhood search. *Computers & Operations Research 24*, 11 (1997), 1097–1100.

[44] Nguyen, V. H., Nguyen, T. T. T. Approximating the asymmetric profitable tour. *Electronic Notes in Discrete Mathematics 36* (2010), 907–914.

[45] Norstad, I., Fagerholt, K., Laporte, G. Tramp ship routing and scheduling with speed optimization. *Transportation Research Part C: Emerging Technologies 19*, 5 (2011), 853 – 865. Freight Transportation and Logistics (selected papers from {ODYSSEUS} 2009 - the 4th International Workshop on Freight Transportation and Logistics).

[46] Nowak, M. *The Pickup and Delivery Problem with Split Loads.* PhD thesis, 2005.

[47] Nowak, M., Ergun, Ö., III, C. C. W. Pickup and Delivery with Split Loads. *Transportation Science 42*, 1 (2008), 32–43.

[48] Nowak, M., Ergun, Ö., III, C. C. W. An empirical study on the benefit of split loads with the pickup and delivery problem. *European Journal of Operational Research 198*, 3 (2009), 734–740.

[49] Parragh, S. N., Doerner, K. F., Hartl, R. F. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft 58*, 1 (2008), 21–51.

[50] Parragh, S. N., Doerner, K. F., Hartl, R. F. Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research 37*, 6 (2010), 1129–1138.

[51] Parragh, S. N., Pinho de Sousa, J., Almada-Lobo, B. The dial-a-ride problem with split requests and profits. *Transportation Science 49*, 2 (2014), 311–334.

[52] Ropke, S., Cordeau, J.-F. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science 43*, 3 (2009), 267–286.

[53] Ropke, S., Pisinger, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science 40*, 4 (2006), 455–472.

[54] Sexton, T. R., Choi, Y.-M. Pickup and delivery of partial loads with âsoftâ time windows. *American Journal of Mathematical and Management Sciences 6*, 3-4 (1986), 369–398.

[55] Shaw, P. *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, p. 417–431.

[56] Silva, M. M. Uma heurística baseada em Iterated Local Search para o Problema de Roteamento de Veículos com Entregas Fracionárias. Dissertação de Mestrado, Universidade Federal Fluminense, Instituto de Computação, Programa de Pós-graduação em Computação, Niterói, 2013.

[57] Silva, M. M., Subramanian, A., Ochi, L. S. An iterated local search heuristic for the split delivery vehicle routing problem. *Computers & Operations Research 53* (2015), 234–249.

[58] Souffriau, W., Vansteenwegen, P., Berghe, G. V., Van Oudheusden, D. A path relinking approach for the team orienteering problem. *Computers & operations research 37*, 11 (2010), 1853–1859.

[59] Souza, M., Coelho, I., Ribas, S., Santos, H., Merschmann, L. A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research 207*, 2 (2010), 1041–1051.

[60] Subramanian, A., Drummond, L., Bentes, C., Ochi, L., Farias, R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research 37*, 11 (2010), 1899–1911.

[61] Tang, H., Miller-Hooks, E. A tabu search heuristic for the team orienteering problem. *Computers & Operations Research 32*, 6 (2005), 1379–1407.

[62] Thangiah, S., Fergany, A., Awan, S. Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research 15*, 4 (2007), 329–349.

[63] Toth, P., Vigo, D. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing 15*, 4 (2003), 333–346.

[64] Tsiligirides, T. Heuristic methods applied to orienteering. *The Journal of the Operational Research Society 35*, 9 (1984), 797–809.

[65] Vansteenwegen, P., Souffriau, W., Berghe, G. V., Van Oudheusden, D. A guided local search metaheuristic for the team orienteering problem. *European journal of operational research 196*, 1 (2009), 118–127.

[66] Vansteenwegen, P., Souffriau, W., Berghe, G. V., Van Oudheusden, D. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research 36*, 12 (2009), 3281–3290.

[67] Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., Rei, W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research 60*, 3 (2012), 611–624.

[68] Vidal, T., Crainic, T. G., Gendreau, M., Prins, C. Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. *European Journal of Operational Research 231*, 1 (2013), 1–21.

[69] Vidal, T., Crainic, T. G., Gendreau, M., Prins, C. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research 234*, 3 (2014), 658 – 673.

[70] Vidal, T., Maculan, N., Ochi, L. S., Vaz Penna, P. H. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science 50*, 2 (2015), 720–734.

[71] Vilhelmsen, C., Larsen, J., Lusby, R. M. *Tramp Ship Routing and Scheduling - Incorporating Additional Complexities*. PhD thesis, 2014.