HYBRID ALGORITHMS FOR SOLVING ROUTING PROBLEMS

Daniel Guimarans Serrano

PhD Thesis

Advisors
Dr. Juan José Ramos González
Dr. Daniel Riera i Terrén

Departament de Telecomunicació i d'Enginyeria de Sistemes
Escola d'Enginyeria
Universitat Autònoma de Barcelona

June 2012

Dr. Juan José Ramos González, Associate Professor at the Universitat Autònoma de Barcelona, and

Dr. Daniel Riera i Terrén, lecturer at the Universitat Oberta de Catalunya

CERTIFY:

That the thesis entitled **Hybrid Algorithms for Solving Routing Problems** and submitted by **Daniel Guimarans Serrano** in partial fulfillment of the requirements for the degree of Doctor, embodies original work done by him under their supervision.

Dr. Juan José Ramos González
Thesis Director

Dr. Daniel Riera i Terrén
Thesis Director

Dpt. Telecomunicació i d'Enginyeria de Sistemes
Escola d'Enginyeria
Universitat Autònoma de Barcelona
June 2012

Contents

Chapter 1

Introduction

The growing flows of freight have been a fundamental component of contemporary changes in economic systems at the global, regional, and local scales. Road transportation is nowadays the predominant way of transporting goods in many parts of the world. Direct costs associated with road transportation have experienced a significant increase in the last decade due to the rise of oil price[1], among other economical factors. Furthermore, road transportation faces new challenges related to other indirect or external related costs, which usually are easily observable —noise, pollution, accidents, etc.— but difficult to quantify. The role of transport and logistics as an economic sector cannot be nowadays neglected since new modes of production are concomitant with new modes of distribution. Achieving flexible, efficient, and sustainable routing is a complex strategy requiring a high level of logistical integration to properly respond to variations of the freight transport demand. The necessity for optimizing road transportation affects to both the public and the private sectors, and constitutes a major challenge for most industrialized regions.

An important component of many distribution systems is routing vehicles to serve customers. Many companies are confronted daily with problems regarding the transportation of people, goods or information. These companies have to optimize transportation by using rational manners and effective tools. The *Vehicle Routing Problem* (VRP) provides a theoretical framework for approaching this class of logistic problems dealing with physical distribution. This is among the most popular research lines in combinatorial optimization.

[1] According to the World Bank Commodity Price Data, the annual price of a Brent barrel has risen from $28.27 in 2000 to $110.94 in 2011, a 292.43 % increase.

It was first defined by Dantzig and Ramser [51] in 1959, and several variants of the basic problem have been proposed and studied later. These variants represent different types of operational constraints such as, for instance, time windows, pick-up and delivery, heterogeneous fleets, or multi-depot problems.

The interest in VRP problems comes from its practical relevance as well as from the considerable difficulty to solve them exactly. In the field of combinatorial optimization, the VRP is regarded as one of the most challenging problems because of its NP-Hardness [158], meaning that it is not solvable in polynomial time. For such problems in real situations, it is often desirable to obtain approximate solutions, so they can be found fast enough and are sufficiently accurate for the purpose.

From the industrial applicability perspective, the VRP characterizes a family of different distribution problems which, one way or another, are present in real industrial problems. Nevertheless, in most of the application cases none of the classical VRP variants can represent uniquely the real problem, i.e. a combination of different operational constraints are present in realistic cases. In this scenario, it becomes evident the need of developing new flexible methods, models, and systems to give support to the decision making process so that optimal strategies can be chosen in physical distribution and, in particular, in road transportation.

During the last fifty years, the VRP has generated an intense research related to exact and heuristic methods. The interest about hybrid optimization methods has grown very fast for the last few years [167] [90]. Hybridization has become a very promising strategy in designing and developing improved metaheuristic solution methods, because of their heuristic nature, greater flexibility, and less strict mathematical formulation. A hybrid metaheuristic method combines structure and efficiency advantages from different principles and approaches, while reducing the effects of their limitations. Thus, hybrid methods often provide a highly flexible and efficient tool in solving difficult combinatorial optimization problems.

This thesis is aimed to introduce three different yet related hybrid methodologies to solve the VRP. These methodologies have been especially designed for being flexible in the sense that they can be used, with minor adaptations, for solving different variants of the VRP present in industrial application cases.

In the three methodologies described in this work, different technologies are used to achieve the desired flexibility, efficiency, and robustness. *Constraint Programming* (CP) has been chosen as the modeling paradigm to describe

the main constraints involved in the VRP. CP provides the pursued flexibility for the three methodologies, since adding side constraints present in most real application cases becomes a modeling issue and does not affect the search algorithm definition. In the first two hybrid methodologies, the CP model is used to check solution's feasibility during search. The third methodology presents a richer model for the VRP capable of tackling different problem variants. In this case, the search is performed and controlled from a CP perspective.

Lagrangian Relaxation (LR) and a probabilistic version of the classic *Clarke and Wright Savings* (CWS) heuristic are used for specific purposes within the proposed methodologies. The former is used for minimizing the total traveled distance and the latter to provide a good initial solution quickly. Both methods provide an efficient approach to the respectively faced problems. Moreover, the use of LR permits reducing the computational complexity of the performed local search processes and therefore reduces the required computational time to solve the VRP.

All methodologies are based on the so-called *Variable Neighborhood Search* (VNS), a quite recent metaheuristic introduced for the first time by Mladenovic and Hansen [121]. The VNS is formed by a family of algorithms which exploits systematically the idea of neighborhood changes both in the search phase to find a local minimum, and in perturbation phase, to escape from the corresponding valley. Although it is an extended method, there are few examples of its application to the VRP. However, interesting results have been obtained even applying the simplest VNS algorithms to this problem.

The present thesis is aimed to contribute to the current research on the application of the VNS metaheuristic to the VRP. It has been chosen as the framework where the mentioned techniques are embedded. Hence, the metaheuristic is used to guide the search, while the desired efficiency is provided by the composing methods. On the other hand, using CP as the modeling paradigm provides the required flexibility. This characteristic is enhanced in the last described methodology. In this case, the CP search is guided by a combination of the VNS and the *Large Neighborhood Search* (LNS) metaheuristics. This methodology represents an initial approach for tackling efficiently more complex and richer VRP, similar to real application cases.

1.1 Objectives

The objectives of this thesis are:

- The development of a hybrid methodology aimed to tackle the VRP based on the VNS metaheuristic framework. This methodology should be flexible, efficient, and robust.
- The study and implementation of different mechanisms and strategies to enhance methodology's performance, such as reducing the size of the neighborhoods to be explored during search and the use of efficient heuristics in different processes.
- The integration of the developed methodology into a parallelized calculation environment to improve its efficiency and competitiveness.
- The study of different VRP variants from a CP perspective and the development of a complete and extendable model, seed of a VRP library based on the CP paradigm.
- The development of a CP-based search methodology based on the VNS and LNS metaheuristics able to tackle small and medium-sized VRP. This methodology is aimed to provide a first approach to the VRP combining these technologies within a CP search environment, since CP-based exact methods have a limited application due to the required computational times.
- The study and implementation of different strategies and heuristics to improve CP-based methodology's performance.
- The assessment of the developed methodologies by their application to different VRP benchmark sets.

1.2 Structure of this Thesis

The current chapter introduces the thesis with a brief description of the research context and the objectives of this work. The remainder of this thesis is structured as follows. Chapter 2 describes the VRP and its variants. It also includes a review of the state of the art for this problem including both

exact and heuristic methods. Chapter 3 presents a background of the different technologies used in this work. Chapter 4 introduces the foundations and characteristics of the three methodologies developed. This chapter includes the two formulations used for the VRP, as well as the algorithms and implemented improvements. Chapter 5 assesses these methodologies by means of different benchmark problems. This work is also positioned in the VRP context by comparing the obtained results with other state-of-art methodologies. Finally, Chapter 6 presents the conclusions of this thesis, its contributions, and the possible future research lines.

Chapter 2

The Vehicle Routing Problem

Vehicle Routing Problem (VRP) is a generic name given to a whole class of problems involving the design of optimal routes for a fleet of vehicles to service a set of customers subject to side constraints. Collection of household waste, gasoline delivery trucks, goods distribution, and mail delivery are some examples of the wide number of real-life applications of the VRP. Thus, the VRP is considered to play a central role in distribution and logistics.

In practice, several variants of the VRP exist, depending on the nature of the transported goods, the quality of service required, and the characteristics of customers and vehicles. In all cases, the objective is to supply the customers at minimum cost. Some typical complications are heterogeneous vehicles located at one or several depots, customers incompatible with certain vehicles types, customers accepting being serviced within specified time windows, multiple-day or periodic planning horizons and vehicles performing multiple routes. According to these side constraints, several variants of the VRP can be defined:

- **Capacitated VRP (CVRP)**: the CVRP is a VRP in which a fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. That is, CVRP is like VRP with the additional constraint that vehicle's capacity is limited, which makes the vehicle periodically return to the depot for reloading. As the capacity constraint always exists in any kind of VRP problems, the CVRP is considered to be the basics, to which all other constraints are added.
- **VRP with Time Windows (VRPTW)**: the VRPTW is the same problem that the VRP with the additional restriction that in the VRPTW

a time window is associated with each customer, defining an interval wherein it has to be supplied. The interval at the depot is called the scheduling horizon. Specific examples of problems with time windows include bank or postal deliveries, industrial refuse collection, school-bus routing and situations where the customer must provide access, verification, or payment upon delivery of the product or service. In these problems, customers can be served only during certain hours of the day, such as office hours or the hours before the opening of a shop.

- **Multiple Depot VRP (MDVRP)**: a company may have several depots from which it can serve its customers. If the customers are clustered around depots, then the distribution problem may be modeled as a set of independent VRPs with a single depot. However, if the customers and the depots are intermingled then a MDVRP should be solved. A MDVRP requires the assignment of customers to depots. A fleet of vehicles is based at each depot. Each vehicle originates from one depot, services the assigned customers, and returns to the same depot. The objective of the problem is to service all customers while minimizing the number of vehicles and traveled distance.

- **VRP with Pick-up and Delivery (VRPPD)**: the VRPPD is a VRP in which some goods should be delivered from one set of customers to another. That means that all vehicles start their trip from the depot empty, pick up some commodities from definite customers and deliver it to others according to the orders. After all orders are satisfied, vehicles return to the depot empty. Usually there is a restriction, that all customers should be served by exactly one vehicle. The main difficulty in dealing with this problem is to find the correct order of pick-up and delivery operations, because these restrictions makes the planning problem more difficult and can lead to bad utilization of the vehicles capacities, increased travel distances or a need for more vehicles. It can be noted that the CVRP and the VRPTW are particular cases of the VRPPD in which either all origins or all destinations are located at the common depot.

- **VRP with Backhauls (VRPB)**: the VRPB is a VRP in which customers can demand or return some commodities. So in VRPB it is needed to take into account that the goods that customers return to the deliver vehicle must fit into it. The critical assumption is that all deliv-

eries must be made on each route before any pick-ups can be made. This arises from the fact that the vehicles are rear-loaded, and rearrangement of the loads on the trucks at the delivery points is not deemed economical or feasible. The quantities to be delivered and picked up are fixed and known in advance.

- **Split Delivery VRP (SDVRP)**: SDVRP is a relaxation of the VRP wherein it is allowed that the same customer can be served by different vehicles if it reduces overall costs. This relaxation is very important if the size of customers' orders is as big as the capacity of a vehicle.

- **Stochastic VRP (SVRP)**: SVRP are VRPs where one or several components of the problem are not deterministic. Different SVRPs can be originated according to stochastic variables: customers may be present with a certain probability, random demands, random service and travel times, etc. In the SVRP, two stages are made for getting a solution. A first solution is determined before knowing the realizations of the random variables. In a second stage, a recourse or corrective action can be taken when the values of the random variables are known.

- **Periodic VRP (PVRP)**: In classical VRPs, typically the planning period is a single day. In the case of the PVRP, the classical VRP is generalized by extending the planning period to a certain number of days.

The most basic VRP is the CVRP, that assumes a fixed fleet of vehicles of uniform capacity housed in a central depot. It is intrinsically a spatial problem with some capacity constraints. In addition to the geographic component, more realistic routing problems include a scheduling part by incorporating travel times between every pair of nodes, customer service times and the maximum tour duration as additional problem data. The VRPTW is an extension of the CVRP with the further complexity of time windows and other time data. In the VRPTW problem, each customer has an associated time window defined by the earliest and the latest time to start the customer service. The depot may also have a time window defining the scheduling horizon. Time windows can be hard or soft. In the hard time window case, a vehicle arriving too early at the customer site is permitted to wait until the customer window is open. However, a vehicle is not permitted at all to arrive at the node after the latest service start time. In contrast, the soft time window case permits time

window violations at the expense of a penalty cost. The VRPPD with Time Windows (PDTW) is a further extension of the VRPTW. In the PDTW, pairs are defined among customers so pick up and drop off locations are determined, in addition to time windows constraints related to each visit.

The symmetric CVRP can be considered as a complete undirected graph $G = (I, E)$, connecting the vertex set $I = \{1, 2, ..., n\}$ through a set of undirected edges $E = \{(i, j) \mid i, j \in I\}$. The edge $e_{ij} \in E$ has associated a travel cost c_{ij}, supposed to be the lowest cost route connecting node i to node j. Each vertex $i \in I \setminus \{1\}$ has a nonnegative demand q_i, while vertex 1 corresponds to a depot without associated demand. A fixed fleet of m identical vehicles, each of capacity Q, is available at the depot to accomplish the required tasks. Solving the CVRP consists of determining a set of m routes whose total travel cost is minimized and such that: (i) each customer is visited exactly once by a single vehicle, (ii) each route starts and ends at the depot, and (iii) the total demand of the customers assigned to a route does not exceed the vehicle capacity. Therefore, a solution to the CVRP is a set of m cycles sharing a common vertex at the depot. In some cases, the fleet size is not fixed and minimizing the total number of used vehicles becomes an additional objective.

The VRPTW extends the CVRP by associating a travel time t_{ij} to each edge $e_{ij} \in E$. Each vertex $i \in I \setminus \{1\}$ has a time demand t_i required to perform the service, which should start within a defined time window $[a_i, b_i]$. The primary objective of the VRPTW is to find the minimum number of routes, i.e. use the minimum number of vehicles. A secondary objective is imposed to minimize the total cost of routes, that can be expressed in terms of the total traveled distance or the total scheduled time.

Complexity theory provides a mathematical framework in which combinatorial problems such as the VRP can be studied so they can be classified as "easy" or "hard" to solve. In order to determine if a combinatorial problem is easy or hard, we study the computational resources (time and memory capacity) required to solve the problem as a function of the size of the considered instance S, denoted by $|S|$. For optimization problems, solving means finding a solution, for any given instance, that is optimal in terms of the objective function, and for decision problems solving is deciding correctly, for any given instance, whether or not the instance has at least one feasible solution. A problem is considered to be "easy" if there exists an algorithm that solves the problem in time bounded by a polynomial function of $|S|$. If a decision problem is easy, then, under a mild restriction on the range of the objective function, one can apply binary search to obtain a polynomial time algorithm

for the optimization problem. On the other hand, if an optimization problem is easy, then clearly the decision version of this problem is also easy. Let P denote the class of decision problems that can be solved in polynomial time.

Let NP denote the class of decision problems that have the property that for every instance S that has a feasible solution, there exists a *certificate* y such that $|y|$ is bounded by a polynomial in $|S|$ and such that it can be checked in polynomial time that y is indeed a certificate for a feasible solution of S. Notice that the membership of NP does not mean that it is easy to find such a certificate.

The class of NP contains an enormous number of problems, including all problems in P. Many problems in NP are not known to be solvable in polynomial time, however. It is not known whether P equals NP, but it is widely conjectured that this is not the case [65].

We can compare the complexity of two decision problems in NP by *reducing* one problem to another. We say that $\prod_1$ reduces to $\prod_2$ ($\prod_1 \propto \prod_2$) if there is a polynomial transformation from every instance of $\prod_1$ to an equivalent instance of $\prod_2$. In this context, equivalent means that the instance of $\prod_2$ is feasible iff the corresponding instance in $\prod_1$ is feasible. In this case, the existence of a polynomial algorithm to solve $\prod_2$ implies the existence of a polynomial algorithm for $\prod_1$. The existence of such a polynomial transformation shows that $\prod_1$ can be studied as a special case of $\prod_2$, or that $\prod_2$ is at least as hard as $\prod_1$.

A problem $\prod$ is said to be *NP-hard* if there exists a polynomial reduction from every problem in NP to $\prod$. If, in addition, $\prod \in NP$, the problem is said to be *NP-complete*. If an NP-complete or NP-hard problem would be solvable in polynomial time, then all problems in NP would be solvable in polynomial time, implying $P = NP$. Hence, for an NP-complete or NP-hard problem, a polynomial time algorithm is unlikely to exist.

Notice that once at least one problem $\prod_1$ is known to be NP-complete, showing NP-completeness of another problem $\prod_2$ requires only showing that $\prod_2 \in NP$, and that $\prod_1 \propto \prod_2$. Cook [47] provided the first NP-completeness proof for a problem, by giving a master reduction from every problem in NP to it.

Almost all vehicle and scheduling problems are NP-hard and hence unlikely to be solvable in polynomial time [109]. The general VRP belongs to the NP-hard problems class. When additional constraints are introduced in the problem, e.g. capacity constraints, time windows, maximum number of cus-

tomers per route, limited fleet, etc., this constrained VRP can be demonstrated to be NP, and so it is NP-complete [158].

2.1 State of the Art

Since it was first defined by Dantzig and Ramser in [51] as a generalization of the *Traveling Salesman Problem* (TSP), the VRP has attracted the attention of many researchers. The most studied variants are the most basic VRP, the CVRP and the VRPTW, because of their high complexity level and its wide applicability to real situations. Both problems have generated an intense research related to exact and heuristic methods during the last fifty years. In this section, we provide a summary of some of the most significant works. For further reading on both exact and heuristic methods for these VRPs, the reader is referred to the exhaustive surveys [174] [35] [36] [114] [164] [19] [18].

2.1.1 Exact approaches

Currently, the most successful exact methods for the VRP are based on the *two-index flow formulation*, the *two-commodity flow formulation*, and the *set partitioning formulation*. Valid lower bounds on the VRP can be derived from the Linear Programming (LP) relaxations of these mathematical formulations. Some of the resulting LP programs cannot be solved directly, even for moderate size VRPs, since either the number of variables or constraints is exponential in the problem size. Thus, the lower bounds are usually computed using *cutting plane* and *column generation* techniques. In addition, to strengthen the relaxations, a variety of valid inequalities have been described in the literature for the different formulations. An extended explanation of these inequalities can be found in [19].

The *branch-and-cut* (BC) algorithms for the CVRP are based either on the *two-index flow* or the *two commodity flow* formulation. Augerat et al. [10] were the first to describe an exact BC algorithm for the CVRP based on the two-index flow formulation, originally proposed by Laporte et al. [105], strengthened by valid inequalities such as the *generalized capacity constraints*, *hypotour inequalities*, *comb inequalities*, and *path-bin inequalities*. The BC algorithm of Augerat et al. [10] was able to solve, for the first time, a CVRP instance involving 135 customers. Naddef and Rinaldi [125] revisited the BC algorithm by Augerat et al. [10] and presented an improved version.

Ralphs et al. [143] described a BC algorithm based on the two-index flow formulation and on the addition of rounded capacity constraints in a cutting plane fashion.

Lysgaard et al. [116] described a BC algorithm based on the two-index flow formulation, strengthened by valid inequalities, including the *rounding capacity*, *generalized capacity*, *framed capacity*, *strengthened comb*, *multistar*, *partial multistar*, *extended hypotour inequalities*, and *Gomory mixed integer cuts*. Their BC algorithm solved several instances not solved by Augerat et al. [10].

Baldacci et al. [16] proposed a two-commodity flow formulation of the CVRP which extends the TSP model introduced by Finke et al. [63]. The BC algorithm based on this model uses rounded capacity inequalities in a cutting plane fashion to strengthen the lower bound obtained by the LP relaxation of the two-commodity formulation. The reported computational results show that this BC algorithm is competitive with the algorithm of Naddef and Rinaldi [125].

The *Set Partitioning* formulation of the CVRP originally proposed in [20] associates a binary variable with each feasible route. It cannot be used directly to solve nontrivial CVRP instances because of the large number of potential routes. The set partitioning model is very general and can take into account several route contraints, e.g. time windows, because the route feasibility is implicitly considered in the definition of the route set.

Christofides et al. [45] introduced the concept of *q-routes*. A *(q,i)*-path is a nonnecessarily elementary path, starting from the depot, visiting a set of vertices (without loops) of total demand q, and ending at vertex i, whose cost can be computed by using dynamic programming. A *q-route* is a *(q,0)*-path. Fukasawa et al. [66] described a *branch-and-cut-and-price* (BCP) algorithm based on the set partitioning model where the variables correspond to the set of *q-routes*, while the constraints correspond to the original set partitioning constraints and inequalities designed for the two-index formulation, such as *rounded capacity inequalities*, *framed capacity*, *strengthened comb*, *multistar*, *partial multistar*, *generalized large multistar*, and *hypotour inequalities*, all presented in Lysgaard et al. [116].

Baldacci et al. [15] proposed a BCP algorithm based on the set partitioning formulation, including *strengthened capacity inequalities* and *clique inequalities*. This method is based on a bounding procedure that computes a lower bound on the CVRP by finding a near-optimal solution of the dual of the LP

relaxation. They propose to solve the LP relaxation using three *column generation* procedures, that produce three lower bounds corresponding to the costs of three different dual solutions of the relaxed problem. The three procedures are executed in sequence, and the dual solution produced by one procedure is used to generate the master problem of the next procedure. In practice, the third procedure requires few iterations to converge to an optimal dual solution of the LP relaxation.

Several exact algorithms have been presented for the VRPTW. A review of the exact methods for this problem is reported in [18] and [96]. The best exact methods recently published on the VRPTW are based on the set partitioning model, where the route set contains any least-cost route satisfying time windows constraints.

The set partitioning model can be strengthened by any valid inequality studied for the CVRP, and by the *k-path inequalities* introduced by Kohl et al. [101]. These inequalities can be considered as a generalization of the rounded capacity constraints. Other valid inequalities, related to *k-path* inequalities and called *Reachability cuts*, have been investigated by Lysgaard [115].

The exact algorithms for solving the resulting set partitioning model of the VRPTW use *column generation* methods for computing the lower bound and either *branch-and-price* (BP) or BCP algorithms to find an optimal integer solution. The key component of these algorithms is the method for solving the pricing problem. This algorithm consists of finding a number of VRPTW routes of negative reduced cost with respect to the duals of the set partitioning constraints, and of the different inequalities separated during the previous iterations. This problem is solved using different dynamic programming strategies to find either nonelementary or elementary routes.

Kohl et al. [101] described a BP algorithm which improves the BP of Desrochers et al. [55] by adding 2-path inequalities to the LP relaxation of the set partitioning formulation and by using nonelementary routes in solving the pricing problem. Irnich and Villeneuve [88] improved the BP of Kohl et al. [101] by using a sophisticated *k-cycle* elimination that substantially improves the lower bounds. This elimination proved to be a key ingredient for solving to optimality more than 15 unsolved Solomon [163] instances with 25, 50, and 100 customers. Nonetheless, this method fails to solve several instances with 25 customers.

Algorithms based on the computation of elementary routes were proposed by Feillet et al. [60], Danna and Le Pape [50], and Chabrier [40]. Righini and

Salani [149] proposed a dynamic programming method, called *decremental state-space* algorithm, to solve the pricing problem by forcing the routes to visit the customers of a selected subset at most once.

A significant contribution was given by Jepsen et al. [89], who extended the BCP framework by adding the *Subset-Row inequalities* (SR3) to the set partitioning master problem. The SR3 inequalities provide better lower bounds but increase the complexity of the pricing problem. To reduce the computing time, they attempted to solve the pricing problem heuristically. The computational results indicate that the algorithm of Jepsen et al. [89] outperforms those of Irnich and Villeneuve [88] and Chabrier [40].

The BCP of Jepsen et al. [89] was improved by Desaulniers et al. [54] by adding both SR3 and generalized k-path inequalities and using a tabu search heuristic, before using dynamic programming, to rapidly generate negative reduced cost routes. Their method outperforms all other algorithms, decreasing the computational time on Solomon [163] instances with 100 customers.

Baldacci et al. [17] extended the BCP algorithm of Baldacci et al. [15] to solve both the CVRP and the VRPTW. They introduce a new route relaxation called *ng-route*, used by different dual ascent heuristics to find near-optimal dual solutions of the LP relaxation of the set partitioning model. They describe a *column-and-cut generation* algorithm strengthened by different valid inequalities, SR3 [89] and *Weak Subset-Row (WSR3) inequalities*, and a new pricing strategy involving multiple dual solutions. This method significantly improves some results and the running times of the other algorithms for both CVRP [116] [66] [15] and VRPTW [89] [54].

2.1.2 Heuristic and Metaheuristic methods

The *Clarke and Wright's Savings* (CWS) constructive algorithm [46] is probably the most cited heuristic to solve the CVRP. The CWS is an iterative method that starts out by considering an initial dummy solution in which each customer is served by a dedicated vehicle. Next, the algorithm initiates an iterative process for merging some of the routes in the initial solution. Merging routes can improve the expensive initial solution so that a unique vehicle serves the nodes of the merged route. The merging criterion is based upon the concept of savings. Given a pair of nodes to be served, a savings value can be assigned to the edge connecting these two nodes. This savings value is given by the reduction in the total cost function due to serving both

nodes with the same vehicle instead of using a dedicated vehicle to serve each node, as proposed in the initial dummy solution. This way, the algorithm constructs a list of savings, one for each possible edge connecting two demanding nodes. At each iteration of the merging process, the edge with the largest possible savings is selected from the list as far as the following conditions are satisfied: (a) the nodes defining the edge are adjacent to the depot, and (b) the two corresponding routes can be feasibly merged, i.e. the vehicle capacity is not exceeded after the merging. The CWS algorithm usually provides relatively good solutions, especially for small and medium-size problems, but it also presents difficulties in some cases. Many variants and improvements of the CWS have been proposed in the literature. For a comprehensive discussion on the various CWS variants, the reader is referred to Toth and Vigo [174] and Laporte [104].

Monte Carlo Simulation (MCS) can be defined as a set of techniques that make use of random numbers and statistical distributions to solve certain stochastic and deterministic problems [108]. MCS has proved to be extremely useful for obtaining numerical solutions to complex problems that cannot be efficiently solved by using analytical approaches. Buxey [38] was probably the first author to combine MCS with the CWS algorithm to develop a procedure for the CVRP. This method was revisited by Faulin and Juan [59], who introduced an entropy function to guide the random selection of nodes. MCS has also been used by Fernández de Córdoba et al. [52] and Juan et al. [91][94][93] to solve the CVRP. In this last paper, the authors make use of MCS to develop an efficient randomized version of the CWS heuristic, which we use in our approach to efficiently generate initial solutions. The authors have later improved the algorithm by introducing efficient computing techniques, intelligent solutions splitting and optimal routes tracking [92].

The VRPTW has also been the subject of intensive research efforts for heuristic approaches. The first example of a route construction heuristic for the VRPTW was given by Solomon [162]. A route construction heuristic select nodes (or arcs) sequentially until a feasible solution has been created. Nodes are chosen based on some cost minimization criterion, often subject to the restriction that the selected nodes do not create a violation of vehicle capacity or time window constraints.

Solomon [162] proposed a route-first cluster-second scheme using a giant-tour heuristic. First, the customers are scheduled into one giant tour, which is then divided into a number of smaller routes. The initial giant tour is often generated by considering the problem as a TSP, i.e. without considering the

capacity and time constraints.

Solomon [163] describes several heuristics for the VRPTW. One of the methods is an extension to the CWS heuristic with a waiting time limit to account for both the spatial and temporal closeness of customers. The second heuristic, a time-oriented nearest neighbor, starts every route by finding an unrouted customer closest to the depot. At every subsequent iteration, the heuristic searches for the customer closest to the last customer added into the route and adds it at the end of the route. A new route is started any time the search fails to find a feasible insertion place, unless there are no more unrouted customers left. Again, the metric used to measure the closeness of any pair of customers attempts to account for both geographical and temporal closeness of customers. Solomon [163] proposed three different insertion criteria: (i) *I1*, based on a time insertion criterion; (ii) *I2*, aiming to select customers whose insertion costs minimize a measure of total route distance and time; and (iii) *I3*, which accounts for the urgency of servicing a customer. Albeit the most successful of the three proposed insertion heuristics is *I1*, Dullaert [57] and Dullaert and Bräysy [58], argue that Solomon's time insertion criterion understimates the additional time needed to insert a new customer between the depot and the first customer in the partially constructed route. The authors introduce new time insertion criteria to solve this problem.

Solomon [163] also describes a time-oriented *sweep* heuristic based on the idea of decomposing the problem into a clustering stage and a scheduling stage. In the first phase, customers are assigned to vehicles as in the original sweep heuristic [75]. In the secong phase, customers assigned to a vehicle are scheduled using an insertion heuristic of type *I1*.

Potvin and Rousseau [138] introduce a parallel version of Solomon's insertion heuristic *I1*, where the set of m routes is initialized at once. Russell [157] embeds global tour improvement procedures within the tour construction process. The construction procedure used is similar to that in Potvin and Rousseau [138].

Ioannou et al. [87] use the generic sequential insertion framework proposed by Solomon to solve a number of theoretical benchmark problems and an industrial example from the food industry. The proposed approach is based on new criteria for customer selection and insertion that are motivated by the minimization function of a greedy look-ahead heuristic.

Balakrishnan [14] describes three heuristics for the VRP with soft time windows. The heuristics are based on nearest neighbor and CWS rules, and

they differ only in the way used to determine the first customer on a route and in the criteria used to identify the next customer for insertion. The motivation behind the use of soft time windows is that by allowing limited time window violations for some customers, it may be possible to obtain significant reductions in the number of vehicles required and the total distance or time of all routes. Among the soft time window problem instances, *dial-a-ride* problems play a central role [48].

In addition to construction heuristics, classical local search methods have also been used to tackle the VRPTW. The edge-exchange neighborhoods for a single route are the set of tours that can be obtained from an initial tour by replacing a set of k of its edges by another set of k edges. Such replacements are called k-exchanges, and a tour that cannot be improved by a k-exchange is said to be k-optimal. Russell [156] reports early work on the VRPTW for a k-optimal improvement heuristic.

Prosser and Shaw [141] extend to the VRPTW some intraroute and interroute operators defined for the CVRP. The intraroute operator is the *2-opt* by Lin [112], while an extension of the *relocate*, *exchange*, and *cross* operators originally proposed by Savelsbergh [158] are used as interroute operators. De Backer et al. [12] report research similar to Prosser and Shaw [141] in the *Constraint Programming* (CP) context. Other frequently applied neighborhood operators for the VRPTW are the λ-interchange of Osman [130], the *CROSS*-exchange of Taillard et al. [166], the *GENI*-exchange of Gendreau et al. [72], and *ejection chains* [76]

Shaw [160] describes a *Large Neighborhood Search* (LNS) based on rescheduling selected customer visits using CP techniques. The search operates by choosing in a randomized fashion a set of customer visits. The selected customers are removed from the schedule, and then reinserted at optimal cost. To create opportunity for interchange of customer visits between routes, the removed visits are chosen so that they are related. A branch-and-bound method coupled with CP is then used to reschedule removed visits. Due to high computational requirements, this approach can be applied only to problems where the number of customers per route is relatively low.

Shaw [161] proposes a similar LNS approach which uses constraint-based *limited discrepancy search* in the reinsertion of customers within the branch-and-bound procedure. The number of visits to be removed is increased during the search each time a number of consecutive attempted moves have not resulted in an improvement of the cost. Limited discrepancy search is used to explore the search tree in order of an increasing number of discrepancies, a

discrepancy being a branch against the best reinsertion places, e.g. inserting a customer at its second cheapest position.

Bent and Hentenryck [23] describe a LNS heuristic for the VRPTW. The authors propose to solve the problem in a two-stage approach. In the first stage the number of routes is minimized by a *Simulated Annealing* (SA) [129] algorithm that uses traditional, small neighborhoods. SA is a stochastic relaxation technique, which has its origin in statistical mechanics. It is based on an analogy from the annealing process of solids, where a solid is heated to a high temperature and gradually cooled for it to crystallize in a low energy configuration. In the second stage of Bent and Hentenryck [23] approach, the total route lengths are minimized with an LNS heuristic. The size of the neighborhood is gradually increased, starting out by only removing one customer and by steadily increasing the number of customers to remove as the search progresses. At regular intervals, the number of customers to remove is reset to one and the neighborhood size increase starts over. The repair method is implemented using a truncated branch-and-bound algorithm. The LNS algorithm only accepts improving solutions. A similar algorithm was also proposed by the same authors for the Pickup and Delivery Problem with Time Windows (PDTW) [24].

Ropke and Pisinger [151] introduce the *Adaptive Large Neighborhood Search* (ALNS) extension of the LNS. The algorithm is applied to the PDTW. Differences with the method in [23] are: (i) several different destroy/repair methods are used, (ii) fast, greedy heuristics are used as repair methods, (iii) the size of the neighborhoods varies from iteration to iteration (the number of customers to remove is chosen randomly from a predefined interval), and (iv) a SA acceptance criterion is used. In subsequent papers [135] [152] it is shown that many VRP variants (including the CVRP and VRPTW) can be transformed to a PDTW and solved using an improved version of the ALNS heuristic from [151]. This unified model can be seen as a *Rich Pickup and Delivery Problem with Time Windows*.

Prescott-Gagnon et al. [139] present an LNS heuristic for the VRPTW with an advanced repair operator that solves a restricted VRPTW through a heuristic branch-and-price algorithm. Four destroy methods are used and are chosen based on performance as in [151]. Overall, the heuristic reaches better solutions than previous LNS approaches, probably due to the advanced repair operator.

Nagata and Bräysy [127] present an efficient heuristic based on the idea of the *ejection pool* [111], combined with *Guided Local Search* (GLS) [177]

to guide the ejections. The authors incorporate an insertion procedure that accepts temporal infeasible insertions, followed by an attempt to restore the feasibility. Results demonstrate that this algorithm is competitive with other state-of-the-art heuristic approaches, such as those by Pisinger and Ropke [135] and Prescott-Gagnon et al. [139].

Using constructive heuristics as a basis, metaheuristics became popular for the CVRP during the 90s. *Tabu Search* (TS) is a local search metaheuristic that explores the solution space by moving at each iteration from a solution s to the best solution in a subset of its neighborhood $N(s)$ [71]. Contrary to classical descent methods, the current solution may deteriorate from one solution to the next. Accepting worse solutions insures new regions of a problem's solution space are investigated with the goal of avoiding local minima. To avoid cycling, solutions possessing some attributes of recently explored solutions are temporarily declared *tabu*, i.e. forbidden. The duration that an attribute remains tabu is called its *tabu tenure*, and it can vary over different intervals of time. The tabu status can be overridden if certain conditions are met. This is called the *aspiration criteria*, and it happens, for example, when a tabu solution is better than any previously seen solution. Finally, various techniques are often employed to diversify or to intensify the search process.

Some early examples of the application of TS metaheuristic to the CVRP are the *Taburoute* method by Gendreau et al. [73] or the *Boneroute* method of Tarantilis and Kiranoudis [171]. TS algorithms, like those proposed by Taillard [165] or Toth and Vigo [175], are among the most cited metaheuristics.

Garcia et al. [68] were the first to apply TS to the VRPTW. The TS they developed is a fairly simple one, involving Solomon's *I1* [163] insertion heuristic to create an initial solution 2*-opt and Or-opt exchanges for improvement. Many authors since that time have presented numerous TS implementations involving sophisticated diversification and intensification techniques, explicit strategies for minimizing the number of routes, complex post-optimization techniques, hybridizations with other search techniques such as SA and *Genetic Algorithms* (GA) [144], parallel implementations, and allowance of infeasible solutions during the search. The initial solution is typically created with some cheapest insertion heuristic, being Solomon's *I1* [163] the most common one. After creating an initial solution, an attempt is made to improve it using local search with one or more neighborhood structures (e.g. 2-opt, Or-opt, relocate, exchange, CROSS-, GENI-, and λ- exchanges) and a best-accept strategy.

To reduce the complexity of the search, some authors propose special strategies for limiting the neighborhood. For instance, Garcia et al. [68] only allow

moves involving arcs that are close in distance. Taillard et al. [166] decompose solutions into a collection of disjoint subsets of routes by using the polar angle associated with the center of gravity of each route. TS is then applied to each subset separately. Another frequently used strategy to speed up the search is to implement the proposed algorithm in parallel on several processors. For instance, Badeau et al [13] apply the solution approach of Taillard et al. [166] using a two-level parallel implementation. On the other hand, to cross the barriers of the search space, created by time window constraints, some authors allow infeasibilities during the search. For instance, Brandão [32], Cordeau et al. [49], and Lau et al. [107] allow violation of each constraint type (load, duration, and time window constraints). The violations of constraints are penalized in the cost function, and the parameter values regarding each type of violation are adjusted dynamically.

Because the number of routes is often considered as the primary objective, some authors use different explicit strategies for reducing the number of routes. For example, the algorithms of Garcia et al. [68] and Potvin et al. [137] try to move customers from routes with a few customers into other routes using Or-opt exchanges. Similarly, the method of Schulze and Fahle [159] tries to eliminate routes having at most three customers by trying to move these customers into other routes. In Lau et al. [107] a limit is set for the number of routes that cannot be exceeded during the search.

Most of the proposed TS use specialized diversification and intensification strategies to guide the search. For instance, Rochat and Taillard [150] propose using a so-called *adaptive memory*. The adaptive memory is a pool of routes taken from the best solutions visited during the search. Its purpose is to provide new starting solutions for the TS through selection and combination of routes extracted from the memory. The selected tours are improved using TS and inserted subsequently back into the adaptive memory. Tan et al. [170] diversify the search each time a local minimum is found by performing a series of random λ-interchange hops combined with the 2^*-opt operator. A candidate list is maintained to record elite solutions discovered during the search process. These elite solutions are then used as a starting point for intensification. Lau et al. [106] present a generic, constraint-based diversification technique, where VRPTW is modeled as a linear constraint satisfaction problem that is solved by a simple local search algorithm.

Carlton [39] and Chiang and Russell [42] test a reactive TS that dynamically adjusts its parameter values based on the current search status to avoid both cycles as well as an overly constrained search path.

Finally, several authors report using various post-optimization techniques. For example, Rochat and Taillard [150] solve exactly a set partitioning problem at the end, using the routes in the adaptive memory to return the best possible solution. A similar approach is used by Desaulniers et al. [54]. Taillard et al. [166] apply an adaptation of the *GENIUS* heuristic [72] for time windows to each individual route of the final solution. Similarly, in Cordeau et al. [49] the best solution identified after n iterations is post-optimized by applying to each individual route a specialized heuristic for the TSP with time windows.

The most recent TS approach to the VRPTW is provided by Moccia et al. [122]. The authors propose an *Incremental neighborhood TS* (ITS) heuristic. They replace the TS neighborhood structure by one which is exponential in size, but with an evaluation procedure of polynomial complexity.

GA have also played a major role in the development of effective approaches for the VRP. GA are a family of adaptive heuristic search methods based on population genetics [144]. GA evolve a population of individuals encoded as chromosomes by creating new generations of offspring through an iterative process until some convergence criteria are met, e.g. maximum number of iterations or reaching an homogeneous population composed of similar individuals. The best chromosome generated is then decoded, providing the corresponding solution. Population's evolution is mainly driven by three major steps: selection, recombination, and mutation. The *selection* phase consists of randomly choosing two parent individuals from the population. The probability of selecting a population member is generally proportional to its *fitness*, a value given through a *fitness function*, to emphasize genetic quality while maintaining genetic diversity. The *recombination* or reproduction process makes use of genes of selected parents to produce offspring that will form the next generation. As for *mutation*, it consists of randomly modifying some genes of a single individual at a time to further explore the solution space and ensure genetic diversity. The *mutation* is generally associated with a low probability.

An example of the use of GA to solve the CVRP is provided by Berger and Barkaoui [25]. The authors concurrently evolve two populations of solutions to minimize the total traveled distance and to solve a time-variant of the problem that helps on guiding the search while keeping balance between diversification and intensification. Prins [140] proposed a hybrid GA algorithm which substitutes classical mutation operators with a local search process. Mester and Bräysy [119] combined the GLS and evolution strategies into an iterative two-stage procedure. Nagata [126] extended a recombination operator originally designed for the TSP by ignoring vehicles' capacity constraint. To address the

constraint violation, the author introduces a penalty function into the evolutionary algorithm.

Thangiah et al. [173] were the first to apply a GA to the VRPTW. This first paper describes an approach that uses a GA to find good clusters of customers, within a cluster-first route-second problem-solving strategy. During the 90s and beginning of 2000s, numerous papers were written on generating good solutions for the VRPTW with GA. Almost all these papers present hybridizations of a GA with different construction heuristics [29] [27], local searches [172] [136] [95], and other metaheuristics such as TS [100] and *Ant Colony Systems* (ACS) [26].

Homberger and Gehring [85] present two *evolution strategies* for the VRPTW. Together with GA and *evolutionary programming*, the evolution strategies form the class of evolutionary algorithms [34]. By definition, the main differences between these three types of algorithms lie in the representation and in the role of mutation [34]. In Gehring and Homberger [69] the evolution strategies of Homberger and Gehring [85] are hybridized with TS to minimize the total distance, and the approach is parallelized using the concept of *cooperative autonomy*, i.e. several autonomous sequential solution procedures cooperate through the exchange of solutions. Gehring and Homberger [70] introduce three different improvements to the parallel method of Gehring and Homberger [69]. In the evaluation of individuals, capacity related information is used to determine the routes for elimination. In Homberger and Gehring [86], a single processor implementation is presented, and capacity ifnormation is not used in the evaluation criterion. Mester [118] also experimented with evolution strategies similar to Homberger and Gehring [85]. Le Bouthillier and Crainic [31] present a parallel cooperative methodology in which several agents communicate through a pool of feasible solutions. The agents consist of simple construction and local search algorithms, GA and adaptations of the Taburoute method of Gendreau et al. [73].

Nagata et al. [128] develop a penalty-based *Memetic Algorithm* (MA) [124] for the VRPTW by extending the *Edge Assembly Memetic Algorithm* (EAMA) of Nagata [126] for the CVRP. MA is a population-based heuristic search approach that combines evolutionary algorithms for the global, more distant search (*exploration*), with local search algorithms to organize a more intensive local search (*exploitation*). For this reason, MAs are often referred to as hybrid GA or genetic local searches. The proposed EAMA is based on a two-stage approach. An initial population of solutions, each consisting of the same number of routes, is generated with the route-minimization procedure

developed by Nagata and Bräysy [127]. A subsequent procedure of the EAMA is then applied for minimizing total travel distance for the determined number of routes.

The initial population is typically created either randomly or using modifications of well-known construction heuristics. A random heuristic can be found in Blanton et al. [29] and Le Bouthillier and Crainic [31], though the last one applies also a set of construction heuristics combined with 2-opt, 3-opt, and Or-opt improvement heuristics.

Fitness values are usually based on routing costs, i.e. number of routes, total distance, and duration. The most typical selection scheme for selecting a pair of individuals for recombination is the *roulette-wheel* scheme. In this stochastic scheme, the probability of selecting an individual is proportional to its fitness value. Tan et al. [169] and Jung and Moon [95] use so-called *tournament selection*. The basic idea is to perform the roulette wheel scheme twice and to select the better out of the two individuals identified by the roulette wheel scheme. In Wee Kit et al. [100], Homberger and Gehring [85] [86], Gehring and Homberger [69] [70], and Mester [118], the parents are selected randomly. Finally, Potvin and Bengio [136] and Le Bouthillier and Crainic [31] use a ranking scheme, where the probability of selecting an individual is based on its rank.

The recombination is the most crucial part of a GA. The traditional two-point crossover, which exchanges a randomly selected portion of the bit string between the chromosomes, is used, for example, in Thangiah [172]. In the context of VRPTW, many authors have proposed specialized heuristic crossover procedures, instead of traditional operators. For instance, in Berger et al. [27][26], a removal procedure is first carried out to remove some key customer nodes in a similar fashion to Shaw's LNS [161]. Then, an insertion procedure inspired from Solomon [163] is locally applied to reconstruct the partial solution.

The mutation is often considered as a secondary strategy, and its purpose in traditional GA is mainly to help escape from local minima. However, in the evolution strategies of Homberger and Gehring [85] [86], Gehring and Homberger [69] [70], and Mester [118], the search is mainly driven by mutation, based on traditional local search operators (2^*-opt, Or-opt, and λ-interchanges). Berger et al. [26] present five mutation operators including Shaw's LNS [161], λ-exchanges, exchange of customers served too late in the current solution, elimination of the shortest route, and within-route reordering using Solomon's heuristic [163].

Different intensification techniques are often coupled to a GA. For instance, Tan et al. [169] introduce a special hill-climbing technique, where a randomly selected part of the population is improved by partial λ-exchanges. In Wee Kit et al. [100], a simple TS is applied to individual solutions in the later generations to intensify the search. Like the in many TS works, many GA allow infeasibilities during the search to escape from local minima. Examples of such strategies can be found in Blanton and Wainwright [29], Thangiah [172], Berger et al. [26], and Le Bouthillier and Crainic [31].

Another important approach to the CVRP is given by the *Greedy Randomized Adaptive Search Procedure* (GRASP) [61] [62] [147]. A GRASP algorithm is a multi-start or iterative process in which each GRASP iteration consists of two phases: a construction phase, in which a feasible solution is produced, and a local search phase, in which a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result. In the construction phase, a feasible solution is iteratively constructed, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements in a candidate list according to a greedy function. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by the random choice of one of the best candidates in the list, but not necessarily the top candidate. This choice technique allows for different solutions to be obtained at each GRASP iteration.

Kontoravdis and Bard [102] proposed a two-phase GRASP for the VRPTW. The construction procedure first initializes a number of routes by selecting seed customers that are either geographically most dispersed or the most time constrained. After initialization, the algorithm finds the best feasible insertion location in each route for every unrouted customer and calculates a specific penalty value using Solomon's cost function [163].

Among metaheuristics, *Variable Neighborhood Search* (VNS) (see section 3.3), introduced for the first time by Mladenovic and Hansen [121], is a quite recent method with far less application examples in VRP research. However, interesting results have been obtained even applying the simplest VNS algorithms. For example, Hasle and Kloster [81] apply a *Variable Neighborhood Descent* (VND) scheme (see section 3.3.1) to solve the CVRP, capable of improving some best-known solutions and providing the first known solution of

a benchmark instance.

Rousseau et al. [155] examine a VND scheme and new large neighborhood operators within a CP framework. The first operator introduced is inspired by ideas from Shaw's LNS [161]. The algorithm removes first, randomly, a subset of customers with a bias toward customers generating the longest detour. A CP version of the *GENI* algorithm for the TSP with time windows by Gendreau et al. [74] is used to reinsert removed customers. The second operator introduced is called *naive ejection chains* [76], and it is used to create the initial solution and diversify the search. To limit search space and prevent cycling, the first completed ejection chain is always accepted and each customer is allowed to be moved only once. The third proposed operator, *SMART*, removes a set of arcs instead of customers from the solution, creating an incomplete solution. The removed arcs can be either consecutive or randomly selected with a bias toward the longer arcs. This smaller routing problem is then solved either exactly by using the modified TSP with time windows model developed by Pesant et al. [132] or, in the case of a larger neighborhood size, by using *limited discrepancy search* with a bounded number of discrepancies. The search oscillates between the two suggested operators to escape local minima. In the end, routes are either exactly reordered using the algorithm of Pesant et al. [132] or, in case of longer routes, using the post-optimization part of the algorithm proposed by Pesant et al. [133].

Bräysy [33] presents a four-phase deterministic metaheuristic algorithm based on a modification of the VNS. In the first phase, an initial solution is created using a construction heuristic that borrows its basic ideas from the works of Solomon [163] and Russell [157]. Routes are built one at a time sequentially and after k customers have been inserted into the route, it is reordered using Or-opt exchanges. Then, a special route elimination operator based on ejection chains is used to minimize the number of routes. In the third phase, the created solutions are improved in terms of distance using VNS oscillating between four improvement procedures. These procedures are based on modifications to CROSS-exchange of Taillard et al. [166] and cheapest insertion heuristics. In the fourth phase, the objective function used by the local search operators is modified to also consider waiting time to escape from local minima.

Bräysy et al. [37] continue the study of Bräysy [33] by introducing modifications to the construction and improvement heuristics, and by applying a new post-optimization technique based on *threshold accepting*, that can be considered as a deterministic modification of the SA. The reordering proce-

dure is removed from the construction heuristic, and an extension of ejection chains that allows for infeasible solutions and removal of several customers from each route is used in the second phase. In the distance optimization phase, only modifications of CROSS-exchange are used. The modifications also include considering inverting the order of the customers in the selected segments, and more insertion positions for segments. The post-optimization is based on an interroute exchange heuristic that combines the ideas of CROSS- and GENIUS-exchanges.

In addition, a variety of other metaheuristics have been applied to the VRP. Chiang and Russell [43] developed a SA approach for the VRPTW. The authors combine the SA process with the parallel construction approach of Russell [157] that incorporates improvement procedures during the construction process. Tan et al. [170] developed a fast SA method based on two-interchanges with best-accept strategy and a monotonously decreasing cooling scheme. After the final temperature is reached, special temperature resets based on the initial temperature and the temperature that produced the current best solution are used to restart the procedure. Li et al. [110] propose a tabu-embedded SA restart metaheuristic. Initial solutions are created by the insertion and extended sweep heuristics of Solomon [163]. Three neighborhood operators based on shifting and exchanging customer segments between and within routes are combined with a SA procedure that is forced to restart from the current best solution several times. Other examples of hybrid SA approaches can be found in the works of Zhong and Pan [182] and Brandão de Oliveira and Vasconcelos [53]. In this last work, the authors combine SA with a *Hill Climbing* heuristic and a random restart policy that helps on diversifying the search.

Kilby et al. [98] introduced GLS for VRPTW. GLS is a memory-based technique which operates by augmenting the cost function with a penalty term based on how close the search moves to previously visited local minima, thus encouraging diversification. GLS moves out of local minima by penalizing particular solution features it considers should not occur in a near-optimal solution weighted by the number of times the feature has already been penalized. Kilby et al. [98] choose arcs as the feature to penalize. In the initial solution, no visits are allocated to any vehicle. A penalty is associated with not performing a visit, and so the search process constructs a solution in the process of minimizing cost using four different local searches.

De Backer et al. [11] test iterative improvement techniques within a CP framework. The improvement techniques are coupled to TS and GLS to avoid the search being trapped in local minima. The CP system is used only as

background operator to check the validity of solutions and to speed up legality checks of improvement procedures.

Gambardella et al. [67] use an *Ant Colony Optimization* (ACO) [56] approach with a hierarchy of two cooperative artificial ant colonies. The first colony is used to minimize the number of vehicles, while the second colony minimizes the total traveled distance. The two colonies cooperate through updating the best solution found, and in case the new best solution contains fewer vehicles, both colonies are reactivated with the reduced number of vehicles. Chen and Ting [41] propose an improved ACO with modified local and global pheromone updating rules. The authors add local search procedures to ant's behavior. Qi and Sun [142] propose an ACO hybridized with a randomized algorithm, which uses some randomly chosen customers in ants' probability evaluation function.

Finally, *Particle Swarm Optimization* (PSO) [97] algorihtms are a research area that have attracted some attention in the last years. For instance, Ai and Kachitvichyanukul proposed PSO approaches for both the CVRP [5] and the VRPTW [4]. Another example is the work by Masrom et al. [117], where the authors introduce a hybrid approach between PSO and GA to overcome the problem of premature convergence that often arises in standard PSO.

Chapter 3

Technologies

In this chapter, the technologies used throughout this work are reviewed. First, we provide a formal definition for *combinatorial optimization problems*, the concept of *neighborhood* and *local search*. Although we have commented the origins and some basic features in section 2.1, the central part of this chapter is devoted to describe some characteristics of the three main technologies applied in the proposed approach: *constraint programming*, *variable neighborhood search* and *large neighborhood search*. Other methods that we use for some specific operations are briefly introduced at the end of this chapter.

3.1 Local Search: an overview

We first formally introduce a *combinatorial optimization problem* and the concept of a *neighborhood*. There are alternative ways of representing combinatorial optimization problems, all relying on some method for representing the set of *feasible solutions*. Here, we will let the set of feasible solutions be represented as subsets of a finite set. This formulation is based on the one presented in [3].

Let $E = \{1, 2, ..., m\}$ be a finite set. In general, for a set S, we let $|S|$ denote its *cardinality*. Let $F \subseteq 2^E$, where 2^E denotes the set of all the subsets of E. The elements of F are called *feasible solutions*. Let $f : F \to \mathbb{R}$. The function f is called the *objective function* or *cost function*. Then an instance of a *combinatorial optimization problem* is represented as follows:

$$\min\{f(S) : S \in F\}$$

We assume that the family F is not given explicitly by listing all its elements; instead, it is represented in a compact form of size polynomial in m. An instance of a combinatorial optimization problem is denoted by the pair (F, f). For most of the problems, the cost function is linear, that is, there is a vector $f_1, f_2, ..., f_m$ such that for all feasible sets S, $f(S) = \sum_{i \in S} f_i$.

Suppose that (F, f) is an instance of a combinatorial optimization problem. A *neighborhood function* is a point to set map $N : F \rightarrow 2^E$. Under this function, each $S \in F$ has an associated subset $N(S)$ of E. The set $N(S)$ is called the *neighborhood* of the solution S, and we assume without loss of generality that $S \in N(S)$. A solution $S^* \in F$ is said to be *locally optimal* with respect to a neighborhood function N if $f(S^*) \leq f(S)$ for all $S \in N(S^*)$. The neighborhood $N(S)$ is said to be *exponential* if $|N(S)|$ grows exponentially in m as m increases.

With these definitions it is possible to define a *neighborhood search* algorithm. The algorithm takes an initial solution x as input. It computes $x' = argmin_{x'' \in N(x)}\{f(x'')\}$, that is, it finds the cheapest solution x' in the neighborhood of x. If $f(x') < f(x)$ then the algorithm performs the update $x \leftarrow x'$. The neighborhood of the new solution x is searched for an improving solution and this is repeated until a local optimum x is reached. When this happens the algorithm stops. The algorithm is denoted a *steepest descent* algorithm if it always chooses the best solution in the neighborhood.

3.2 Constraint Programming

Constraint Programming (CP) is powerful paradigm for representing and solving a wide range of combinatorial problems [154]. In the last few decades, it has attracted much attention among researchers due to its flexibility and its potential for solving hard combinatorial problems in areas such as scheduling, planning, timetabling and routing. CP combines strong theoretical foundations (e.g. techniques originated in different areas such as Mathematics, Artificial Intelligence, and Operations Research) with a wide range of application in the areas of modeling heterogeneous optimization and satisfaction problems. Moreover, CP nature provides other important advantages such as fast program development, economic program maintenance and efficient runtime performance.

Problems are expressed in terms of three entities: *variables*, their corresponding *domains*, and *constraints* relating them. Constraints can be consid-

ered as the heart of CP. They are treated as logical relations among several *unknowns* (or *variables*), each taking a value from a set of accepted values called *domain*, which can be a range with lower and upper bounds or a discrete list of numbers. The representation of the problem, in terms of constraints, results in short and simple programs easily adaptable to future changing requirements. Furthermore, quick developing and modification of programs makes it possible to experiment with different models until the best and fastest program has been found - without the programming task becoming unmanageable. This helps the programmer to concentrate only on finding the best model for the problem.

The practical benefits of CP really began to emerge when it was embedded in a programming language. Thus, CP is usually found embedded in a logic programming language, such as Prolog. In that case, it is called *Constraint Logic Programming* (CLP), but it does not necessarily mean that CP is restricted to CLP. Constraints can be integrated also to typical imperative languages like C/C++, e.g. COMET [176] or ILOG [2], and Java, e.g. Cream [168]. The programs implementing the methodologies presented in this thesis have been made using the CLP platform ECLiPSe [9].

A CLP language combines:

- *logic*, which is used to specify a set of possibilities to be explored by means of very simple search methods like generate-and-test, backtracking or backmarking,
- and *constraints*, which are used to minimize the search by eliminating impossible alternatives in advance by the use of consistency techniques like node-consistency, arc-consistency, path-consistency or directional arc-consistency.

Thus, the system combines reasoning and search. The constraints are used to restrict and guide the search. This combination is a common way of solving problems with a set of constraints to be satisfied. There are mainly two approaches, depending on the way reasoning and search complement each other:

- *look back* schemes, such as backtracking, backjumping, backchecking, and backmarking,
- and *look ahead* schemes, like forward checking and partial look ahead.

Since CP is the study of computational systems based on constraints, its idea is to solve problems by stating constraints (requirements) about the problem area and, consequently, finding a solution satisfying all the constraints. This class of problems is usually termed *Constraint Satisfaction Problems* (CSP) and the core mechanism used in solving them is *constraint propagation.*

Constraint propagation embeds any reasoning which consists in explicitly forbidding values or combinations of values for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise [28]. In other words, constraint propagation is a way to produce the consequences of a decision.

In order to formally describe how constraint propagation works, some definitions should be introduced first.

Definition A *label* is a variable-value pair that presents the assignment of the value to a variable. It is used $< x, v >$ to denote the label of assigning the value v to the variable x.

Definition A *compound label* is the simultaneous assignment of values to a (possibly empty) set of variables. It is used $(< x_1, v_1 >, < x_2, v_2 > , ..., < x_n, v_n >)$ to denote the compound label of assigning $v_1, v_2, ..., v_n$ to $x_1, x_2, ..., x_n$, respectively.

Definition A *Constraint Satisfaction Problem (CSP)* is a triple

$$(Z, D, C)$$

where

- Z is a finite set of *variables* $x_1, x_2, ..., x_n$;
- D is a function which maps every variable in Z to a set of objects of arbitrary type: $D : Z \rightarrow$ finite set of objects (of any type)

  We shall take D_{x_i} as the set of objects mapped from x_i by D. We call these objects possible values of x_i and the set D_{x_i} the *domain* of x_i;
- C is a finite (possibly empty) set of *constraints* on an arbitrary subset of variables in Z. In other words, C is a set of sets of compound labels.

A solution to a CSP is a full assignment to the variables of the problem, in such a way that all constraints are satisfied at once. We may want to find:

- just one solution, with no preference to which one,
- all solutions,
- an optimal, or at least a good solution, given some objective function defined in terms of some or all of the variables. In this case, the CSP becomes a *Constraint Optimization Problem* (COP).

Propagation is a generalization of data-driven computation. In general, when a variable belonging to a constraint is labeled, that value is propagated to the rest of variables involved in that constraint.

The next example intuitively shows how propagation works. Let us consider a problem with three variables, X, Y, and Z, whose domains are defined as follows:

$$[X, Y] :: [0..10] \rightarrow X\{0..10\}, Y\{0..10\}$$

$$Z :: [2..8] \rightarrow Z\{2..8\}$$

where the symbol '::' is used to define the domain of a variable or a set of variables. Thus, $X :: [D_{X_{min}}..D_{X_{max}}]$ declares the possible values for the variable X to be between the bounds $D_{X_{min}}$ and $D_{X_{max}}$. Once the domain of a variable has been defined, that variable is represented as $X\{D_{X_{min}}..D_{X_{max}}\}$.

We consider these variables subject to the following set of constraints:

$$C1 : X\{0..10\} = Y\{0..10\} + 1$$

$$C2 : Y\{0..10\}^2 > Z\{2..8\}$$

$$C3 : Z\{2..8\} = X\{0..10\} + 1$$

$$C4 : Z\{2..8\} > 4$$

Before binding a variable, due to the unary constraint $C4$ ($Z > 4$), the values 2 to 4 are removed from the domain of Z. Its new domain is [5..8]. All the unary constraints, that is, constraints involving only one variable, are evaluated before any variable assignation.

$$X\{0..10\}, Y\{0..10\}, Z\{2..8\} \xrightarrow{C4} X\{0..10\}, Y\{0..10\}, Z\{5..8\}$$

Binary and more complex constraints are used to propagate the changes made to one of their variables to the other variables involved in that constraint.

During the search of a solution, an assignation $Y = 0$ makes the constraint $C1$ and $C2$ propagate and find an inconsistency. Hence, the value 0 is rejected from the domain of Y and the search continues:

- Propagation over $C1$:

$$X\{0..10\} = Y\{0\} + 1 \rightarrow X\{0..10\} = 0 + 1 = 1 \rightarrow X\{1\}$$

- Propagation over $C2$:

$$Y\{0\}^2 > Z\{5..8\} \rightarrow 0^2 = 0 > Z\{5..8\}$$

  An inconsistence is found, since Z has no value lower than 0 in its domain.

This is a first assignation to a variable in the search of a solution satisfying the constraints, i.e. a *feasible solution*. From this point, a new value from the domain of Y would be tried and so on.

An important contribution of CP is to allow the end user to control the search. The topic of search comes from the heart of AI, which has developed several algorithms to perform the search in a solution space. End user's search control is achieved by combining generic techniques, when the generation of the whole search tree is unfeasible, and problem-specific techniques, when there is an extra knowledge about special features of the problem. Thus, while mathematical programming is mainly based in the application of certain algorithms to a model, CP allows the user to take some decisions on the search stage like the order of instantiation of the variables and the order of selection of values from domains. Depending on those decisions the way decisions are made is totally different and the performance of the search algorithm can be highly affected.

Given the benefits carried by the combination of these features, several research groups are studying different fields. One line of research has focused on propagation, showing how to improve the way information propagates. Another line of research is the global shape of the problem. It considers the

problem like a graph, where each variable is a node and each constraint an edge (or an hyper-edge) in the graph. This is a technique mainly used with binary CSPs, i.e. CSPs where the primitive constraints have at most two variables. Tree-structured problems are relatively easy to solve, but research has also revealed a variety of ways of dealing with more awkward structures, by breaking down a problem into easier subproblems, whose results can be combined into a solution of the original problem.

3.2.1 Constraint Satisfaction

As it has been said above, constraint satisfaction is related to problems defined over finite domains. Solutions to a CSP can be found by searching (systematically) through the possible assignments of values to variables, that is generating the whole search tree. Search methods can be divided into two broad classes: those that traverse the space of partial solutions (or partial value assignments), and those which explore the space of complete value assignments (to all variables) stochastically.

From the theoretical point of view, solving a CSP is trivial using systematic exploration of the solution space. But that is not true from the practical point of view, where the efficiency takes place. Even if systematic search methods (without additional improvements) look very simple and non-efficient, they are important because they make the foundations of more advanced and efficient algorithms.

The simplest algorithm that searches the space of complete labelings, is called *Generate-and-Test* (GT). The idea of GT is very simple: firstly, a complete labeling of variables is randomly generated and, consequently, if this labeling satisfies all the constraints then the solution is already found; otherwise, another labeling is tried.

The GT algorithm is clearly a weak generic algorithm used only if everything else failed. Its efficiency is very poor for two reasons: it has a non-informed generator and there is a late discovery of inconsistence. There are two ways to improve efficiency in GT:

- To program a smart (informed) generator of valuations, i.e. able to generate the complete valuation in such a way that the conflict found by the test phase is minimized.

- To merge the generator and the tester, i.e. the validity of the constraint is tested as soon as its respective variables are instantiated. This method is used by the *backtracking* approach.

Backtracking (BT) [8] is a method used for solving CSPs by incrementally extending a partial solution that specifies consistent values for some of the variables, towards a complete solution, by repeatedly choosing a value for another variable consistent with the values in the current partial solution.

As said above, BT is a merge of the generating and testing phases of GT. The variables are labeled sequentially and as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial solution violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available. Clearly, whenever a partial instantiation violates a constraint, backtracking is able to eliminate a subspace from the Cartesian product of all variables' domains. Hence, backtracking is strictly better than GT. However, its running complexity for most non-trivial problems is still exponential.

There are three major drawbacks of the standard BT:

- *Thrashing*: it is a repeated failure (and consequent backtrack) due to the same reason. This happens because there is no information stored when a failure occurs. Thus, if there is a similar situation in the future the search will also fail and backtrack.

- Redundant work: conflicting values of variables are not remembered. This makes the search fail the same way in different branches of the tree.

- Late detection of conflicts: conflict is not detected before it really occurs.

Improved methods for solving the first two drawbacks were proposed, namely *backjumping* and *backmarking* (introduced in section 3.2.2), but more attention was paid to detect the inconsistency of partial solutions sooner using *consistency techniques* [28]. The latter are based on the idea of removing inconsistent values from variables' domains until a solution is found. It is very important to note that consistency techniques are deterministic.

There exist several consistency techniques, but most of them are not complete [8]. For this reason, these techniques are rarely used alone to solve a CSP completely. The names of basic consistency techniques are derived from

the graph notions. As said above, a binary CSP can be represented as a constraint graph where nodes correspond to variables and edges are labeled by constraints [123]. Although this representation can be applied only to binary CSPs, it is easy to show that every CSP can be transformed to an equivalent binary CSP [153]. However, in practice this operation is not likely to be worth doing and it is easier to extend the algorithms so they can tackle non binary CSPs as well.

Among consistency techniques, some of the most common are:

- *Node-Consistency*: it removes values from variables' domains that are inconsistent with constraints involving one variable, i.e. unary constraints. It is the simplest consistency technique.
- *Arc-Consistency*: it removes values from variables' domains which are inconsistent with constraints involving two variables, i.e. binary constraints.
- *Path-Consistency*: it requires for every pair of values of two variables x and y satisfying the respective binary constraint that there exists a value for each variable along some path between x and y such that all binary constraints in the path are satisfied.
- *K-Consistency* and *Strong K-Consistency*: a constraint graph is k-consistent if for every system of values for $k-1$ variables satisfying all the constraints among these variables, there is a value for an arbitrary k^{th} variable such that the constraints among these variables are satisfied. A constraint graph is strongly k-consistent if it is j-consistent for all $j \leq k$. All previously mentioned techniques can be generated by k-consistency and strong k-consistency.

Attention should be paid to the use of these consistency techniques. They provide a good mechanism to remove inconsistent values from variables' domains during search, but they often penalize with respect to efficiency terms. For this reason, they are often neglected on designing efficient search algorithms and substituted by heuristic approaches.

3.2.2 Search schemes

Although both systematic search and some consistency techniques can be used alone to solve a CSP completely, this is rarely done. A combination of both

approaches is a more common way of solving a CSP. There are mainly two schemes to solve a CSP using constraint propagation: the *look back* and the *look ahead* schemes.

A *look back* schema uses consistency checks among already instantiated variables. BT is a simple example of this schema. To avoid some problems of BT, like thrashing and redundant work, other look back schemes were proposed.

Backjumping (BJ) is a method to avoid thrashing in BT. The control of BJ is exactly the same as BT, except when backtracking takes place. Both algorithms pick one variable at a time and look for a value for this variable making sure that this new assignment is compatible with values committed to so far. However, if BJ finds an inconsistency, it analyses the situation in order to identify the source of inconsistency. It uses the violated constraints as a guidance to find out the conflicting variable. If all the values in the domain are explored, then the BJ algorithm backtracks to the most recent conflicting variable. This is the main difference from the BT algorithm, which backtracks to the immediate past variables.

Another look back schemes, called *Backchecking* (BC) and *Backmarking* (BM), avoid redundant work of BT. Both BC and its descendent BM are useful algorithms for reducing the number of compatibility checks. If the algorithm finds that some label $< Y, b >$ is incompatible with any recent label $< X, a >$ then it remembers this incompatibility. As long as $< X, a >$ is still committed to, the $< Y, b >$ will not be considered again.

BM is an improvement over BC that avoids some redundant constraint checking as well as some redundant discoveries of inconsistencies. It reduces the number of compatibility checks by remembering for every label the incompatible recent labels. Furthermore, it avoids repeating compatibility checks which have already been performed and which have succeeded.

To cope with COPs, one should take into account the cost function. The appropriate modification of the BT search schema is called *Branch-and-Bound* (BB). During the search, BB maintains the current best value of the cost function (*bound*) and, each time a solution with a smaller cost is found, its value is updated. There are many variations on the BB algorithm. One consideration is what to do after a solution with a new best cost is found. The simplest approach is to restart the computation with the *bound* variable initialized to this new best cost. A less naive approach is to continue the search for better solutions without restarting. In this case, the cost function upper bound is

constrained to the *bound* variable value. Each time a solution with a new best cost is found, this cost is dynamically imposed through this constraint. The constraint propagation triggered by this constraint leads to a pruning of the search tree by identifying the nodes under which no solution with a smaller cost can be present.

All look back schemes share the disadvantage of late detection of the conflict. As a matter of fact, they solve the inconsistency when it occurs but do not prevent the inconsistency to occur. Therefore, *look ahead* schemes were proposed to prevent future conflicts.

Forward Checking (FC) is the easiest example of *look ahead* strategy. It performs arc-consistency between pairs of not yet instantiated variable and instantiated variable, i.e. when a value is assigned to the current variable, any value in the domain of the "future" variable which conflicts with this assignment is (temporally) removed from the domain. Therefore, FC maintains the invariance that for every unlabeled variable there exists at least one value in its domain that is compatible with the values of instantiated/labeled variables. FC does more work than BT when each assignment is added to the current partial solution. Nevertheless, it is almost always a better choice than chronological backtracking.

Even more future inconsistencies are removed by the *Partial Look Ahead* (PLA) method. While FC performs only the checks of constraints between the current variable and the future variables, the PLA extends this consistency checking even to variables which does not have direct connection with labeled variables.

The approach that uses full arc-consistency after each labeling step is called *(Full) Look Ahead* (LA) or *Maintaining Arc Consistency* (MAC). It can use arbitrary arc-consistency algorithms to achieve arc-consistency. However, it should be noted that LA does even more work than FC and PLA when each assignment is added to the current partial solution. Actually, in some cases LA may be more expensive than BT and, therefore, FC and BT are still used in applications.

Stochastic and Heuristic Algorithms

As mentioned, one of the main contributions of CP is to allow the end user to control the search, so problem-specific and generic techniques can be combined to obtained more efficient algorithms. Thus, depending on the problem to be

solved, it is possible to apply specific CP techniques. However, it is important to notice that, although a search improved by these techniques can be useful to find a faster solution for a problem, it can significantly slow the solution of a different problem.

An important technique is to select the order of labeling for the variables and the order in which the values in the domains of these variables are selected. This point represents one of the most important differences with Linear Programming (LP): when using LP, once the problem is modeled, the rest of the work is done by the solver. In the CP methodology, the order of variable labeling and value selection is essential to drive the search. It could be unnecessary to find the optimal values for a set of variables if they do not affect the cost function variables. In this case, they can be labeled only once or leave them represented by their bounded feasible domains. Some of these techniques come usually implemented in CP platforms such as ECLiPSe [9].

Generic techniques for local search such as Genetic Algorithms, Simulated Annealing (SA), or Tabu Search (TS) can also be used to aid CP to find quasi-optimal solutions when it is not feasible to generate the whole search tree (due to memory or CPU time problems). Some examples have been introduced in section 2.1.2. These methods are used when the size of the problem is huge and it is not possible to find the optimal solution. Then, CP is used to find fast poor solutions which will be used as initial values for these techniques. A good solution is sought from these input values. If the best solution found by these techniques is not good enough then new initial values are generated by CP. To avoid the same values than in previous searches, either new constraints are added or some of the existing constraints are removed.

For further descriptions on CP modeling techniques, algorithms, tools, and applications, the reader is referred to [154].

3.3 Variable Neighborhood Search

Variable Neighborhood Search (VNS), as defined in [80], is a metaheuristic, or a framework for building heuristics, aimed at solving combinatorial and global optimization problems. It exploits systematically the idea of neighborhood changes both in descent phase to find a local minimum, and in perturbation phase, to emerge from the corresponding valley. Since it was first proposed in 1997 [121], VNS has rapidly grown into many developments and has been applied in different fields. Its application to VRP problems has been depicted

in section 2.1.2. A complete and revised description of different VNS methods can be found in [79].

VNS concept strongly depends on the following observations:

1. A local minimum with respect to one neighborhood structure is not necessarily a local minimum for another neighborhood structure.

2. A global minimum is a local minimum with respect to all possible neighborhood structures.

However, experiments show that, for many problems, local minima with respect to one or several neighborhoods are relatively close to each other. It implies that a local optimum often provides some information about the global optimum. This fact is used in some VNS schemes to guide the search.

One of the main advantages of VNS with respect to many other metaheuristics relies upon the fact that the basic schemes of VNS and their extensions are simple and require few, and sometimes no parameters.

In order to formally describe the different VNS algorithms, some notation should be introduced first. Let us denote with N_k, $(k = 1, ..., k_{max})$, a finite set of pre-selected neighborhood structures, and with $N_k(x)$ the set of feasible solutions in the k^{th} neighborhood of x. An optimal solution x_{opt} (or global minimum) is a feasible solution where a minimum of the problem is reached. We call $x' \in X$ a local minimum of the problem with respect to N_k, if there is no solution $x \in N_k(x') \subseteq X$ such that $f(x) < f(x')$. In the description of all algorithms that follow, we assume that an initial solution x is given.

3.3.1 Variable Neighborhood Descent (VND)

The *Variable Neighborhood Descent* (VND) method is obtained if the change of neighborhoods is performed in a deterministic way. It is the simplest VNS schema and so is often used as a local search method for more complex frameworks, such as VNS itself. The VND algorithm is presented in Algorithm 3.1.

As it can be inferred from Algorithm 3.1, the final solution should be a local minimum with respect to all neighborhoods; hence the chances to reach a global one are larger when using VND than with a single neighborhood structure.

Algorithm 3.1 Variable Neighborhood Descent (VND)

Initialization: Find an initial solution x.

Repeat the following sequence until no improvement is obtained:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Exploration of neighborhood: find the best neighbor x' of x ($x' \in N_k(x)$).
   (b) Move or not: if the solution x' thus obtained is better than x, set $x \leftarrow x'$ and $k \leftarrow 1$; otherwise, set $k \leftarrow k + 1$.

Alternatively to the *best accept* strategy used in steps *(a)* and *(b)*, where the neighbor $x' \in N_k(x)$ with the lowest cost value is selected, a *first accept* strategy can be used. The latter consists on accepting the first neighbor $x' \in N_k(x)$ found during the search that is able to improve the current solution x cost value. Thus, the first accept strategy draws an incomplete search able to speed up neighborhoods exploration when they are huge. In this case, performing a complete search using the best accept strategy could lead to unmanageable computational times.

Although VND is a very simple algorithm, some drawbacks may arise when applying this method to some problems. In particular, the complexity of the different moves used to define the neighborhoods has a large influence on algorithm's performance. Often, a move defined over a simple elementary change may lead to a very large neighborhood. If exploring such a neighborhood involves checking too many elementary changes, the resulting heuristic may be very slow and often take more time than an exact algorithm on small or medium size examples.

Another important question when implementing a VND algorithm is which order is the best in applying the different moves. It bears upon computing times in relation to the quality of solutions obtained. A frequent implementation consists of ranking moves by order of complexity of their application and returning to the first one each time a direction of descent is found and a step made in that direction.

Algorithm 3.2 Reduced VNS

Initialization: Find an initial solution x; choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Shaking: generate a point x' at random from the k^{th} neighbourhood of x ($x' \in N_k(x)$).
   (b) Move or not: if this point is better than the incumbent, move there ($x \leftarrow x'$) and set $k \leftarrow 1$; otherwise, set $k \leftarrow k + 1$.

Finally, it is important to ensure that the moves considered guarantee a thorough exploration of the region containing x. For some problems, elementary moves are not sufficient to leave a narrow valley, and heuristics using them only can give very poor results. Thus, this question is crucial. It is also related to the desired precision of the VND final solution. In most situations, a better solution will be pursued when VND is used alone, while poorer solutions may be accepted when it is embedded in a larger framework, such as VNS itself. In this case, one may prefer to get a good-enough solution fairly quickly by the deterministic VND, using simpler moves or incomplete search, and to improve it later by faster stochastic search in VNS.

3.3.2 Reduced VNS

The Reduced VNS (RVNS) method is obtained if random points are selected from $N_k(x)$ and no local search is made. Rather, the values of these new points are compared with that of the incumbent and updating takes place in case of improvement. We assume that a stopping condition has been chosen, among various possibilities, e.g., the maximum CPU time allowed t_{max}, or the maximum number of iterations between two improvements. Therefore, RVNS uses two parameters: t_{max} (or, alternatively, the maximum number of iterations) and k_{max}. Its steps are presented in Algorithm 3.2.

A set of neighborhoods $N_1(x), N_2(x), ..., N_{k_{max}}(x)$ is considered around the current point x. Since any generated solution $x' \in N_k(x)$ may be accepted

during the search, the current point x may be or not a local optimum. Usually, the neighborhoods $N_1(x), N_2(x), ..., N_{k_{max}}(x)$ are nested, i.e., each one contains the previous. Then a point is chosen at random in the first neighborhood. If its value is better than that of the incumbent ($f(x') < f(x)$), the search is recentered there ($x \leftarrow x'$). Otherwise, one proceeds to the next neighborhood. After all neighborhoods have been considered, one begins again with the first ($k \leftarrow 1$), until the stopping condition is satisfied.

Due to the nestedness property, the size of successive neighborhoods will be increasing. Therefore, one will explore more thoroughly close neighborhoods of x than farther ones. However, the search will reach these larger neighborhoods whenever it is not able to find further improvements within the first, smaller ones.

RVNS is useful in very large instances, for which local search is costly. RVNS is akin to a Monte Carlo method, but is more systematic.

3.3.3 Basic VNS and General VNS

The Basic VNS (BVNS) method combines deterministic and stochastic changes of neighborhood. It combines a local search with systematic changes of neighborhoods around the local optimum found. The BVNS can be seen as an extended RVNS where a local search is performed any time a random point is generated. The BVNS is presented in Algorithm 3.3.

Often successive neighborhoods N_k are nested as in the RVNS. The point x' is generated at random in order to avoid cycling, which might occur if a deterministic rule is applied. Then, a descent from x' is done with the local search routine. This leads to a new local minimum x''. At this point, three outcomes are possible: (i) $x'' = x$, i.e., the search is again at the bottom of the same valley; in this case the procedure is iterated using the next neighborhood $N_k(x)$, $k \geq 2$; (ii) $x'' \neq x$ but $f(x'') \geq f(x)$, i.e., another local optimum has been found, which is not better than the previous best solution; in this case the procedure is iterated using the next neighborhood too; (iii) $x'' \neq x$ and $f(x'') < f(x)$, i.e., another local optimum, better than the incumbent has been found; in this case the search is recentered around x'' and begins again with the first neighborhood. Should the last neighborhood be reached without a solution better than the incumbent being found, the search begins again at the first neighborhood $N_1(x)$ until the stopping criterion is met.

Algorithm 3.3 Basic VNS

Initialization: Find an initial solution x; choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Shaking: generate a point x' at random from the k^{th} neighborhood of x ($x' \in N_k(x)$).
   (b) Local Search: apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum.
   (c) Move or not: if the local optimum x'' is better than the incumbent x, move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.

If instead of simple local search, one uses VND and if one improves the initial solution (generally, by means of VND or RVNS), the algorithm obtained is known as the General VNS or, simply VNS. Its steps are presented in Algorithm 3.4. This algorithm has led to the most successful applications reported so far (see section 2.1.2).

Note that in the scheme presented in Algorithm 3.4, neighborhood structures have been denoted as N_k for the shaking and N_l for the local search. However, they are usually chosen to be the same structures for both processes. The main question arising from the selection of these neighborhood structures is what properties should hold in order to be able to find a globally optimal or near-optimal solution. To avoid being blocked in a valley, while there may be deeper ones, the union of the neighborhoods around any feasible solution x should contain the whole feasible set:

$$X \subseteq N_1(x) \cup N_2(x) \cup ... \cup N_{k_{max}}(x), \forall x \in X$$

These sets may cover X without necessarily partitioning it, which is easier to implement, e.g. when using nested neighborhoods, i.e.,

$$N_1(x) \subset N_2(x) \subset ... \subset N_{k_{max}}(x), X \subset N_{k_{max}}(x), \forall x \in X$$

Algorithm 3.4 General VNS

Initialization: Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$, that will be used in the shaking phase, and the set of neighborhood structures N_l, for $l = 1, ..., l_{max}$, that will be used in the local search; find an initial solution x and improve it by some local search process; choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Shaking: generate a point x' at random from the k^{th} neighborhood of x ($x' \in N_k(x)$).
   (b) Local Search by VND:
      i. Set $l \leftarrow 1$.
      ii. Repeat the following steps until $l = l_{max}$:
         - Exploration of neighborhood: find the best neighbor x'' of x' ($x'' \in N_l(x')$).
         - Move or not: if the solution x'' thus obtained is better than x', set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise, set $l \leftarrow l + 1$.
   (c) Move or not: if the local optimum x'' is better than the incumbent x, move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.

If these properties do not hold, the search might still be able to explore X completely, by traversing small neighborhoods around points on some trajectory, but it is no more guaranteed. However, often the size of the feasible set X is too large to be explored completely in an efficient manner, so a trade-off between thoroughness and efficiency should be adopted on defining the neighborhood structures.

Nested neighborhoods are easily obtained for many combinatorial problems by defining a first neighborhood $N_1(x)$ by a type of move (e.g. 2-opt in the TSP or the VRP) and then iterating it k times to obtain neighborhoods $N_k(x)$ for $k = 2, ..., k_{max}$. They have the property that their sizes are increasing. Therefore if, as is often the case, the search goes many times through the whole

Algorithm 3.5 Variable Neighborhood Decomposition Search
Initialization: Find an initial solution x; choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Shaking: generate a point x' at random from the k^{th} neighborhood of x ($x' \in N_k(x)$); in other words, let y be a set of p solution attributes present in x' but not in x ($y = x' \setminus x$).
   (b) Local Search: find a local optimum in the space of y either by inspection or by some heuristic; denote the best solution found with y' and with x'' the corresponding solution in the whole space S ($x'' = (x' \setminus y) \cup y'$).
   (c) Move or not: if the solution thus obtained is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.

sequence of neighborhoods, the first one will be explored more thoroughly than the last ones. This is a desirable property since, as mentioned, experiments show that for many problems local minima tend to be close one from another.

3.3.4 Other VNS approaches

While the BVNS is clearly useful for obtaining an approximate solution to many combinatorial and global optimization problems, it remains difficult or lengthy to solve very large instances. The Variable Neighborhood Decomposition Search (VNDS) method extends the BVNS into a two-level VNS scheme based on decomposition of the problem. Its steps are presented in Algorithm 3.5.

Note that the only difference between the BVNS and VNDS is in step 2b: instead of applying some local search method in the whole solution space S (starting from $x' \in N_k(x)$), in VNDS one solves at each iteration a subproblem in some subspace $V_k \subseteq N_k(x)$ with $x' \in V_k$. When the local search used in

this step is also VNS, the two-level VNS-schema arises. Thus, VNDS can be viewed as embedding the classical successive approximation schema in the VNS framework.

Other approaches based on VNS include the idea of parallelization, such as the Parallel VNS (PVNS). For this extension, several ways of parallelizing VNS have been outlined: (i) parallelize local search; (ii) augment the number of solutions drawn from the current neighborhood and make a local search in parallel from each of them; and (iii), do the same as (ii) but updating the information about the best solution found.

3.4 Large Neighborhood Search

In *Large Neighbourhood Search* (LNS), proposed by Shaw [161], an initial solution is gradually improved by alternately destroying and repairing the solution. Over the years, LNS has proved to be competitive with other local search techniques, especially when combined with CP. It complements the CP framework as LNS benefits from improved propagation while CP benefits from this efficient, while simple, search framework [131]. Some examples of its application to the VRP have been outlined in section 2.1.2. A complete introduction to the subject can be found in [134].

The LNS metaheuristic belongs to the class of heuristics known as *Very Large Scale Neighborhood search* (VLSN) algorithms. A neighborhood search algorithm is considered as belonging to the class of VLSN algorithms if the neighborhood it searches grows exponentially with the instance size or if the neighborhood is simply too large to be searched explicitly in practice [3]. Although the concept of VLSN was not formalized until recently, algorithms based on similar principles have been used for decades.

All VLSN algorithms are based on the observation that searching a large neighborhood results in finding local optima of high quality, and hence overall a VLSN algorithm may return better solutions. However, searching a large neighborhood is time consuming, hence various filtering techniques are used to limit the search. In VLSN algorithms, the neighborhood is typically restricted to a subset of the solutions which can be searched efficiently.

Intuitively, searching a very large neighborhood should lead to higher quality solutions than searching a small neighborhood. Nevertheless, in practice, small neighborhoods can provide similar or superior solution quality if embedded in a metaheuristic framework because they typically can be searched more

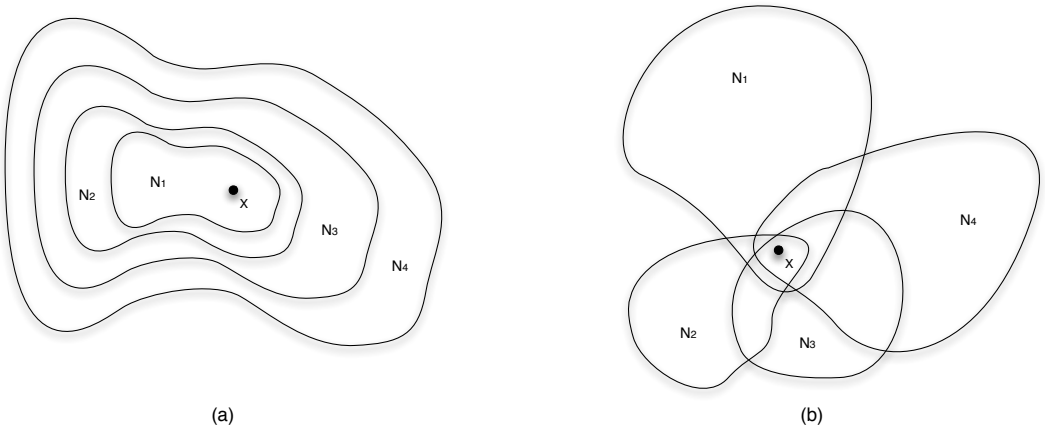

Figure 3.1: Illustration of the neighborhoods used by (a) VDNS and (b) VNS. The current solution is marked x. VDNS typically operated on one type of neighborhood with variable depth, while VNS operates on structurally different neighborhoods.

quickly. Large neighborhoods generally lead to local solutions of better quality, but the search is more time-consuming. Hence, a natural idea is to gradually extend the size of the neighborhood, each time the search gets trapped in a local minimum, as in the nested neighborhoods referred to in the previous section.

Variable-Depth Neighborhood Search (VDNS) methods search a parameterized family of still deeper neighborhoods $N_1, N_2, ..., N_k$ in a heuristic way (see Figure 3.1). A typical example is the *1-exchange* neighborhood N_1 where one variable/position is changed. Similarly, the *2-exchange* neighborhood N_2 swaps the value of two variables/positions. In general the *k-exchange* neighborhood N_k changes k variables. Variable-depth search methods are techniques that search the *k-exchange* neighborhood partially, i.e. up to a certain k not covering the whole search space, hence reducing the time used to search the neighborhood.

In the LNS metaheuristic, the neighborhoods are implicitly defined by methods (often heuristics) which are used to destroy and repair an incumbent solution. A *destroy* method destructs part of the current solution while a *repair* method rebuilds the destroyed solution. The destroy method typically contains an element of stochasticity such that different parts of the solution

Algorithm 3.6 Large Neighborhood Search

Initialization: Find an initial solution x and set $x^b \leftarrow x$ as the best solution found so far; choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. $x' = r(d(x))$
2. If $accept(x', x)$ then $x \leftarrow x'$
3. If $c(x') \leq c(x^b)$ then $x^b \leftarrow x'$

are destroyed in every invocation of the method. The neighborhood $N(x)$ of a solution x is then defined as the set of solutions that can be reached by first applying the destroy method and then the repair method. Since the destroy method can destruct a large part of the solution, the neighborhood contains a large amount of solutions which explains the name of the heuristic.

The steps of the LNS method are shown in Algorithm 3.6. Three variables are maintained by the algorithm: the variable x^b is the best solution observed so far during the search, x is the current solution, and x' is a temporary solution that can be discarded or promoted to the status of current solution. The function $d(\cdot)$ is the destroy method while $r(\cdot)$ is the repair method. More specifically, $d(x)$ returns a copy of x that is partially destroyed. Applying $r(\cdot)$ to a partly destroyed solution repairs it, i.e. it returns a feasible solution built from the destroyed one. Both destroy and repair methods can be implemented in different ways obeying different criteria. In step 2 the new solution is evaluated, and then the heuristic determines whether this solution should become the new current solution or whether it should be rejected. The *accept* function can be implemented in different ways. The simplest choice is to only accept improving solutions, but some works propose an acceptance criteria borrowed from SA [151], that is, accepting solutions that may be worse than the incumbent aiming to diversify the search.

From the pseudo-code presented in Algorithm 3.6, it can be noticed that the LNS metaheuristic does not search the entire neighborhood of a solution, but merely samples this neighborhood.

The destroy method is an important part of the LNS heuristic. The most important choice when implementing the destroy method is the degree of de-

struction: if only a small part of the solution is destroyed then the heuristic may have troubles exploring the search space as the effect of a large neighborhood is lost. If a very large part of the solution is destroyed, then the LNS heuristic almost degrades into repeated re-optimization or a multi-start process. This can be time consuming or yield poor quality solutions dependent on how the partial solution is repaired. Shaw [161] proposed to gradually increase the degree of destruction, while Ropke and Pisinger [151] choose the degree of destruction randomly at each iteration from a specific range dependent on the instance size. The destroy method must also be chosen such that the entire search space can be reached, or at least the interesting part of the search space where the global optimum is expected to be found. Therefore, it cannot focus on always destroying a particular component of the solution but must make it possible to destroy every part of the solution.

Choosing the repair method permits much more freedom when implementing an LNS heuristic. A first decision is whether the repair method should be optimal in the sense that the best possible full solution is constructed from the partial solution, or whether it should be a heuristic assuming that one is satisfied with a good solution constructed from the partial solution. An optimal repair operation will be slower than a heuristic one, but may potentially lead to high quality solutions in a few iterations. However, from a diversification point of view, an optimal repair operation may not be attractive: only improving or identical-cost solutions will be produced and it can be difficult to leave valleys in the search space unless a large part of the solution is destroyed at each iteration.

The *Adaptive Large Neighborhood Search* (ALNS) heuristic was proposed in [151] and extends the LNS heuristic by allowing multiple destroy and repair methods to be used within the same search. Using neighborhood search terminology, one can say that the ALNS extends the LNS by allowing multiple neighborhoods within the same search. Each destroy/repair method is assigned a weight that controls how often the particular method is attempted during the search. The weights are adjusted dynamically as the search progresses depending on the performance of each neighborhood, so that the heuristic adapts to the instance at hand and to the state of the search.

The considerations for selecting destroy and repair methods in the LNS heuristic also holds for an ALNS heuristic. However, the ALNS framework gives some extra freedom because multiple destroy/repair methods are allowed. In the pure LNS heuristic we have to select a destroy and repair method that is expected to work well for a wide range of instances. In an ALNS heuristic one

can afford to include destroy/repair methods that only are suitable in some cases, while the adaptive weight adjustment will ensure that these heuristics seldom are used on instances where they are ineffective. Therefore, the selection of destroy and repair methods can be turned into a search for methods that are good at either diversification or intensification.

Diversification and intensification for the destroy methods can be accomplished as follows: to diversify the search, one may randomly select the parts of the solution that should be destroyed (*random destroy* method). To intensify the search, one may try to remove a number of "critical" variables, i.e. variables having a large cost or variables spoiling the current structure of the solution (*worst destroy* or *critical destroy*, respectively). One may also choose a number of related variables that are easy to interchange while maintaining feasibility of the solution (*related destroy* method). Finally, one may use *history based destroy* where a number of variables are chosen according to some historical information.

The repair methods are often based on concrete well-performing heuristics for the given problem. They can be also based on approximation algorithms or exact algorithms. Exact algorithms can be relaxed to obtain faster solution times at the cost of solution quality. Time consuming and fast repair methods can be mixed by penalizing the time consuming methods. Furthermore, it can often be advantageous to use noising or randomization in the repair methods to help on obtaining a proper diversification.

The ALNS is related to the VNS metaheuristic in the sense that both heuristics search multiple neighborhoods. VNS makes use of a parameterized family of neighborhoods. When the algorithm reaches a local minimum using one of the neighborhoods, it proceeds with an often larger neighborhood from the parameterized family. When the VNS gets out of the local minimum, it proceeds with the smaller neighborhood. Thus, VNS performs the search in a systematic fashion. On the contrary, ALNS operates on a predefined set of large neighborhoods corresponding to the destroy and repair heuristics, and chooses which one to explore at each iteration according to its performance in previous stages of the search. Therefore, ALNS may be able to adapt the search as it progresses.

3.5 Other techniques

In this thesis, two other methods are used for some specific operations: *Lagrangian Relaxation* and a modified *Clarke and Wright Savings* heuristic. In this section, their main characteristics are outlined, although the concrete procedures used in this work are further explained in sections 4.2.1 and 4.3.1, respectively.

3.5.1 Lagrangian Relaxation

Lagrangian Relaxation (LR) is a general mathematical programming method applied for decomposing or relaxing problems to exploit their special structures. It has long been used for discovering theoretical insights and developing solution algorithms for various difficult mathematical programming problems. For discrete and combinatorial optimization problems, LR is typically used to relax a set of complicating side constraints and accordingly compensate a penalty term in the objective function [64]. By adjusting the values of *Lagrangian multipliers* with the penalty term to an appropriate level, the optimal solution may be found by solving the relaxed Lagrangian problem that can often take advantage of various previously developed algorithms. For a complete review on the subject, the reader is referred to [77].

Consider the following general optimization problem P, which we call the *primal problem* (P):

$$\min f(\mathbf{x}) \tag{3.1}$$

subject to:

$$g_i(\mathbf{x}) \geq b_i,\ i = 1, 2, ..., m \tag{3.2}$$

$$\mathbf{x} \in \mathbf{X} \tag{3.3}$$

where the functions $f(\mathbf{x})$, and $g_i(\mathbf{x})$, $i = 1, 2, ..., m$ can be arbitrary nonlinear or nonconvex functions. The feasible region of the problem consists of explicit constraints (3.2) and other constraints (including nonnegativity and integrality restrictions) which are represented by set (3.3). We assume that the problem would be easy to solve in the absence of constraints (3.2).

LR relaxes constraints (3.2) by moving them to the objective function with associated multipliers $u_i \geq 0$, $i = 1, 2, ..., m$, which results in the following *Lagrangian function*:

$$L(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}) + \sum_{i=1}^{m} u_i(b_i - g_i(\mathbf{x})) \tag{3.4}$$

where $u_i \geq 0$, $i = 1, 2, ..., m$ are called *Lagrangian multipliers* or *dual variables*. If a relaxed constraint i is a "$\leq$" constraint, then the associated Lagrangian multiplier is $u_i \leq 0$. Thus, equality constraints have unrestricted multipliers.

The *Lagrangian dual function* is defined as

$$h(\mathbf{u}) = \min_{\mathbf{x} \in \mathbf{X}} L(\mathbf{x}, \mathbf{u}) \tag{3.5}$$

This problem is called *Lagrangian subproblem*. The dual function is obtained by minimizing the Lagrangian function subject to the constraints (3.3). Then, the *Lagrangian dual problem* (D) of the problem P is formulated as

$$\max_{\mathbf{u} \geq \mathbf{0}} h(\mathbf{u}) \tag{3.6}$$

where

$$h(\mathbf{u}) = \min_{\mathbf{x} \in \mathbf{X}} \{f(\mathbf{x} + \sum_{i=1}^{m} u_i(b_i - g_i(\mathbf{x}))\} \tag{3.7}$$

It is important to notice that the optimal solution obtained from the Lagrangian subproblem may not be feasible to the primal problem P, since we have eliminated the constraints (3.2). However, for any $\mathbf{u} \geq \mathbf{0}$ it can be proved that $h(\mathbf{u}) \leq f(\mathbf{x}^*)$, where $\mathbf{x}^*$ is an optimal solution to the problem P and $f(\mathbf{x}^*)$ denotes the optimal objective value of P. Thus, for any value of the Lagrangian multipliers $\mathbf{u} \geq \mathbf{0}$, the value of the dual function provides a lower bound on the optimal objective function value of the original problem P.

To obtain the best lower bound for all possible $\mathbf{u}$, the dual problem (3.6) is to be solved. The solution of the dual problem reduces to the search for the maximum of the concave function $h(\mathbf{u})$ over the convex set $\mathbf{u} \geq 0$. Since $h(\mathbf{u}) \leq f(\mathbf{x}^*)$ for any $\mathbf{u} \geq \mathbf{0}$, we can infer the relation between the objective function values of the Lagrangian dual problem D and the primal problem P:

$$h(\mathbf{u}) \leq h(\mathbf{u}^*) \leq f(\mathbf{x}^*) \leq f(\mathbf{x}) \tag{3.8}$$

where $\mathbf{x}$ and $\mathbf{u}$ are the feasible solutions to the primal and the dual problems, respectively. The difference between $f(\mathbf{x}^*)$ and $h(\mathbf{u}^*)$ is called *duality gap*. In general, it is positive, but for linear problems its relation becomes an equality $h(\mathbf{u}^*) = f(\mathbf{x}^*)$.

Under the LR framework, it is well known that the effectiveness of the LR method mainly depends on how to determine the values of Lagrangian multipliers $\mathbf{u}$. The conventional way to do so is to employ the *subgradient optimization* (SO) method [82], which consists of adjusting the Lagrangian multipliers values based on the results of repeatedly solving the Lagrangian problem until the Lagrangian multiplier values converge to a satisfied level. The main difficulty of this algorithm lays on choosing a correct step-size λ_k aiming to ensure algorithm's convergence [145]. However, given that the relaxed Lagrangian problem is still a difficult combinatorial optimization problem, a procedure that requires solving it repeatedly may not be cost-effective.

In order to address this limitations, the method introduced in section 4.2.1 combines the SO algorithm with a heuristic to obtain a feasible primal solution from a dual solution. It can get a better upper bound, so it improves the convergence on the optimal solution starting at an initial upper bound obtained with a *Nearest Neighbor* heuristic. Although optimality may not always be reached, this method is able to provide a feasible solution with a tight duality gap in a reasonable number of iterations.

3.5.2 Clarke and Wright Savings Heuristic

The *Clarke and Wright's Savings* (CWS) constructive heuristic is probably the best known approach to solve the CVRP. The CWS is an iterative method that starts out by considering an initial dummy solution in which each customer is served by a dedicated vehicle. Next, the algorithm initiates an iterative process for merging some of the routes in the initial solution. Merging routes can improve the expensive initial solution so that a unique vehicle serves the nodes of the merged route. The merging criterion is based upon the concept of *savings*.

Consider two customers i and j visited on separate routes. An alternative is to visit the two customers on the same route. Since the CVRP is usually

symmetric, sequences $i-j$ and $j-i$ yield the same results. Because the transportation costs are given, the savings that result from serving both customers in the same or separate routes can be calculated. Denoting the transportation cost between two given customers i and j by c_{ij} (see chapter 2 for notation), the total transportation cost $D_{2routes}$ is

$$D_{2routes} = c_{0i} + c_{i0} + c_{0j} + c_{j0} \tag{3.9}$$

where c_{0i} (c_{i0}) denotes the cost of traveling from (to) the depot to (from) customer i.

Equivalently, the transportation cost of serving both customers in a single route D_{1route} is

$$D_{1route} = c_{0i} + c_{ij} + c_{j0} \tag{3.10}$$

Therefore, by combining the two routes one obtains the savings S_{ij}:

$$S_{ij} = D_{2routes} - D_{1route} = c_{i0} + c_{0j} - c_{ij} \tag{3.11}$$

Relatively large values of S_{ij} indicate that is desirable, with regard to costs, to visit customers i and j on the same route. However, i and j cannot be combined if the resulting tour violates capacity constraints.

With these definitions, it is possible to depict the CWS algorithm. Its steps are outlined in Algorithm 3.7. In the first stage, the savings for all pairs of customers are calculated, and all pairs of customers are sorted in descending order of the savings. Hence, the CWS heuristic follows a *steepest descent* strategy. At each iteration of the merging process, the edge with the largest possible savings is selected from the list (step 1) as far as the following conditions are satisfied: (a) the nodes defining the edge are adjacent to the depot (conditions 2a, 2b, and 2c), and (b) the two corresponding routes can be feasibly merged, i.e. the vehicle capacity is not exceeded after the merging. In the final solution, any customer that have not been assigned to a route during the search should be served by a route that begins at the depot, visits the unassigned customer, and returns to the depot.

The CWS algorithm usually provides relatively good solutions, especially for small and medium-size problems, but it also presents difficulties in some cases. For this reason, many variants and improvements of the CWS have been proposed since it was first defined by Clarke and Wright [46]. For a

Algorithm 3.7 Clarke and Wright's Savings Heuristic

Initialization: calculate the savings S_{ij} for every pair $(i, j), \forall\ i, j \in I$. Create the savings list S, ranking the savings S_{ij} in descending order. Build the initial solution.

Repeat the following sequence until S is empty:

1. Pick the first element S_{ij} from S
2. Include the edge e_{ij} in a route if capacity constraints are not violated, *and* if:
   (a) Neither i nor j have already been assigned to a route $\rightarrow$ Initiate a new route including both i and j
   (b) One of the two customers has already been included in an existing route and that point is not interior to that route $\rightarrow$ Add the edge e_{ij} to that same route
   (c) Both i and j have already been included in two different existing routes and neither point is interior to its route $\rightarrow$ Merge both routes by adding edge e_{ij}.
3. Remove the element S_{ij} from S

comprehensive discussion on the various CWS variants, the reader is referred to [174] and [104]. The randomized variant used in this work is described in detail in section 4.3.1.

Chapter 4

Hybrid Methodologies

As explained in Chapter 2, solving a VRP consists of determining a set of routes whose total travel cost is minimized and such that all side constraints are satisfied. In some cases, the fleet size is not fixed and minimizing the total number of used vehicles becomes an additional objective.

In this chapter, three hybrid methodologies aimed to solve the VRP are described. The first two approaches are focused on solving the most basic — yet complicated from a computational point of view— VRP, the CVRP. They are based on a VNS framework where CP and LR techniques are embedded. The third methodology, based on the CP paradigm, is able to deal with more complex VRP variants, such as the VRPTW.

The CVRP has been defined in two different ways according to the methodology used to solve it. From the hybrid VNS perspective, the CVRP has been divided into two subproblems, concerning customers' allocation and routing optimization separately. On the other hand, a complete formulation has been developed to tackle the problem by using CP techniques. Both formulations are introduced in sections 4.1 and 4.4.1, respectively.

After introducing the problem formulation, section 4.2 is devoted to the hybrid VNS methodology adopted as the initial framework. The multi-start approach explained in section 4.3 is an improved evolution of this initial method. Finally, a complete CP-based methodology is described in section 4.4. This hybrid approach combines CP characteristics, such as constraint propagation, with a metaheuristics framework to guide the search. In addition, some techniques to improve its efficiency are introduced at the end of this chapter.

4.1 Hybrid CVRP formulation

As said before, in the present hybrid approach the CVRP has been divided into two subproblems, concerning customers' allocation and routing optimization separately. The first is aimed to assign customers to vehicles fulfilling capacity limitations. The latter is used to solve each independent route to optimality, giving the best solution for a particular allocation. Thus, routing optimization process can be viewed as solving a set of m independent symmetric TSPs. CP is used to find a feasible solution in terms of capacity, while routing problems are solved by using LR.

Capacity problem

The proposed capacity subproblem definition uses the following variables:

- $R = R_1, ..., R_n$ with an integer domain $[1..m]$
- $Q_v = Q_1, ..., Q_m$ with a real domain $[0..Q]$

R is a list of n variables, corresponding to the n customers. Each R_i value indicates which vehicle is serving the i^{th} customer, and so it can take values from 1 to m. R_1 value is not relevant, since it corresponds to the depot and has no associated demand. However, it is included for simplicity reasons on defining variables lists.

Q_v is a list of m variables used to trace the cumulative capacity at each of the m routes. Capacity constraints are enforced through domains definition, forcing each Q_v to take positive values up to a maximum corresponding to vehicles' capacity Q. Thus, the definition of additional constraints to impose capacity limitations for each vehicle is avoided through an appropriate domains' definition. Hence, the use of CP and its characteristics lead to a simpler and neater formulation of the capacity subproblem.

A set of dimension $m \times n$ of binary variables B has been introduced to relate R and Q_v values. For each vehicle $v \in V$ $(v = 1, ..., m)$, a list of n binary variables B_{vi} $(i = 1, ..., n)$ is defined, taking value 1 whenever customer $i \in I$ is assigned to vehicle v and 0 otherwise. Since each customer i is visited by a single vehicle, for all values of v the binary variable B_{vi} can take value 1 only once, that is, only one element per column can take value 1. This

constraint is expressed in terms of the global constraint *occurrences*, included as a built-in predicate in most CP platforms such as the software ECLiPSe [9].

$$occurrences(1, B_{vi}, 1)\ \forall v \in V \tag{4.1}$$

Expression (4.1) states that value 1 can occur only once in the list of variables B_{vi}, i.e. for a fixed value i, only one of the m elements of the list B_{vi} can take value 1. Predicate *occurrences* may be seen as an implementation of the general global constraint *cardinality* [21].

Using global constraints increases the search efficiency. Whenever a variable is instantiated during the search process, propagation mechanisms reduce uninstantiated variables' domains to some degree [28]. Global constraints ensure a faster reduction of domains through specifically programmed propagation methods. Moreover, they allow a clean and fast definition of constraints patterns for sets of variables of any size.

The binary set B and allocation variables R are related through the following statement:

$$R_i = r_i \rightarrow B_{r_i i} = 1\ \forall i \in I \tag{4.2}$$

Expression (4.2) states that the i^{th} element of the r_i list of B will have value 1 whenever the i^{th} component of R takes value r_i. Global constraint (4.1) ensures propagation so all values of $B_{vi} \mid v \in \{1, ..., m\} \setminus r_i$ are set to 0 automatically. Therefore, cumulative capacities can be traced simply by using the following equation:

$$Q_v = \sum_{i \in I} B_{vi} q_i\ \forall v \in V \tag{4.3}$$

The proposed formulation is used to find a partial initial solution fulfilling capacity constraints. By solving resultant routing problems, which are always feasible because they do not contain any additional constraints, a complete solution may be easily obtained in most cases. Thus, capacity problem's goal is to find a feasible solution with the minimum number of required vehicles. With this objective, a depth-first search method is applied to find a feasible solution that uses all available vehicles. A vehicle is removed from the list and the process is repeated recursively. The algorithm stops when unfeasibility is reached, returning the last feasible solution found in the previous iteration.

The described formulation is also used to check feasibility any time a new candidate partial solution is generated. During diversification processes, as well as in local search, new points in the search space are generated within the neighborhood of the current solution, according to the defined moves. Before any further calculation is done, this CP model is used to check the feasibility of the new candidate partial solution. If it is feasible, the cost is updated by calculating the corresponding new routes. Otherwise, the point is rejected and either the diversification process or the local search is resumed.

Routing problem

The routing problem, tackled for each vehicle separately, can be viewed as a *Traveling Salesman Problem* (TSP) instance. The TSP is probably the best known combinatorial problem: "A salesman is required to visit once and only once each of n different customers starting from a depot, and returning to the depot. What path minimizes the total distance travelled by the salesman?" [22]. The CVRP and the TSP are closely related: the CVRP is usually defined as a generalization of the TSP. Similarly, the TSP can be viewed as a simplified CVRP with one single vehicle without capacity limitations.

According with the notation introduced in section 2, the TSP can be formulated over subsets of the defined variables. For each vehicle $v \in V$, the related TSP can be considered as a complete undirected graph $G = (I_v, E_v)$, connecting assigned customers $I_v = \{i \in I \mid R_i = v\}$ through a set of undirected edges $E_v = \{(i,j) \in E \mid i,j \in I_v\}$. The solution is a path connected by edges belonging to E_v that starts and ends at the depot ($i = 1$) and visits all assigned customers.

Since there are no capacity limitations, a feasible solution of the TSP should, by definition, satisfy the CVRP constraints enforcing that each customer is visited exactly once and that each route starts and ends at the depot. The TSP goal is minimizing the total travel cost of the route.

The proposed mathematical formulation requires defining the binary variable x_e to denote that the edge $e_{ij} \in E_v$ is used in the path:

$$x_e = \begin{cases} 1 & \text{if customer } j \text{ is visited immediately after } i \\ 0 & \text{otherwise} \end{cases}$$

The proposed mathematical formulation for the TSP problem is as follows:

$$min \sum_{e \in E_v} c_e x_e \tag{4.4}$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2 \; \forall i \in I_v \tag{4.5}$$

$$\sum_{e \in E_v(V)} x_e \leq |V| - 1 \; \forall V \subset I_v, \; |V| \leq \frac{1}{2}|I_v| \tag{4.6}$$

where

- $\delta(i) = \{e \in E_v \;\mid\; \exists j \in I_v, \; e = (i,j) \; or \; (j,i)\}$ represents the set of arcs whose starting or ending node is i;
- $E_v(V) = \{e = (i,j) \in E_v \;\mid\; i,j \in V\}$ represents the set of arcs whose nodes are in the subset of vertices V.

The objective function (4.4) aims to minimize the total cost of the route, being c_e the associated cost to the undirected edge e (e_{ij} or e_{ji}).

Constraint (4.5) states that every node $i \in I_v$ must be visited once. Being $\delta(i)$ the subset of arcs whose starting or ending node is i, expression (4.5) constraints every customer $i \in I_v$ to have only two incident edges.

Subtour elimination constraint (4.6) states that the tour must be a Hamiltonian cycle [120], i.e. it cannot have any subcycle. This constraint avoids any subcycle of the subset $V \subset I_v$, since the number of edges must be lower than the size of the subset. It only considers the subsets V which $|V| \leq \frac{1}{2}|I_v|$. For any solution containing more than one subcycle, at least one of the subcycles will connect a number of vertices V such that $|V| \leq \frac{1}{2}|I_v|$ is fulfilled. Constraint (4.6) ensures that these subcycles are banned and, in consequence, all subcycles are avoided.

4.2 General Variable Neighborhood Search

The CVRP, decomposed and formulated as described in section 4.1, has been tackled using a hybrid approach. The proposed methodology combines CP and

LR within a VNS framework in order to improve algorithm's performance. As mentioned in section 2.1.2, even the most basic VNS algorithm, known as VND (see section 3.3.1), has provided promising results when solving different VRP variants. In the proposed approach, a general VNS framework (see section 3.3.3) has been chosen to embed the selected paradigms.

In any case, other well-known metaheuristics could have been used to embed CP and LR, such as Tabu Search (TS) or Genetic Algorithms (GA). Both metaheuristics have been widely used for tackling different VRP variants obtaining good results (see section 2.1.2). However, VNS permits overcoming some of their limitations. On the one hand, TS is based on a local search where the process may lead to worse solutions in order to escape from local minima. It may be comparable to the VND algorithm, but the latter has the advantage of alternating different moves to explore the search space. Swapping these neighbourhoods structures allows escaping from local minima in a more natural manner and avoids defining and tuning tabu lists and aspiration criteria. The latter parameters affect algorithm's behavior and are critical for TS performance. Moreover, by using a general VNS scheme, a diversification process is naturally introduced and integrated within the algorithm, so search is restarted at each iteration from a point obtained from the best solution found so far. This diversification process may be tuned so the algorithm behaves conservatively or following a multi-start strategy. On the other hand, GA require defining some parameters that become critical for algorithm's performance, such as the population size or crossover and mutation ratios. According to problem dimensions, managing correctly the memory used to store population data may become a major issue. In some cases, reducing the population size may lead to misleading results [144]. Thus, finding a trade-off between efficiency and effectiveness often needs a fine-tuning process that is not required when using a VNS algorithm. For these reasons, the general VNS has been selected as the main structure that leads the search process in the methodologies presented in this thesis.

Within the general VNS framework, CP and LR are used in different processes. During algorithm's initialization, CP is used to find an initial feasible solution by means of capacity constraints. CP is also used to check solutions feasibility within diversification and local search processes.

In turn, a tailored LR method is applied to calculate routes every time a partial solution is generated either during initialization, diversification or local search processes. Using LR allows reducing the computation time when compared to other routing post-optimization methods, such as a VND with

single-route classical moves, e.g. classical 2-opt or 3-opt. So, the proposed LR approach provides optimal routes in very low times and, at the same time, permits reducing algorithm's definition and complexity.

4.2.1 Tailored Lagrangian Relaxation method

As mentioned in section 3.5.1, LR is a well-known method that works by moving hard-to-satisfy constraints into the objective function, associating a penalty in case they are not satisfied.

LR exploits the structure of the problem, so it reduces considerably problem's complexity. Thus, the Lagrangian Problem needs less computational effort to be solved. However, as mentioned, it is often a major issue to find the optimal Lagrangian multipliers. The commonly used approach is the Subgradient Optimization (SO) method [82]. It guarantees convergence, but it is too slow to become a method of real practical interest.

Given the assigned customers to each vehicle, the LR-based method is used to solve the associated routing problems. In this proposed approach, LR relaxes the constraints set requiring that all customers must be served (4.5), presented in section 4.1:

$$\sum_{e \in \delta(i)} x_e = 2 \ \forall i \in I_v \tag{4.5}$$

Constructing the solution x as a *1-tree* permits avoiding all subcycles. A 1-tree can be defined as a tree on the graph induced by nodes $\{2, ..., n_v\}$ plus two incident edges at node 1 [82]. Actually, a feasible solution of the TSP is a 1-tree having two incident edges at each node.

The Lagrangian Dual problem obtained from the TSP formulation presented in section 4.1, moving into the objective function equalities (4.5), is:

$$\max_{\mathbf{u} \in \Re^{|I_v|}} L(\mathbf{u}) \tag{4.7}$$

where the Lagrangian function is:

$$L(\mathbf{u}) = \min_{x_{1-tree}} \sum_{e \in E_v} c_e x_e + \sum_{i \in I_v} u_i (2 - \sum_{e \in \delta(i)} x_e) \tag{4.8}$$

This LR relaxes constraints (4.5) weighting them with a multiplier vector $\mathbf{u}$ of appropriate dimension and unrestricted sign, defining the subgradient:

$$\gamma_i = 2 - \sum_{e \in \delta(i)} x_e \ \forall i \in I_v \tag{4.9}$$

Finding a *minimum spanning tree* induced by nodes $\{2, ..., n_v\}$ is relatively easy, so the presented relaxation becomes potentially interesting. A minimum spanning tree is a minimum cost *spanning tree*, i.e. a tree connecting all the vertices, in a connected, undirected graph. The commonly used technique for finding a minimum spanning tree is Prim's algorithm [103]. Its time complexity is $O(n^2)$, where $n = |I_v| - 1$ is the number of vertices involved, excluding the depot.

The SO is an iterative method used to solve the Lagrangian problem finding a maximum value of the lower bound [83]. The main difficulty of this algorithm lays on choosing a correct step-size λ_k [178]. This is a critical choice, since the convergence can be highly influenced by this parameter [145].

The LR-based method used in this thesis to tackle the TSP problem associated to each route improves the convergence on the optimal solution of the SO by using a heuristic which obtains a feasible solution from a LR solution. If the optimal solution is not reached at a reasonable number of iterations, the proposed method is able to provide a feasible solution with a tight gap between the primal and the optimal cost. This approach is explained in detail by Herrero et al. [84].

Being LB a dual lower bound and L^* the optimal value, so $LB \leq L^*$, the step-size λ_k at iteration k is defined according to the expression:

$$\lambda_k = \delta_k \frac{LB - L(\mathbf{u}^k)}{\| \gamma^k \|^2}, \ 0 < \delta_k \leq 2 \tag{4.10}$$

Then, $L(\mathbf{u}^k) \to L$, or the algorithm finds $\mathbf{u}^k$ with $LB \leq L(\mathbf{u}^k) \leq L^*$ for some finite k. In practice, LB is typically unknown and it is more likely to know a good primal upper bound $UB \geq L^*$. Such an upper bound UB is then used initially in place of LB. However, if $UB \gg L^*$, the term $UB - L(u^k)$ in the numerator will not tend to zero, and so sequences $\{\mathbf{u}^k\}$ and $\{L(\mathbf{u}^k)\}$ will not converge. In order to find a feasible solution of the TSP, which may give an accurate UB, a Nearest Neighbor Heuristic is applied. This method is commonly used with this purpose, since it is computationally efficient and easy to implement.

The used LR-based method is implemented within the local search process to solve routing subproblems to optimality. It can be considered a specification of the Lagrangian Metaheuristic presented on Boschetti and Maniezzo [30]. As mentioned, it uses the SO algorithm combined with a heuristic able to obtain a feasible solution from the dual variable, aiming to improve algorithm's convergence to the optimal solution [84]. This method tries to improve the UB with the values of these feasible solutions, so a better convergence is obtained. Eventually, this feasible solution found by the heuristic may be provided as the best solution if the method is stopped. The stopping criterion is based on the maximum number of iterations ($k < max_{iterations}$) and also on a floating-point exception on the step-size ($\lambda_k < 10^{-15}$). The described tailored LR-based method is shown in Algorithm 4.1.

Algorithm 4.1 Tailored LR-based method for the TSP

Inicialization:

- Initialize parameters $\mathbf{u}^0 = \mathbf{0}$; $\delta_0 = 2$; $\alpha_L = 1/3$
- Obtain an UB applying Nearest Neighbor Heuristic
- Initialize $\overline{L} = L(\mathbf{u}^0) + \alpha_L(UB - L(\mathbf{u}^0))$

Repeat the following sequence until $k < max_{iterations}$ or $\lambda_k < 10^{-15}$:

1. Solve the Lagrangian function $L(\mathbf{u}^k)$
2. Check the subgradient $\gamma_i^k = 2 - \sum_{e \in \delta(i)} x_e$
3. If $\| \gamma^k \|^2 = 0$ then the optimal solution is found $\Rightarrow$ EXIT
4. If $\| \gamma^k \|^2 < \zeta$ then apply a heuristic to improve the UB
5. Update the parameter $\overline{L}$
6. Calculate the step-size $\lambda_k = \delta_k \frac{\overline{L} - L(\mathbf{u}^k)}{\|\gamma^k\|^2}$
7. Update the Lagrangian multipliers $\mathbf{u}^{k+1} = \mathbf{u}^k + \lambda_k \gamma^k$
8. $k \leftarrow k + 1$

The proposed heuristic to improve the UB is applied when the solution is nearly a route, i.e. if the subgradient γ is below a certain value ζ, $\parallel \gamma^k \parallel < \zeta$ (step 4). As any solution is a 1-tree, this criterion means that the solution has few vertices without two incident edges. This heuristic replaces an edge e_{ij} where the vertex j has some extra incident edges for an edge e_{il} where the vertex l has one single incident edge. Before applying the exchange, the heuristic checks if the new solution is a 1-tree. Otherwise, the heuristic can divide it into more trees having some subtours. The chosen vertices i, j, l minimize the cost of the exchange:

$$\{i, j, l\} = \text{argmin}\{c_{il} - c_{ij} : \ \gamma_j < 0, \gamma_l > 0, \gamma_i \leq 0, x_{ij} = 1, x_{il} = 0\} \quad (4.11)$$

The parameter ζ depends on the number of variables. A good estimation of ζ value would avoid increasing the computation time. First, its value may be large, for instance $|I_v|/2$, but it should be updated whenever a feasible solution is found according to $\zeta = \parallel \gamma^k \parallel^2$. If this parameter is not correctly updated, the heuristic becomes time consuming. Eventually, the heuristic could find the optimal solution without detecting it, so the method would continue iterating until $LB = UB$.

As mentioned, algorithm's convergence is critically influenced by the step-size λ_k. This value relies on either the LB or the UB, which are normally unknown or bad estimated. Therefore, convergence may not be assured for all cases. In order to overcome this limitations, the use of a parameter $\overline{L}$, such that $LB \leq \overline{L} \leq UB$, is proposed. By definition, this parameter corresponds to a better estimation of the optimum L^* than those obtained for LB and UB. Therefore, the calculation of the step-size turns into the expression (4.12) (step 6):

$$\lambda_k = \delta_k \frac{\overline{L} - L(\mathbf{u}^k)}{\parallel \gamma^k \parallel^2} \quad (4.12)$$

The convergence is guaranteed if the term $\overline{L} - L(\mathbf{u}^k)$ tends to zero. In turn, convergence efficiency can be improved as long as the new parameter gets closer to the (unknown) optimal solution. The main idea is very simple: as the algorithm converges to the solution, new better lower bounds are known and new better upper bounds estimations can be obtained by using the heuristic designed to get feasible solutions. Therefore, the parameter $\overline{L}$ is updated according to the following conditions:

- It is initialized $\overline{L} = L(\mathbf{u}^0) + \alpha_L(UB - L(\mathbf{u}^0))$ with $0 < \alpha_L < 1$;
- If $L(\mathbf{u}^k) > \overline{L}$, it is updated $\overline{L} = L(\mathbf{u}^k) + \alpha_L(UB - L(\mathbf{u}^k))$;
- If $\overline{L} > UB$, then $\overline{L} = UB$.

Finally, the parameter δ_k is initialized to the value 2 and is updated as Zamani and Lau [181] suggest. If the lower bound is not improved, δ_k is decreased, using the formula (4.13):

$$\delta_{k+1} = \delta_k \rho,\ 0 < \rho < 1 \tag{4.13}$$

On the other hand, if the lower bound is improved, then δ_k is increased according to the formula (4.14):

$$\delta_{k+1} = \delta_k \frac{3-\rho}{2},\ 0 < \rho < 1 \tag{4.14}$$

provided that $0 \leq \delta_k \leq 2$ to ensure convergence.

4.2.2 Movements definition

The VNS metaheuristic is based on the exploration of different neighborhoods around a given feasible solution (see section 3.3). In order to establish these neighborhoods, different moves are defined. In our approach, four different inter-routes classic moves [158], represented in Figure 4.1, have been defined so they can be used within diversification and local search processes:

(a) **Relocate**: moves a customer from one route to a different one.

(b) **Swapping**: exchanges two customers belonging to two different routes.

(c) **Chain**: swaps sections of two contiguous customers from two different routes.

(d) **Ejection chain**: swaps the end portions of two different routes.

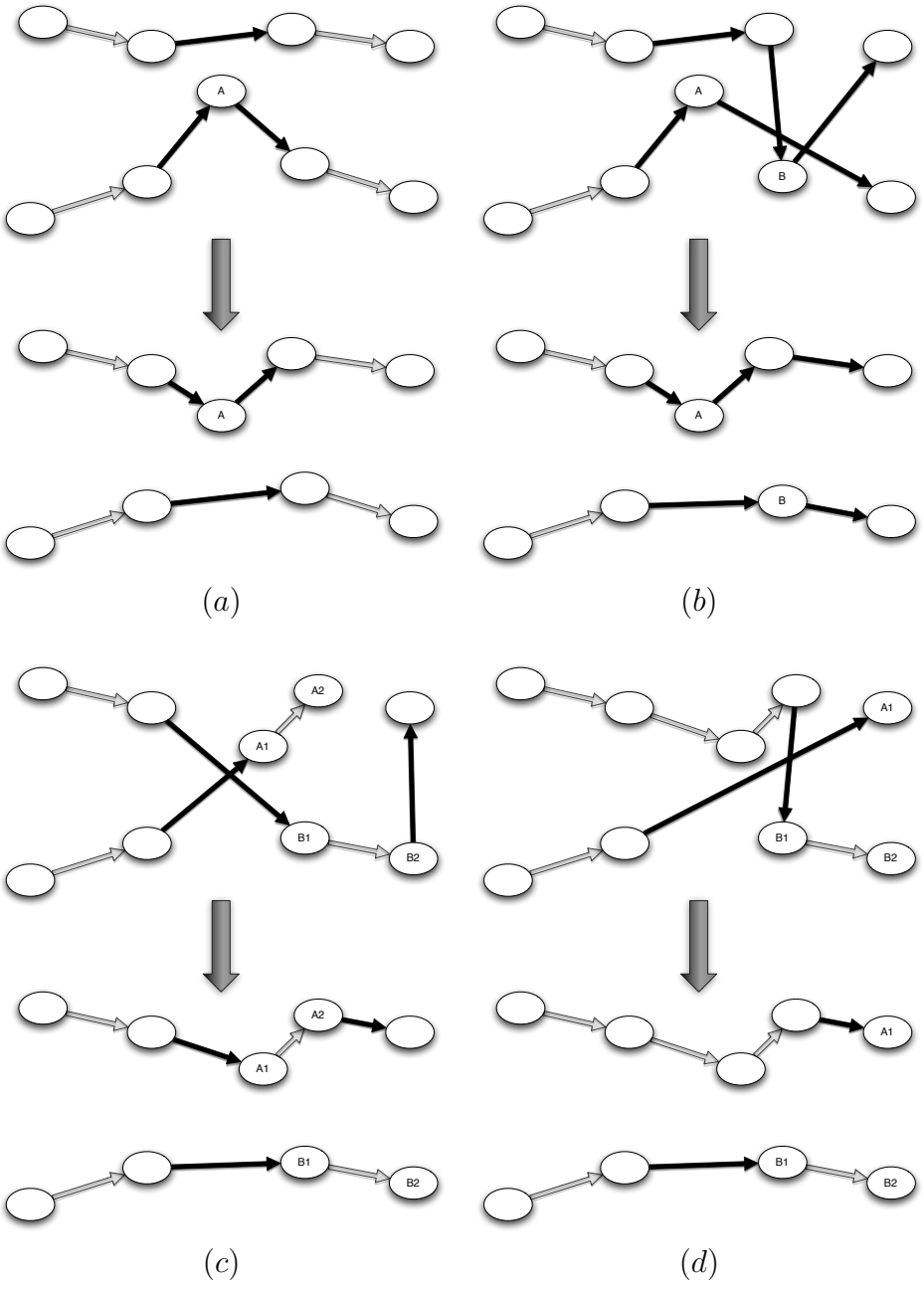

Figure 4.1: Inter-route movements used in our approach: (a) Relocate, (b) Swapping, (c) Chain, and (d) Ejection Chain.

The *relocate*, *swapping*, and *ejection chain* moves can be characterized as 2-opt methods, although in some cases they involve the deletion and generation of more than two arcs. This is because only two arcs have to be determined to fully describe a given move. As an example, the relocate move depicted in Figure 4.1(a) can be fully described by the deletion of the arc in the upper route and the one ending at the node A, while the deleted arc departing from this node is implicitly defined by the move mechanism. Analogously, the swapping and ejection chain moves represented in Figure 4.1(b) and 4.1(d), can be fully described by the deleted arcs ending at nodes A and B (swapping), or $A1$ and $B1$ (ejection chain). The computational complexity of exhaustively exploring these neighborhoods is $O(n^2)$. A similar reasoning can be used to describe the *chain* move as a specialization of the 3-opt operator, whose computational complexity is $O(n^3)$. For a further discussion on this subject, the reader is referred to section 4.2.4.

Usually, a post-optimization method based on intra-route moves is applied to improve each single route quality [155]. The use of LR ensures the partial optimality of most solutions from the routing perspective. The reason is that, since we are considering a relatively small number of customers per route, the LR-based method described in section 4.2.1 can quickly find the optimal solution to most routing problems. In effect, the respective lower bounds and upper bounds converge rapidly, keeping their gap between 0 and 10^{-10}, which guarantees the solution optimality. In addition, LR solves all routes in negligible times, while reducing local search complexity (see section 4.2.4). Thus, LR is an efficient alternative for intra-route optimization processes and avoids defining intra-route moves.

4.2.3 Variable Neighborhood Search framework

As a first approach, a general VNS framework, as presented in section 3.3, has been implemented embedding the described methods. At each iteration, a local minimum is reached departing from an initial solution. A diversification process (*shaking*) ensures that different regions from the search space are explored by changing the initial solution at each iteration. This diversification process allows escaping from the valleys surrounding the local minima found during search. Algorithm 4.2 and Figure 4.2 outline the main steps of our approach, clearly identifying where CP and LR methods are used. The stopping criterion is chosen to be based on the maximum number of iterations.

Algorithm 4.2 General VNS hybrid approach

Initialization:
- Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$, that will be used in the shaking phase, and the set of neighborhood structures N_l, for $l = 1, ..., l_{max}$, that will be used in the local search;
- Find an initial solution x: Use CP to assign customers to vehicles and LR to calculate the corresponding routes;
- Choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Shaking: generate a point x' at random from the k^{th} neighborhood of x ($x' \in N_k(x)$). Use CP to check feasibility and LR to calculate the cost of modified routes.
   (b) Local Search by VND:
      i. Set $l \leftarrow 1$.
      ii. Repeat the following steps until $l = l_{max}$:
         A. Exploration of neighborhood:
            - Find all neighbors x'' of x' ($x'' \in N_l(x')$);
            - Check feasibility of capacity constraints using CP;
            - Calculate the cost of modified routes using LR;
            - Choose the best neighbor x'' of x' ($x'' \in N_l(x')$).
         B. Move or not: if the solution x'' thus obtained is better than x' ($f(x'') < f(x')$), set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise, set $l \leftarrow l + 1$.
   (c) Move or not: if the local optimum x'' is better than the incumbent x ($f(x'') < f(x)$), move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.

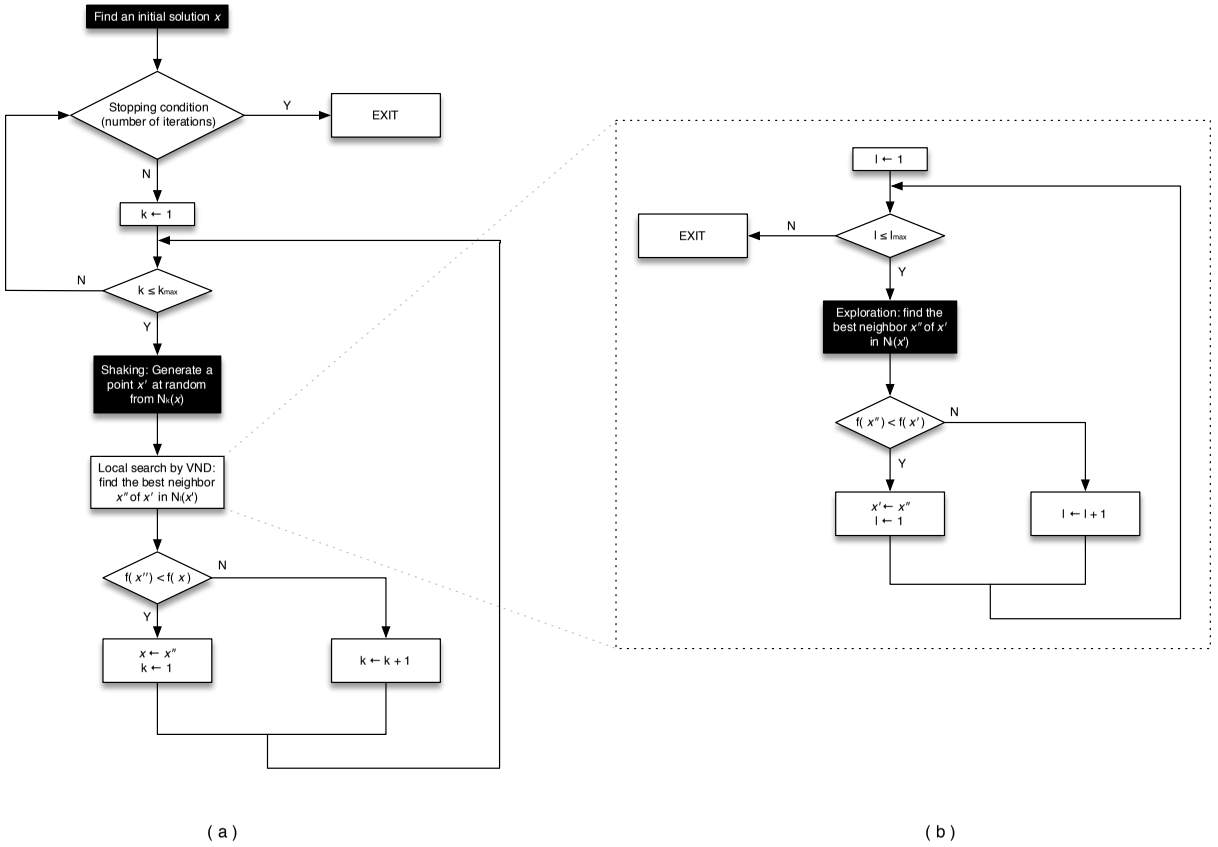

Figure 4.2: Flow chart of the General VNS hybrid approach, representing (a) the general VNS framework, and (b) the VND algorithm used for local search. The processes that use the described CP and LR methods are highlighted.

In this approach, the four moves described in the previous section have been selected to be used in the shaking and local search neighborhoods. As a first step in the algorithm, an initial feasible solution is found using CP and LR. CP is used to assign all customers to available vehicles fulfilling capacity constraints, while resulting routes are solved to optimality by means of LR.

Every time a new point is randomly generated from the k^{th} neighborhood $N_k(x)$ of x in order to diversify the search, its feasibility is immediately checked using CP. If the generated point is unfeasible, the process is repeated until a new feasible point is found. However, if the valley surrounding the solution x is large, a thorough diversification should be done aiming to avoid getting trapped in a local optimum. For this reason, the implemented shaking process is repeated several times, tracing a path of feasible solutions in the search space. The cost of these feasible solutions is ignored until the last iteration, when routes are recalculated using LR to provide a complete solution. Hence, the time required for the diversification is reduced while keeping solutions' feasibility.

The local search process is performed by means of a VND algorithm (Algorithm 3.1 in section 3.3.1). It should be remarked that it takes advantage of all neighborhood structures for $l = 1, ..., l_{max}$, instead of applying a simple local search method. Thus, the VND approach increases the chances to reach a global minimum with respect to greedy local search methods. In addition, no parameters are to be introduced, unlike other metaheuristic methods such as TS. In any case, the VND may eventually get trapped in a local optimum. For this reason, VND is embedded within a diversification/local search iterative process, i.e. the VNS framework.

Within the VND algorithm, an exhaustive exploration of the l^{th} neighborhood $N_l(x')$ of x' is performed. Departing from the solution x', the l^{th} move is applied and new solution's feasibility is checked using CP. Whenever it is proved feasible, LR is used to recalculate only modified routes. This approach permits to consider only two routes per solution, reducing the computation time. Finally, the best neighbor x'' is chosen in terms of its solution value $f(x'') = \sum_{v=1}^{m} UB_v$.

A slightly different approach has also been considered for the exhaustive exploration. In this case, CP consistency techniques (see section 3.2.1) are applied to get feasible domains for the variables to be modified, i.e. when moving customers, only those routes with enough spare capacity to include them are allowed. Thus, only values corresponding to feasible solutions are explored and capacity constraints do not need to be checked afterwards. This approach

provides the same results, but the calculation time is dramatically increased due to consistency algorithms' high complexity. However, tailored propagation and consistency techniques could lead to an important time reduction and so it is a promising line for future research work.

4.2.4 Computational complexity reduction of local search

The mechanisms of evaluating solution neighborhoods is a crucial aspect on the efficiency of local search methods. The neighborhood of a given solution consists of every solution generated from it, by performing a certain movement. The simplicity of these movements is an objective mainly for computational reasons: the population of generated solutions (*neighborhood cardinality*) should be limited to manageable levels, and the evaluation of the neighboring solutions quality should require a reasonable (and usually constant) time.

In local search methods, to pass from one solution to the subsequent one, an exhaustive examination of the neighboring solutions is performed. Then, the highest quality neighboring solution is chosen and the current solution is updated. The computational complexity of a local search method is defined by the number of calculations needed for exhaustively evaluating the neighborhood of a candidate solution. The computational time required per iteration is mainly determined by the neighborhood cardinality. Therefore, the computational time is bounded by a polynomial function of the instance size.

In the present thesis, we propose a strategy for reducing the computational complexity required for exhaustively exploring the neighborhoods within the local search processes. This strategy is similar to the one proposed by Zachariadis and Kiranoudis [180]. The central idea for achieving this complexity reduction is straightforward: when moving from one solution to the next one, only a small part of the incumbent solution is modified. Therefore, to examine the next solution neighborhood, only the tentative moves affecting the modified parts are to be explored. On the other hand, moves that refer to unchanged parts of the solutions have already been evaluated in previous neighborhood explorations. If they have been appropriately recorded, their recalculation is therefore unnecessary.

In order to explain the applied strategy for reducing the computational complexity in local search processes, we will consider the *relocate* movement introduced in section 4.2.2. In any case, the discussion can be extended to the remaining moves, i.e. *swapping*, *chain*, and *ejection chain* movements.

The relocate move can be seen as the traditional local search operator 2-opt. In general, the complexity of exhaustively examining the k-exchange neighborhood of a solution is $O(n^k)$, where n is the number of customers. As a result, the cardinality of the 2-opt neighborhood is $O(n^2)$, and so the computational effort required for exhaustively examining this neighborhood is $O(n^2)$.

By using the LR approach for the TSP, we can reduce the size of the relocate neighborhood. The traditional 2-opt operator considers every position where a customer can be inserted, either in the same route or in a different one. Applying the LR method for routes calculation permits obviating the exact point of insertion. Thus, provided that the capacity constraints are fulfilled, customers are assigned to a route and LR returns the optimal position within the route for each one. This way, the size of the relocate neighborhood is reduced to $O(n \cdot m)$, where n is the number of customers and m the number of considered routes. Since $m \ll n$ for most instances, search complexity is significantly reduced, and so is the required computational time with respect to classical approaches.

The size of the relocated neighborhood can be further reduced by considering how a solution is modified after an improving movement is applied. When a higher quality neighboring solution is found, a customer is moved from one route to another one and inserted in the best possible position. Therefore, only two routes are modified, while the rest remains unchanged. Exhaustively exploring the new solution neighborhood would imply checking again all possibilities between every two routes, repeating evaluations that have already been done in previous explorations. Hence, it would be more efficient to evaluate only the two changed routes against all others.

This approach reduces the neighborhood search space to be explored, keeping a linearithmic complexity in respect to the instance size. However, since we apply a best neighbor strategy, it may be other improving movements between two other different routes that are rejected. If only the two modified routes are checked in the next iteration, the search will avoid these movements and we may be loosing such improvements. For this reason, a slightly different strategy is adopted: all improving moves are recorded and ranked in descending order according to its quality. Starting from the best one, these moves are successively applied whenever they do not affect already modified routes, i.e. if two moves affect the same route, we only apply the one leading to a higher improvement. As in the previous approach, only those routes that have been modified are considered in the next iteration. Therefore, all improving moves

that were rejected because there was a better one affecting the same route are reevaluated, while calculations related to not improving moves are omitted.

The computational complexity of this approach remains linear on the instance size. Furthermore, this approach improves the local search convergence to a local minimum. In the general case, considering that k routes have been modified in the previous iteration, the required computational effort is

$$O((k+1)n + (m-k-1)\sum_{i \in K} n_i)$$

where K represents the set of modified routes and k is its cardinality ($2 \leq k \leq m$). Since the number of customers allocated in a route may change at each iteration, so does K, the term $\sum_{i \in K} n_i$ may change at each step. Nevertheless, it is clear that $\sum_{i \in K} n_i \leq n$. In the worst case, when all routes are modified ($k = m$), we retrieve the computational complexity related to an exhaustive exploration $O(n \cdot m)$.

Using the proposed hybrid approach provides advantages for reducing the computational effort required to solve the CVRP. On the first hand, using the LR method to solve the routing subproblems permits updating the solution cost by recalculating only the costs related to the modified routes. Thus, if all routes costs are appropriately recorded, the solution cost can be updated in constant time. On the other hand, the proposed decomposition and using the CP model for the capacity subproblem permits checking moves' feasibility before any calculation is done. This way, in practice, the required computational time to explore the relocate neighborhood is reduced and remains below the bounds discussed above.

The ideas presented in this section have been implemented within the VNS frameworks described in this chapter. The corresponding modifications of the general VNS hybrid approach described in section 4.2.3 are shown in Algorithm 4.3. Results show that our strategy is efficient and achieves the desired behavior. These results are further discussed in the corresponding section 5.1.1.

Algorithm 4.3 General VNS hybrid approach with improved local search efficiency

Initialization:
- Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$, that will be used in the shaking phase, and the set of neighborhood structures N_l, for $l = 1, ..., l_{max}$, that will be used in the local search;
- Find an initial solution x: Use CP to assign customers to vehicles and LR to calculate the corresponding routes;
- Initialize the set *LastModified* $\leftarrow V$;
- Choose a stopping condition.

Repeat the following sequence until the stopping condition is met:

1. Set $k \leftarrow 1$.
2. Repeat the following steps until $k = k_{max}$:
   (a) Shaking: generate a point x' at random from the k^{th} neighborhood of x ($x' \in N_k(x)$). Use CP to check feasibility and LR to calculate the cost of modified routes.
   (b) Local Search by VND:
      i. Set $l \leftarrow 1$.
      ii. Repeat the following steps until $l = l_{max}$:
         A. Exploration of neighborhood:
            - Find all neighbors x'' of x' ($x'' \in N_l(x'$, *LastModified*$)$);
            - Check feasibility of capacity constraints using CP;
            - Calculate the cost of modified routes using LR;
            - If the solution x'' is better than x' ($f(x'') < f(x')$), include it in a list of improving changes.
         B. Choose the best compatible neighbors:
            - Set *LastModified* $\leftarrow \emptyset$;
            - Sort the list of improving changes;
            - Apply the first improving change;
            - Add in descending order the next compatible improvements;
            - Add the modified routes identifiers to *LastModified*.
         C. If the list is empty, set $l \leftarrow l + 1$ and *LastModified* $\leftarrow V$; otherwise, set $x' \leftarrow x''$ and $l \leftarrow 1$.
   (c) Move or not: if the local optimum x'' is better than the incumbent x ($f(x'') < f(x)$), move there ($x \leftarrow x''$), and continue the search with N_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.

4.3 Multi-Start Variable Neighborhood Search

In this section, we describe a *Multi-Start Variable Neighborhood Descent* (VND) method, based on the VND algorithm described in section 3.3.1, to tackle the CVRP. The local search process of the VND is supported by CP and LR, as in the hybrid approach presented in the previous section. Again, CP is used to check solutions feasibility within the local search process, while the LR-based algorithm is used to efficiently find the optimal routing solution for each transportation resource. Using the CP paradigm provides the desired flexibility in case additional constraints are required to model other operational constraints, beyond vehicle capacity. A probabilistic *Clarke and Wright Savings* (CWS) (see section 4.3.1) constructive method is used to generate initial solutions. This algorithm provides different good quality solutions that are used as seeds to launch the exploration of different regions of the search space. Therefore, algorithm's probabilistic behavior introduces a natural diversification mechanism and turns the schema into an approach likely to be parallelized.

Although solutions' quality is comparable for both methodologies, the Multi-Start VND approach presented in this section is much more competitive with other state-of-the-art metaheuristics in terms of computational time (see section 5.1.2) than the hybrid approach described in section 4.2. Methodology's efficiency has been significantly enhanced by including a multi-start procedure which makes use of a randomized CWS heuristic in order to quickly provide a set of different "good" initial solutions, over which a flexible local-search process is applied. Thus, the VNS diversification procedure is substituted by a multi-start approach, where different regions are explored thanks to the diversity of solutions provided by the randomized CWS algorithm. The local search process has also been enhanced with respect to the initial hybrid VNS approach by incorporating the new mechanisms for reducing computational complexity described in section 4.2.4. Finally, the methodology described in this section has been parallelized to improve its efficiency.

It can be notice that this approach has some similarities with the *Greedy Randomized Adaptive Search Procedure* (GRASP) [61]. GRASP is a typically two-phase approach where in the first phase a constructive heuristic is randomized. The second phase includes a local search phase. In the approach presented in this section, we use a randomized version of the classic CWS heuristic to generate good initial solutions, which are afterwards improved by means of a VND method. In addition, these processes are performed in a

parallel multi-start fashion.

4.3.1 Randomized Clarke and Wright Savings Heuristic

As discussed in section 3.5.2, in the classic Clarke and Wright Savings (CWS) algorithm, the edge with the largest possible savings is selected from the list at each iteration of the merging process, as far as the following conditions are satisfied: (a) the nodes defining the edge are adjacent to the depot, and (b) the two corresponding routes can be feasibly merged —i.e. the vehicle capacity is not exceeded.

The algorithm presented by Juan et al. [93], the *Generalized* or *Randomized Clarke and Wright Savings* (RCWS) heuristic, combines the CWS algorithm with the use of Monte Carlo Simulation (MCS) techniques [108]. This approach assigns a selection probability to each edge in the savings list. This probability should be coherent with the savings value associated with each edge, i.e. edges with larger savings will be more likely to be selected from the list than those with smaller savings. In addition, this approach adds this biased random behavior without introducing too many parameters in the algorithm.

In order to introduce such a probabilistic behavior, the RCWS uses different geometric statistical distributions during the solution construction process: every time a new edge must be selected from the list of available edges, a quasi-geometric distribution is randomly selected. This distribution is used to assign exponentially diminishing probabilities to each eligible edge, according to its position inside the sorted savings list. This way, edges with higher savings values are always more likely to be selected from the list, but the exact probabilities assigned are variable and they depend on the concrete distribution selected at each step. By iterating this methodology, the algorithm performs a random but efficient search process.

Formally, every time a new edge must be selected, we choose a real value α, $0 < \alpha < 1$, and then consider the following probability distribution (4.15) for selecting the k^{th} edge:

$$P(X = k) = \alpha \cdot (1 - \alpha)^{k-1} + \epsilon, \ \forall k = 1, 2, ..., s \tag{4.15}$$

where s is the current size of the list.

Algorithm 4.4 Randomized Clarke and Wright's Savings Heuristic

Initialization: calculate the savings S_{ij} for every pair $(i, j), \forall\ i, j \in I$. Create the savings list S, ranking the savings S_{ij} in descending order. Build the initial solution.

Repeat the following sequence until S is empty:

1. Pick the element S_{ij} from S at random:
   (a) Choose an α-value in the interval $[0.25, 0.35]$
   (b) Assign the corresponding probability to each element in S
   (c) Pick the element S_{ij} from S according to its probability.
2. Include the edge e_{ij} in a route if capacity constraints are not violated, *and* if:
   (a) Neither i nor j have already been assigned to a route $\rightarrow$ Initiate a new route including both i and j
   (b) One of the two customers has already been included in an existing route and that point is not interior to that route $\rightarrow$ Add the edge e_{ij} to that same route
   (c) Both i and j have already been included in two different existing routes and neither point is interior to its route $\rightarrow$ Merge both routes by adding edge e_{ij}.
3. Remove the element S_{ij} from S

The geometric distribution assigns a positive probability to every value in the interval $[1, +\infty)$. Since the algorithm always works with a finite savings list, the error term ϵ (4.16) is introduced in the geometric distribution to obtain the quasi-geometric one (4.15).

$$\epsilon = \sum_{k=s+1}^{+\infty} \alpha \cdot (1-\alpha)^{k-1} = 1 - \sum_{k=1}^{s} \alpha \cdot (1-\alpha)^{k-1} \tag{4.16}$$

As the process evolves, the savings list size s diminishes as new edges are extracted from it. If the size of the savings list is large enough, the term ϵ is

close to zero and the probability distribution (4.15) behaves like a geometric distribution. Thus, the parameter α can be interpreted as the probability of selecting the edge with the highest savings value at the current step of the construction process. Choosing a relatively low α-value implies considering a large number of edges from the savings list as potentially eligible. On the contrary, choosing a relatively high α-value implies reducing the list of potential eligible edges significantly. Once a value for α is chosen, this same value can be used for all future steps. However, Juan et al. [93] recommend to consider this α-value as a random variable to avoid minor fine-tuning processes. In this thesis, the value for α is chosen according to a uniform distribution in the interval $[0.25, 0.35]$ at each edge-selection step, as Juan et al. [93] suggest.

The RCWS algorithm, presented in Algorithm 4.4, is able to provide a random feasible solution at the end of each constructive process. Moreover, it outperforms the results obtained by means of the classic deterministic CWS, getting quasi-optimal solutions in very low times. These characteristics turn the RCWS into a good alternative for generating an initial solution for any local search process, such as VNS. In addition, its randomized behavior permits obtaining a set of quasi-optimal solutions with different characteristics, which is a desirable feature for feeding a multi-start schema like the one proposed in this section. For these reasons, we have chosen the RCWS method to generate the initial solutions required in our Multi-Start VND approach.

4.3.2 Parallel Multi-Start Variable Neighborhood Descent

A general VND, as explained in section 3.3.1, has been implemented embedding CP and LR methods. The VND method starts from an initial solution x', obtained by means of the RCWS heuristic, which is afterwards improved by a local search process. Figure 4.3 shows the flow chart corresponding to this hybrid VND approach.

In the proposed approach, outlined in Algorithm 4.5, four moves, described in section 4.2.2, are selected to be used in local search neighborhoods: *relocate*, *swapping*, *chain*, and *ejection chain*.

In the exploration of each neighborhood N_k, starting from the solution x', the k^{th} move is applied and the new solution's feasibility is checked using CP. Whenever it is proved feasible, LR is used to recalculate only modified routes. As mentioned for the hybrid VNS methodology, this approach permits to consider only two routes per solution, reducing the computation time when

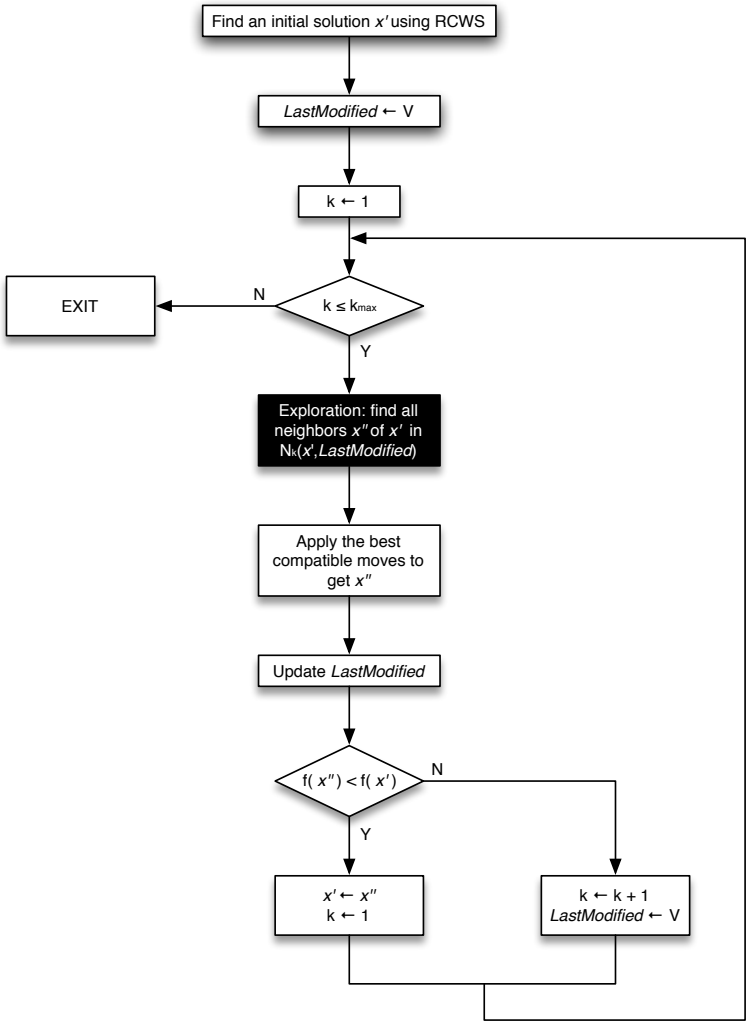

Figure 4.3: Flow chart of the hybrid VND approach. The process that uses the described CP and LR methods is highlighted.

compared to other routing post-optimization methods [155]. Improvements are stored in a sorted list until no more feasible solutions are left in the k^{th} neighborhood. Then, all those which are independent, i.e. affect different route pairs, are applied in descending order on x' to get a better solution, x''. This way, solution improvement is faster than applying a single change at each iteration.

Algorithm 4.5 Hybrid VND algorithm with improved local search efficiency
Initialization:
- Select the set of neighborhood structures N_k, for $k = 1, ..., k_{max}$, that will be used in the local search;
- Let x' be the initial solution, obtained by means of the RCWS algorithm;
- Initialize the set *LastModified* $\leftarrow V$;
- Choose a stopping condition;
- Set $k \leftarrow 1$.

Repeat the following steps until $k = k_{max}$:

1. Exploration of neighborhood:
   - Find all neighbors x'' of x' ($x'' \in N_k(x'$, *LastModified*$)$);
   - Check feasibility of capacity constraints using CP;
   - Calculate the cost of modified routes using LR;
   - If the solution x'' is better than x' ($f(x'') < f(x')$), include it in a list of improving changes.

2. Choose the best compatible neighbors:
   - Set *LastModified* $\leftarrow \emptyset$;
   - Sort the list of improving changes;
   - Apply the first improving change;
   - Add in descending order the next compatible improvements;
   - Add the modified routes identifiers to *LastModified*.

3. If the list is empty, set $k \leftarrow k + 1$ and *LastModified* $\leftarrow V$; otherwise, set $x' \leftarrow x''$ and $k \leftarrow 1$.

After the first exhaustive exploration of each neighborhood, only those changes affecting routes modified by previous movements are explored in order to reduce the computation time. The modified routes are stored in the set $LastModified$.

If the obtained neighbor x'' is better than the incumbent ($f(x'') < f(x')$), the current solution x' is updated and neighborhoods' exploration is restarted. Otherwise, the algorithm keeps x' as the best solution found so far and continues exploring the next neighborhood. When the VND process reaches a local optimum, no solution improvement may be found according to defined neighborhoods, and x' is returned as the best solution found.

Algorithm 4.6 Parallel Multi-Start VND algorithm

Initialization:
- Let x be the best solution;
- Create a thread pool with N_{total} threads.

Repeat the following steps until N_{total} threads end or until t_{max} time is consumed:

1. Execute N_{max} simultaneous threads:
   (a) Generate an initial feasible solution x' using RCWS.
   (b) Improve x' to obtain x'' by using the hybrid VND.
   (c) If x'' is better than x ($f(x'') < f(x)$), set $x \leftarrow x''$.

As mentioned, the VND-based local search process requires some type of diversification in order to overcome local optimality. As more constraints are introduced in the problem, it usually becomes more efficient —in terms of computational time employed— to generate new feasible solutions from scratch than to apply complex shaking processes that might end in non-feasible solutions. This is especially certain if we consider that the Randomized version of the CWS introduced in section 4.3.1 is a really fast method for generating different feasible and good solutions that can serve as initial solutions in a multi-start approach.

Thus, a Multi-Start strategy provides an appropriate framework which achieves diversification by re-starting the search from a new solution once a region has been extensively explored. Moreover, this approach is likely to be parallelized as long as the best solution found so far is correctly updated.

A simplified schema of the Parallel Multi-Start strategy is presented in Algorithm 4.6 and depicted in Figure 4.4. The RCWS algorithm is used to find a good initial solution. Then, the VND method helps to reach a local minimum in the neighborhood of the solution. The Parallel Multi-Start VND generates N_{total} tasks within a thread pool. If a thread is not available for the task, they wait in a queue for an active task to end. The algorithm stops when all tasks have been completed, or the maximum execution time t_{max} is reached, whichever happens first. Each task executes two phases: a first one in which a new feasible solution is constructed, and a second one in which the

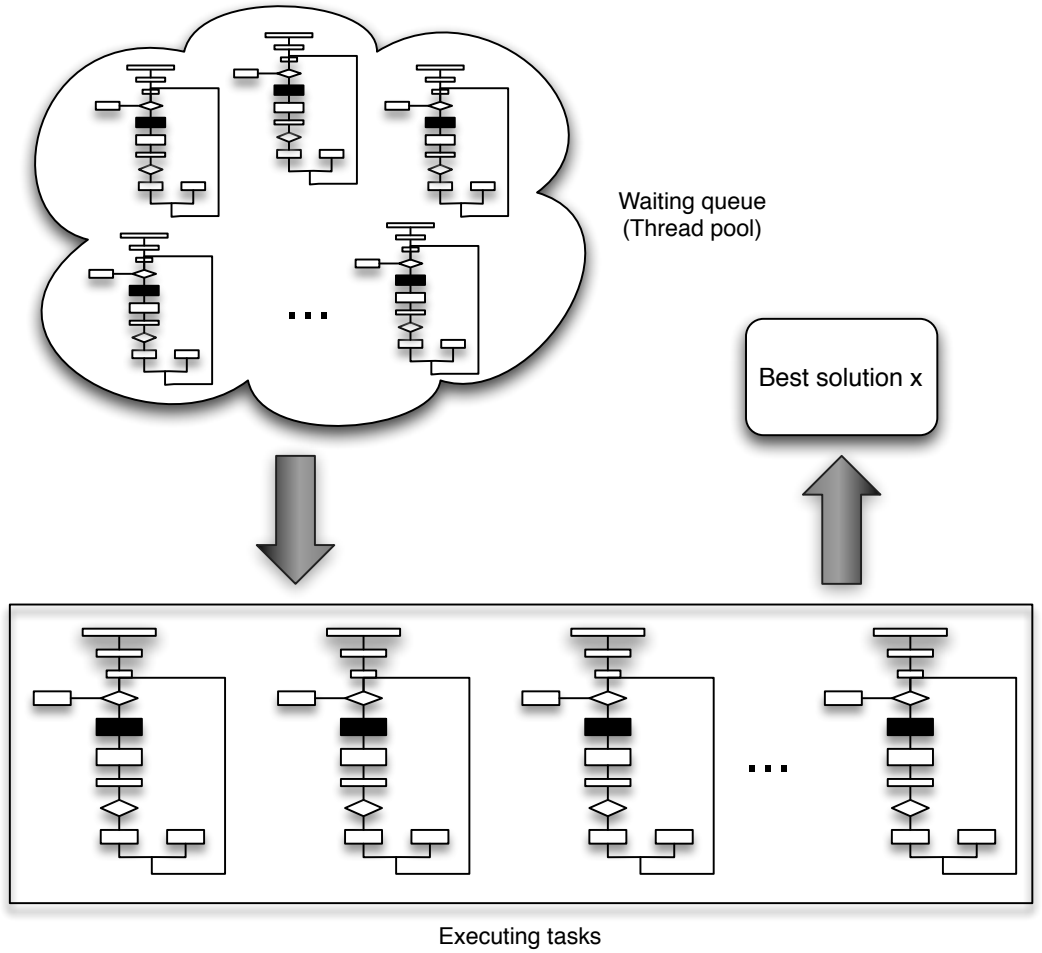

Figure 4.4: Parallel Multi-Start VND schema. A set of waiting tasks is generated within a thread pool. Anytime an executing task finds an improving solution, it updates the best solution found so far.

initial solution is improved through a VND local search process. Starting from a different initial solution ensures certain diversification, overcoming local optimality. Finally, anytime a task finds a feasible solution x'' better than the incumbent ($f(x'') < f(x)$), the current solution x is updated.

4.4 Constraint Programming Approach to the VRP

In the previous sections, we have presented two hybrid methodologies focused on solving the CVRP. Both approaches are based on the problem formulation described in section 4.1, which departs from its decomposition into two subproblems: resources allocation and routing solving. The former is modeled according to the CP paradigm, while the latter is formulated so it can be easily tackled from a LR perspective. As explained in sections 4.2 and 4.3, this decomposition provides an efficient approach to the problem.

Other VRP variants, such as the *VRP with Time Windows* (VRPTW) or the *Pick-Up and Delivery Problem with Time Windows* (PDTW) (see Chapter 2), define additional constraints that could lead to the need of a different decomposition strategy and complex problem formulations. In these cases, CP becomes a suitable approach to model these constraints in a compact and natural way. This paradigm permits defining a complete model of the problem at hand and, in addition, provides the flexibility needed to add new constraints in order to cope with richer instances. Moreover, CP allows modeling these constraints without altering the search procedures, intrinsically defined.

For these reasons, we present in this section a complete CP formulation for the VRP, based on the work by Kilby and Shaw [99]. This formulation has been implemented in order to overcome some limitations of the formulation described in section 4.1 and the hybrid methodologies introduced in the previous sections, such as instances dealing with an heterogeneous fleet or time windows constraints. It should be noticed that this CP formulation may be considered as a first step on the implementation of a VRP library based on the CP paradigm, able to cope with rich VRP variants and flexible enough to accept new constraints based on real applications. This library was first introduced in the article by Riera et al. [148].

4.4.1 Problem formulation

This section details a CP formulation of the VRP with quantity of goods and time constraints maintained along each route. As described in section 3.2, CP paradigm is based in three entities: (i) variables, (ii) their corresponding domains, and (iii) the constraints relating these variables. Therefore, the proposed CP formulation is defined according to these elements. This model can

be used as a basis for different VRP variants, and is specially suitable for the CVRP and the VRPTW. Nevertheless, one of the main virtues of CP is its versatility, so new constraints may be added without adjustment to this basic model.

Assuming the notation introduced in Chapter 2, we consider a set of n customers and a fleet of m vehicles. Then, the variables used in this formulation are:

- $C = c_1...c_n$ are the customers to serve;
- $M = m_1...m_m$ are the available vehicles;
- $Q_m = q_{m_1}...q_{m_m}$ are the vehicles capacities;
- $V = v_1...v_{n+2m}$ are the visits, with domain $V :: [1..m]$.

It should be noticed that there is one visit per customer and two special visits per vehicle: the starting and ending nodes for the vehicle, usually the depot. Thus, two subsets of V, F and L, are defined as the vehicles departure and arrival nodes:

- $F = \{n + 1...n + m\}$ is the set of first visits;
- $L = \{n + m + 1...n + 2m\}$ is the set of last visits.

In addition, two indexes are defined to denote the first and last visit for each vehicle:

- $f_k = n + k$ is the first visit of vehicle k;
- $l_k = n + m + k$ is the last visit of vehicle k.

To deal with routes, the predecessors set P is introduced to model the direct predecessor p_i of each visit i ($\forall i \in V - F$):

- $P = p_1...p_{n+m}$ is the predecessors set, with domain $P :: [1..n + m] :: (V - L)$;

Similarly, the set S is introduced to model the direct successor s_i of each visit i ($\forall i \in V - L$):

- $S = s_1 ... s_{n+m}$ is the successors set, with domain $S :: [1..n, n+m+1..n+2m] :: (V - F)$.

By convention, each first visit of a vehicle has as predecessor the vehicle's last visit ($\forall k \in M \; p_{f_k} = l_k$). Likewise, each last visit of a vehicle has as successor the vehicle's first visit ($\forall k \in M \; s_{l_k} = f_k$).

The use of the predecessor and successor sets creates a symmetric model. Although the solution space could be specified using only one of these sets without changing the set of solutions of the problem, defining both variables helps on propagating values more efficiently during the search. Thus, this redundant modeling permits making additional inferences which can significantly prune the search tree.

The predecessor and successor variables form a permutation of V and are therefore subject to the *difference constraints* (4.17) and (4.18).

$$p_i \neq p_j \quad \forall i, j \in V \wedge i < j \tag{4.17}$$

$$s_i \neq s_j \quad \forall i, j \in V \wedge i < j \tag{4.18}$$

These equations force predecessor and successor sets to contain no repetitions. Thus, one customer can have one and only one predecessor and successor. Notice that, when minimizing the number of vehicles is an objective, these constraints domains are restricted to $\forall i, j \in V_1 ... V_n \wedge i < j$ to permit the spare vehicles to remain at the depot.

The successor variables are kept consistent with the predecessor variables via the following *coherence constraints*:

$$s_{p_i} = i \quad \forall i \in V - F \tag{4.19}$$

$$p_{s_i} = i \quad \forall i \in V - L \tag{4.20}$$

Equations (4.19) and (4.20) connect the concepts successor and predecessor as follows: the former says that i is the successor of its predecessor, and the latter says that i is the predecessor of its successor.

To model multiple vehicles, we let the variables v_i from the set V to have a domain $[1..m]$ for each visit i, representing the vehicle which performs the visit. Along a route, all visits are performed by the same vehicle. This is maintained by the following *path constraints*:

$$v_i = v_{p_i} \quad \forall i \in V - F \tag{4.21}$$

$$v_i = v_{s_i} \quad \forall i \in V - L \tag{4.22}$$

Equations (4.21) and (4.22) are used to ensure that the vehicle assigned to i is the same as that assigned to its predecessor and successor. Naturally, for the first and last visits, the following constraints (4.23) are imposed:

$$v_{f_k} = v_{l_k} = k \quad \forall k \in M \tag{4.23}$$

Another three sets of variables are defined to represent visits' demands, cumulated capacities and the time for each visit:

- $R = r_1...r_n$ is the demands list, determining the amount of goods to be picked up ($r_i > 0$) or delivered ($r_i < 0$) at each visit i;
- $Q = q_1...q_n$ is the cumulated capacity list. After every visit i, $q_i \geq 0$ is a constrained variable representing the quantity of goods in the vehicle serving the visit;
- $T = t_1...t_n$ is the times list, indicating the time when the visit i is performed.

To maintain the load on the vehicles at each point in their route, the following *capacity constraints* are enforced:

$$q_i = q_{p_i} + r_i \quad \forall i \in V - F \tag{4.24}$$

$$q_i = q_{s_i} - r_{s_i} \quad \forall i \in V - L \tag{4.25}$$

Equations (4.24) and (4.25) count the goods picked up in a route, and the goods delivered in that route. Thus, the first constraint says the goods accumulated after visiting customer i is the addition of those accumulated in its predecessor plus those picked up in i ($r_i \neq 0$). The second constraint is similar but using the successor. Furthermore, the maximum capacity Q_k defined for a vehicle k must limit the accumulated capacities for every customer visited by that vehicle. This can be done either with an extra constraint or with domains bounding in case all vehicles have the same maximum capacity:

$$Q :: [0..Q_k] \longleftrightarrow 0 \leq q_i \leq Q_k \quad \forall i \in V, k \in M \tag{4.26}$$

A constraint (4.27) can be added to model that all vehicles begin their routes empty, as in the CVRP or the VRPTW. Leaving these quantities unconstrained would permit modeling different problems.

$$q_i = 0 \quad \forall i \in F \tag{4.27}$$

Time is maintained in the same manner as vehicle load except that waiting is normally allowed, and so the *time constraints* (4.28) and (4.29) maintain an inequality rather than an equality.

$$t_i \geq 0 \quad \forall i \in V$$

$$t_i \geq t_{p_i} + \tau_{p_i i} \; [+\tau_i] \quad \forall i \in V - F \tag{4.28}$$

$$t_i \leq t_{s_i} - \tau_{i s_i} \; [-\tau_i] \quad \forall i \in V - L \tag{4.29}$$

Equations (4.28) and (4.29) bound the accumulated time spent by a vehicle visiting the customer i. This time is, at least, the accumulated time in the predecessor of i, plus the travel time from the predecessor to i ($\tau_{p_i i}$). Equally, this time must be, at most, the accumulated time in the successor, minus the travel time from i to its successor ($\tau_{i s_i}$). This constraints may include the time spent at customer i (τ_i) if it is required by the problem instance.

Time windows on customers are specified by adding *time windows constraints* on the time variables:

$$a_i \leq t_i \leq b_i \quad \forall i \in V \tag{4.30}$$

Equation (4.30) states that customer i must be visited between times a and b. Usually, depot's time window is known as the *scheduling horizon*. Depending on the problem at hand, a customer i may be serviced out of its time window with a related penalty in the objective function (*soft time windows*). Minimizing these penalties becomes a secondary goal of the problem. In case this is not permitted, a vehicle is allowed to wait, but it cannot perform the visit i after its latest service time b_i (*hard time windows*). To model both situations, time windows constraints (4.30) are turned into (4.31) and (4.32).

$$\Delta a_i \geq a_i - t_i \quad \forall i \in V \tag{4.31}$$

$$\Delta b_i \geq t_i - b_i \quad \forall i \in V \tag{4.32}$$

The soft time windows are allowed by defining the Δa and Δb values as

$$\begin{array}{l} \Delta a_i \geq 0 \quad \forall i \in V \\ \Delta b_i \geq 0 \quad \forall i \in V \end{array} \tag{4.33}$$

while the hard time windows case is obtained by enforcing the Δb to be equal to zero:

$$\begin{array}{l} \Delta a_i \geq 0 \quad \forall i \in V \\ \Delta b_i = 0 \quad \forall i \in V \end{array} \tag{4.34}$$

Finally, the cost function of the problem is based on the total traveled distance. Considering δ_{ij} as the travel distance from visit i to visit j, the cost function is defined as follows:

$$d = \sum_{i \in V-F} \delta_{p_i i} \tag{4.35}$$

$$d = \sum_{i \in V-L} \delta_{i s_i} \tag{4.36}$$

It should be noticed the use of both the predecessor and successor variables to constrain the cost, which is usually more effective during search than using one single set.

In general, for CVRP problems the distance is considered to be equal to the traveling time, i.e. time variables may replace distances in the cost function. Therefore, equations (4.37) and (4.38) may be used to bind the cost function value.

$$d = \sum_{i \in V-F} \delta_{p_i i} \longleftrightarrow d = \sum_{i \in V-F} t_{p_i i} \tag{4.37}$$

$$d = \sum_{i \in V-L} \delta_{i s_i} \longleftrightarrow d = \sum_{i \in V-L} t_{i s_i} \tag{4.38}$$

For the VRPTW, in case soft time windows are allowed, the cost function may include penalty terms representing the time windows violations:

$$d = \sum_{i \in V-F} \delta_{p_i i} + \rho \sum_{i \in V} \Delta a_i + \omega \sum_{i \in V} \Delta b_i \tag{4.39}$$

$$d = \sum_{i \in V-L} \delta_{i s_i} + \rho \sum_{i \in V} \Delta a_i + \omega \sum_{i \in V} \Delta b_i \tag{4.40}$$

where ρ and ω are penalty weights that can be adjusted according to different modeling purposes.

Eventually, the cost function may also be defined according to the last visit performed by each vehicle, following the equation (4.41).

$$d = \sum_{k \in M} t_{l_k} \tag{4.41}$$

In order to improve the propagation during the search, equation (4.41) may be used simultaneously to the other defined cost functions, so tighter bounds are obtained on the total cost.

4.4.2 Constraint Programming-based search methods

Constraint Programming can offer many modeling advantages, as well as when solving routing problems, due to the increased pruning achieved through propagation. On the other hand, local search methods are an effective tool for solving such problems. However, some difficulties may arise when these two methodologies are used together. In general, a sequence of moves performed in local search may violate a basic operating principle in CP, the so-called *chronological backtracking*. Under chronological backtracking, decisions must be undone in the reverse of the order they were made. So in order to undo the last decision made during local search, we would have to undo all operations performed since that time, which would be unacceptable from a local search point of view.

In order to solve this problem, mainly two ways have been identified so far. The first is to allow a heuristic or metaheuristic method to control the search, as in the methods presented in sections 4.2 and 4.3. In this case, CP is used simply to check constraints fulfillment. The second way is to insulate the CP system from the changes made at the local search level, which is embedded within an *operator*. Many of these operators are based on serial insertion and block deletion, modifications well suited to use within a CP framework. One such insertion-based technique is *Large Neighborhood Search* (LNS) (see section 3.4), specifically developed by Shaw [161] to be used in a CP environment.

As mentioned in section 3.4, in LNS an initial solution is gradually improved by alternately destroying and repairing the solution. In this metaheuristic, the neighborhoods are implicitly defined by the *destroy* and *repair* operators. The destroy operator typically contains an element of stochasticity such that

different parts of the solution are destroyed in every invocation of the method. Nevertheless, deterministic destroy methods can also be implemented.

Searching a very large neighborhood should potentially lead to higher quality solutions than searching a small neighborhood. However, in practice, small neighborhoods can provide similar or superior solution quality if embedded in a metaheuristic framework. Hence, a natural idea is to gradually extend the size of the neighborhood, each time the search gets trapped in a local minimum. This leads to a nested structure that can be searched efficiently. In this sense, destroy operators can be defined in a way such that the degree of destruction is gradually increased, as Shaw [161] proposes. The degree of destruction is an important choice when defining a destroy method: if only a small part of the solution is destroyed it may be difficult to leave a valley in the search space; if a very large part of the solution is destroyed, the LNS heuristic may degrade into a multi-start process.

Choosing the repair method permits much more freedom when implementing a LNS heuristic. A first decision is whether the repair method should be exact or heuristic. The former, although slower, may lead to high quality solutions in few iterations. The latter may be more interesting from a diversification point of view.

As introduced in section 3.2, CP search methods are mainly based on assigning values to variables, in such a way that constraints are satisfied and other variables' domains are reduced to their compatible values through constraint propagation. Therefore, CP-based destroy and repair methods will *unassign* and *assign* values to variables, respectively, at different stages of the search. The definition of both concepts is straightforward.

Definition: An *unassignment* is a pair of variable x and value a (denoted $x \not\leftarrow a$) such that a has been ruled out as a possible value for x at the point in search under consideration. An *assignment* is a pair of variable x and value a (denoted $x \leftarrow a$) where variable x is set to a at the point in search under consideration.

From these definitions, it can be inferred that a solution is a complete assignment (or complete *labeling*) to the variables of the problem, in such a way that all constraints are satisfied at once.

A CP-based destroy operator unassigns some values from a solution, destroying it partially. For the VRP, it is useful to remove some visits from a

Algorithm 4.7 SPLIT LNS-based operator

Initialization:
- Select the set of k ($k = 1, ..., k_{max}$) areas that will be used for splitting;
- Let x be the initial solution;
- Set $k \leftarrow 1$.

Repeat the following steps until $k = k_{max}$:

1. Assign the partial labeling $x' = x \setminus \{x_k\}$
2. Repair the partial solution x' using branch-and-bound during t_{max} time to get x''
3. If the solution x'' is better than x ($f(x'') < f(x)$), set $x \leftarrow x''$ and $k \leftarrow 1$; otherwise, set $k \leftarrow k + 1$

solution, i.e. unassign all values from variables related to these visits. The solution can then be re-optimized by re-inserting these visits, using a repair method. One iteration of removal and re-insertion can be considered as the examination of a neighborhood move. If a re-insertion is found that results in a cost below that of the best solution found so far, this new solution is kept as the current one.

A critical choice on designing a destroy method is how to select the customers to be removed and re-inserted. A general strategy consists of choosing related visits, since removing visits whose re-insertion is independent of the others is less likely to lead to a solution improvement. Based on the observation that visits geographically close to one another are more related than remote ones, we define two strategies for choosing the customers to be removed from a solution.

The first strategy for choosing the visits to be removed is based on the geographical distribution of customers. It is implemented in the *SPLIT* operator outlined in Algorithm 4.7. The nodes scatterplot is divided into the four quadrants according to the Cartesian axes, and a different VRP is solved on each subspace. First, all customers laying in the first quadrant are selected for removal and their corresponding variables are unassigned. These visits are then re-inserted in order to reduce the solution cost. Figure 4.5 shows how the SPLIT operator performs the search. The four quadrants are visited in a

VND fashion: anytime an improving solution is found, the first quadrant is revisited and the process is repeated; otherwise, the algorithm switches to the next quadrant. The process stops when no improvements are found. It should be remarked that other geographical-based strategies may be implemented, like the ones proposed by Juan et al. [92].

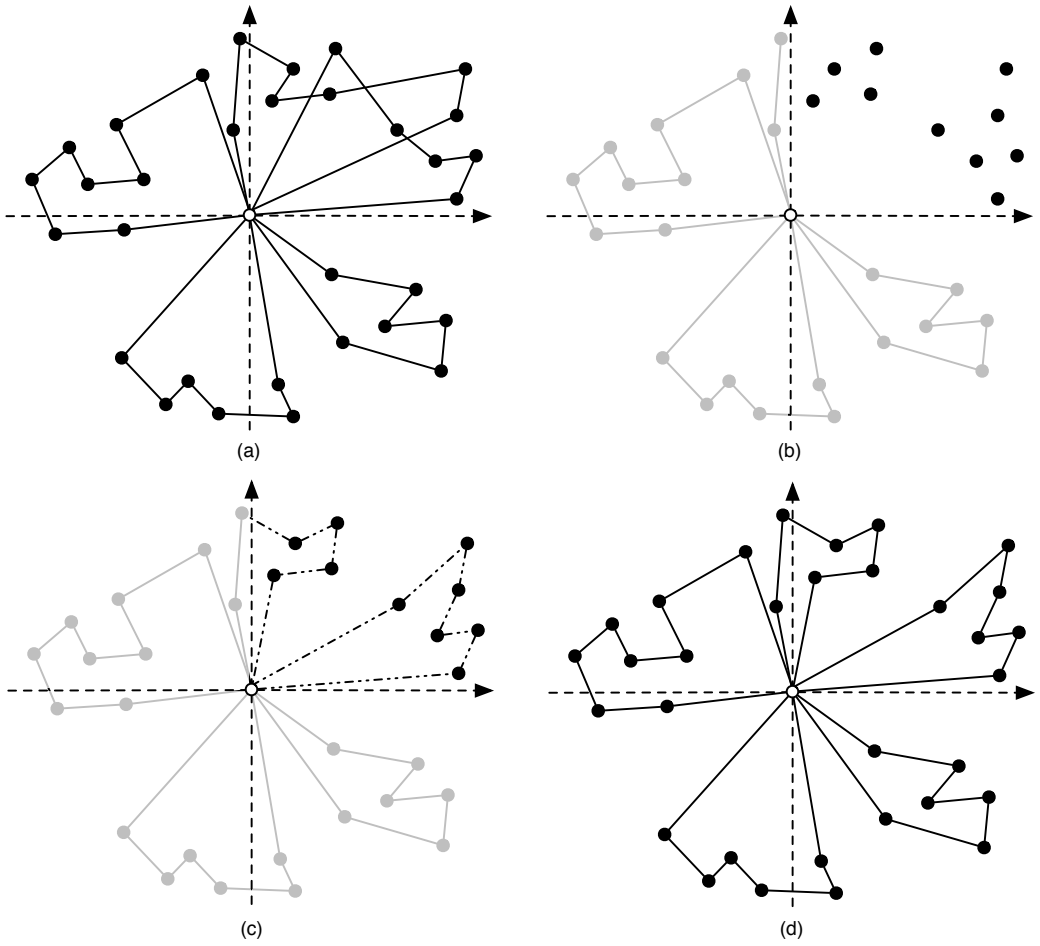

Figure 4.5: SPLIT operator. (a) Original solution; (b) customers from the quadrant to be explored are removed; (c) the repair method re-inserts the removed visits; (d) new solution.

Algorithm 4.8 RPOP LNS-based operator

Initialization:
- Select a maximum number of pivots k_{max} to be used;
- Select the number of customers l around each pivot to be removed;
- Let x be the initial solution;
- Set $k \leftarrow 1$.

Repeat the following steps until $k = k_{max}$:

1. Choose randomly k pivots from the solution $x \Rightarrow x_k$
2. Choose the l closest neighboring visits around each pivot $k \Rightarrow x_l = \bigcup_k x_{l_k}$
3. Assign the partial labeling $x' = x \setminus \{x_k \cup x_l\}$
4. Repair the partial solution x' using branch-and-bound during t_{max} time to get x''
5. If the solution x'' is better than x ($f(x'') < f(x)$), set $x \leftarrow x''$ and $k \leftarrow 1$; otherwise, set $k \leftarrow k + 1$

The second destroy method chooses the customers to be removed randomly. The steps of this operator, which we call *Random Pivot OPerator* (*RPOP*), are outlined in Algorithm 4.8. For efficiency reasons, we want to remove the smallest set of customers that could lead to an improvement on the cost when the visits are re-inserted. Initially, we choose one visit to be removed, which we call a *pivot*. As this only could lead the search to the previous solution, a set of neighboring customers are also removed, creating a hole around the pivot. It is important to notice that the set of removed customers may or may not belong to the same route, since they have been chosen according to their proximity. This characteristic permits swapping visits between different routes that could decrease the solution cost. Figure 4.6 depicts the RPOP behavior. During search, if re-inserting these removed customers has not resulted in an improvement in the cost, the number of pivots is increased by one at next iteration. This way, the degree of destruction is gradually increased. An upper limit on the number of pivots is to be defined to avoid generating neighborhoods too large to be explored. A VND exploration strategy is again adopted: anytime an improvement is found, search is redirected to the exploration of the smaller neighborhood, i.e. starts over by selecting one single pivot.

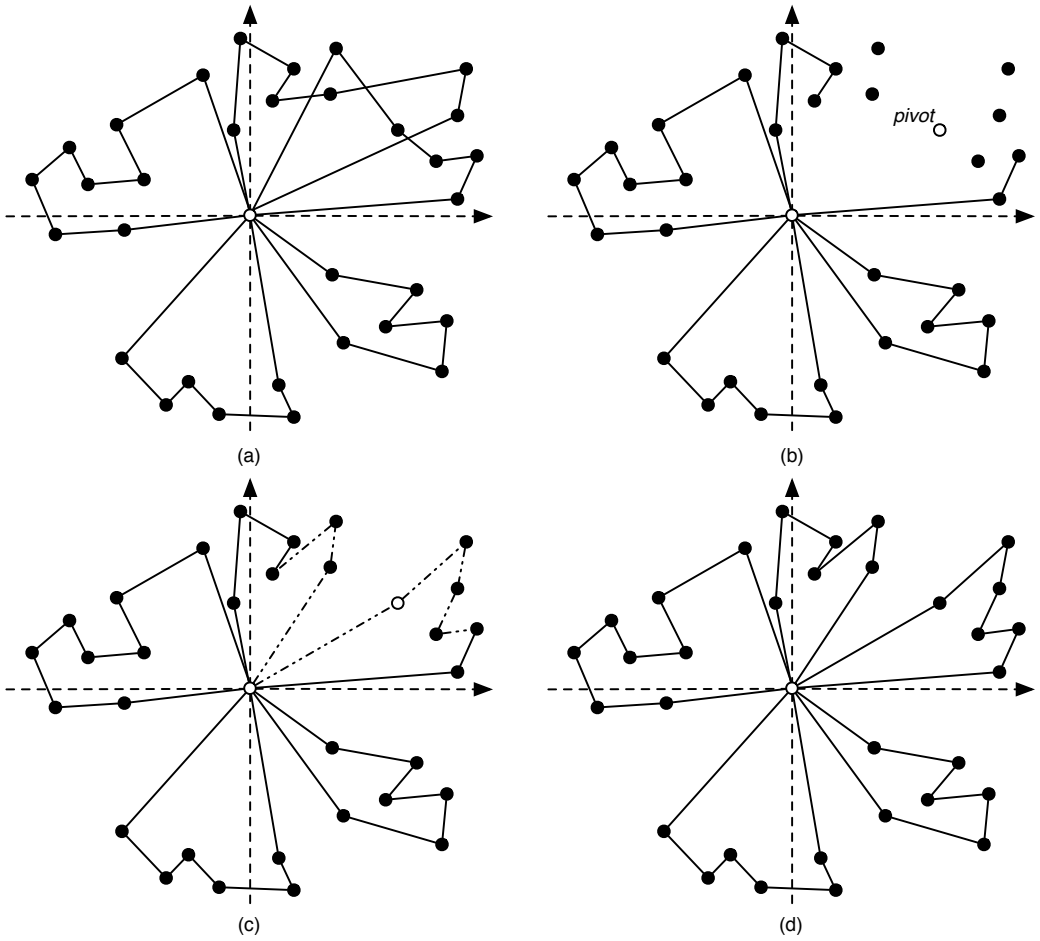

Figure 4.6: RPOP operator. (a) Original solution; (b) pivot customers and their closest neighbors are removed; (c) the repair method re-inserts the removed visits; (d) new solution.

We have chosen a CP-based exact repair method so it takes advantage of improved propagation as it departs from a partial solution which helps on pruning the search tree. Concretely, the repair method consists of a *branch-and-bound* method with constraint propagation, with a limited execution time. Thus, although slower than heuristic methods, we benefit from high quality solutions while not being penalized with an excessive computational time. During search, the upper bound is set to the cost of the best solution found so far. Insertion positions for visits can be forbidden if they take the lower bound on the cost over the defined upper bound. We form the lower bound as the current

Algorithm 4.9 VND-LNS algorithm

Initialization:
- Select the set of operators O_k, for $k = 1, ..., k_{max}$, that will be used;
- Find an initial solution x and improve it by means of a branch-and-bound method for a limited time;
- Set $k \leftarrow 1$.

Repeat the following steps until $k = k_{max}$:

1. Apply the O_k operator to obtain the solution x'
2. If the solution x' is better than x ($f(x') < f(x)$), set $x \leftarrow x'$ and $k \leftarrow 1$; otherwise, set $k \leftarrow k + 1$

cost of the partial solution constructed during search, i.e. the lower bound is not computed separately by any other method. This makes the repair method faster, but the search tree is larger than it would be if an accurate lower bound was calculated. In its simplest form, the branch-and-bound search explores the whole tree for the re-insertion of all visits at minimum cost. However, variable and value selection heuristics may be defined to improve the search efficiency (see section 4.4.3).

The SPLIT and RPOP operators are used in a local descent strategy, as they never allow the objective function to increase. As these strategies are likely to get trapped in a local minimum, we embed them within a VND algorithm, as explained in section 3.3.1. The steps of this algorithm are presented in Algorithm 4.9. Combining LNS with a VND metaheuristic permits exploring the search space in a systematic fashion. Whenever the algorithm reaches a local minimum for any of the operators, it changes to a different, usually smaller neighborhood defined by a different operator. Thus, the VND oscillates between the two operators, hoping that changes in neighborhood structure will permit escaping from most local minima.

Although we are using an exact method such as branch-and-bound to repair a partially destroyed solution, we are limiting its execution time. For this reason, further improvements may be found during subsequent executions. In addition, every time the algorithm finds a better solution, the upper bound is updated and the search tree is pruned in consequence. Therefore, the branch-and-bound method can explore more efficiently the search space in order to

find a better solution in the destroyed region. At this stage, the combination of LNS and VND play a key role, so different neighborhoods can be explored and revisited iteratively with improved upper bounds.

As for the destroy methods, the SPLIT operator deserves special attention. If all routes are completely separated into quadrants and the customers are re-inserted optimally, re-optimizing these areas would imply a waste of computational time. However, this is very unlikely and usually routes fall over different quadrants, so any change in one of them may affect some others. For example, relocating one customer in a different route affects capacity and time constraints of the original route and the destination one. These changes may turn useful re-optimizing the quadrants affected by these two routes. Again, using VND and LNS strategies ensure revisiting neighborhoods anytime a change that can modify them is found during search.

It should be remarked that the addition of time constraints to the VRP makes the problem much more difficult. In the VRPTW, even finding a feasible solution is NP-hard [158]. For this reason, the application of CP exact methods to explore the search space is limited and the definition of heuristic incomplete methods becomes mandatory. Even though, tackling the VRPTW by using pure CP techniques can take high computational times, even for finding an initial feasible solution or solving the associated subproblems [132]. The approach presented in this section may help on solving the VRPTW, but it requires further improvements to be competitive with other state-of-art methods. Some of these improvements are introduced in the next section.

Eventually, this approach may be used to solve some small CVRP instances. The methods introduced in this section are not able to cope with most of the medium and large VRPTW instances. However, it may solve most instances if soft time windows are considered. Thus, this methodology may be useful to find appropriate lower bounds to the VRPTW instances, since the soft time windows variant is a relaxed problem of its original VRPTW instance. In any case, as mentioned, none of these applications are competitive with other approaches and further efforts need to be addressed in this direction.

The aim of this approach was to establish an initial framework as a first approach to more complex, rich VRP. The proposed formulation can be used to check feasibility, as well as the VND-LNS approach can be used to solve or re-optimize small parts of a problem. Although other faster heuristic methods could be used for the latter, they do not provide the flexibility obtained by using CP. Adding a new constraint in CP is a modeling issue, while adding the same constraint in other methods may imply important changes in algorithm's

structure and performance. Hybrid methodologies that use CP to validate solutions, such as the ones presented in sections 4.2 and 4.3, would suffer from the introduction of complex and very tight constraints because solutions acceptable by the CP solver would become more rare. As for this approach, it would gain in performance, since propagation would be used more extensively to prune the search space.

4.4.3 Variable and value ordering heuristics and improvement methods

As mentioned, the algorithm's performance may be improved by introducing several modifications. First hand, although the formulation presented in section 4.4.1 exactly describe the set of feasible solutions, additional constraints may improve the logical reasoning and reduce the solution space. On the other hand, modifying the order in which variables are assigned and their values can drastically alter the shape of the search tree.

Time windows defined for each customer may be preprocessed to add new constraints to the problem. These additional constraints may help on pruning the search space by reducing variables' domains. Depending on the problem instance, some time windows may be overlapped, so a vehicle cannot serve those customers without violating their respective working times. Time windows preprocessing is used to detect these situations and add constraints defining which customers are incompatible to be in the same route.

Two customers are determined to be incompatible if a vehicle cannot perform both visits within their respective time windows. Considering a vehicle arriving to customer $i \in V$ and starting the service at a time corresponding to the lower bound of its time window (a_i), it cannot be visited previously in the same route than a second customer $j \in V$ if the cost to reach j from i exceeds the latter's time window upper bound (b_j).

$$a_i + \tau_{ij} \; [+\tau_i] > b_j \tag{4.42}$$

If the expression (4.42) still holds swapping indexes i and j, then customers $i, j \in V$ cannot lead to a feasible solution fulfilling time windows when they are included in the same route. Therefore, an additional constraint is defined so they cannot be assigned to the same route:

$$a_i + \tau_{ij}\ [+\tau_i] > b_j\ \wedge\ a_j + \tau_{ij}\ [+\tau_j] > b_i \Rightarrow v_i \neq v_j\ \ \forall i, j \in V \tag{4.43}$$

Constraint (4.43) is defined for each pair of incompatible visits. Thus, time windows preprocessing helps to avoid the exploration of some unfeasible solutions, since it provides a more constrained problem.

Additional constraints may be inferred from the time windows preprocessing. Although the path constraints (4.21) and (4.22) propagate on P and S variables when constraint (4.43) is imposed, explicit constraints may be defined on these variables. In addition, it should be noticed that two customers may be feasibly included in the same route, but with some limitations on the precedence order. For example, it may happen that customers i and j fulfill the expression (4.42), but the inverse swapping indexes is not satisfied. In this case, in order to fulfill the time windows constraints, i cannot be visited immediately before j. In consequence, constraints (4.44) and (4.45) are defined.

$$a_i + \tau_{ij}\ [+\tau_i] > b_j \Rightarrow p_j \neq i \wedge s_i \neq j\ \ \forall i \in V - L, j \in V - F \tag{4.44}$$

$$a_j + \tau_{ij}\ [+\tau_j] > b_i \Rightarrow p_i \neq j \wedge s_j \neq i\ \ \forall i \in V - F, j \in V - L \tag{4.45}$$

By definition, if constraint (4.43) is imposed, both constraints (4.44) and (4.45) are automatically enforced.

Though the imposed constraints may contribute to significantly reduce the size of the search space to be explored, search strategies still play a major role. In this sense, the order in which variables are fixed and their values tried have a large influence on search algorithms' performance.

Variable ordering is by far the most important since it can drastically alter the shape of the search tree. Value ordering determines which parts of the tree are first explored and how soon a very good solution, or even an optimal one, may be found. An appropriate combination of both strategies may lead the search efficiently, while a wrong choice could force the search method to explore infertile regions of the search space.

Dynamic variable ordering strategies are generally recognized to be more effective rather than fixing an ordering *a priori*. Thus, the most promising variable is chosen according to the information available at that point of search.

Within this schemes, the *first-fail* heuristic emerges as a good strategy. It states that the search should provoke dead-ends as early as possible to reduce the search effort. For this reason, the variable with the smallest domain size is favored to be chosen. A first-fail variant, the *most-constrained* heuristic, works in a similar way, but in case several variables have the same domain size, the one with the largest number of attached constraints is selected first. Finally, variable ordering heuristics may be based on the values associated to each domain. For example, a *smallest* heuristic will pick the variable with the smallest value in its domain.

The value ordering strategies determine how the values in a domain are explored. A common strategy for most optimization problems is to explore a domain so values are tried in increasing order. However, multiple strategies have been defined over the years, such as starting from the middle value of the domain, successively splitting and removing intervals in the domain, or trying values in a random order. Most of these variable and value ordering heuristics are implemented in most CP platforms.

Following these guidelines, we define two strategies for variable and value ordering for the VRPTW. These strategies take advantage of the problem's structure.

The first strategy is based on a *Nearest Neighbor* heuristic. Since close visits are more likely to be in the same route in the optimal solution, variables to be assigned are chosen according to this criterion. Before starting the search, the distances matrix is used to determine the distances from each visit to all its feasible successors. By associating these values to each variable s_i ($\forall i \in V - L$) and using the *smallest* heuristic as the variable ordering strategy, the order in which the variables s_i are labeled is based on the distance to their closest neighbor. Moreover, if the values are tried in increasing order, anytime a successor is found unfeasible, the second closest neighbor will be tried, and so on.

The second strategy is defined according to the start time on each visit time window (a_i). In a similar fashion to the strategy presented above, the earlier starting time for each feasible successor is associated to each variable s_i ($\forall i \in V - L$), and the search is performed using the *smallest* heuristic as the variable ordering strategy. Thus, variables corresponding to visits with the earliest starting time window are assigned first. Again, the values are tried in increasing order, since minimizing the traveling time is a goal of the problem.

Though this time criterion is used for ordering the variables labeling, it

should be remarked that time variables are not assigned during search. For efficiency reasons, only lower and upper bounds for each time variable are maintained and updated through propagation. Time constraints (4.28) and (4.29) are used to reduce the domains of the time variables anytime a predecessor or successor variable is assigned. Keeping the time variables bounded to their feasible domains permits pruning the search tree, while reducing the computational burden required for assigning each time variable individually.

The described strategies have been included in the CP-based search method presented in section 4.4.2. Although these heuristics may help on guiding the search, this methodology still requires further improvements to be competitive with state-of-art methods. Even applying such techniques, the described methodology may not be able to solve the VRPTW in a reasonable time. Similarly, Pesant et al. [132] propose a model alike with improved propagation strategies to solve the TSPTW, but they are not able to solve many instances in reasonable times. The computational complexity associated to the VRPTW makes it a hard problem to tackle from a pure CP perspective. In this sense, the methodology and strategies presented in this section are an initial approach which keeps open different lines for future research. Some of them are outlined in section 6.2.

Chapter 5

Application and Case Studies

In this chapter, the computational results obtained for the different methodologies described in Chapter 4 are presented. First, we provide some results for the two hybrid methodologies —the hybrid VNS approach and the Multi-Start VND method—, focused on the CVRP. These results are analyzed separately, and subsequently compared with other state-of-art heuristics in section 5.1.3. Finally, results for the CP-based approach are reviewed in the last section. This methodology has been used to tackle the CVRP and the VRPTW, yielding interesting results as a first approach to these problems.

5.1 Hybrid approaches to the CVRP

A total of 97 classical CVRP benchmark instances have been used to test the efficiency of the presented approaches. They have been obtained from *branchandcut.org* [1], a reference site with a large number of benchmark sets for different combinatorial problems, yet not updated results. Best known solutions for problems not solved to optimality have been updated with recent references in order to provide a thorough comparative with results obtained by using the methodologies presented in this thesis. Instances were selected according to the distance type used in their definition. Only those instances whose distance is defined as Euclidean or Geographic have been selected, in order to ensure triangular inequality's fulfillment. Therefore, all problems from benchmark sets A, B, F, G, M, and P have been included. In addition, those instances from the set E accomplishing the mentioned criterion are also

considered, as well as three TSPLib [146] converted problems (*att-n48-k4* and both *ulysses* instances).

For further references on the size of each instance, all problems are labeled according to the following notation: the first letter denotes the benchmark set which the instance belongs to; the number after the n is the total number of customers, including one depot, i.e. it indicates the size of the problem; the number after the k is the maximum number of available vehicles to solve the instance. For example, the instance *A-n32-k5* belongs to the set A, 31 customers and one depot are defined, and a feasible solution may use at most 5 vehicles.

5.1.1 General Variable Neighborhood Search

The methodology described in section 4.2 has been implemented in Java and linked to the open-source CP software system ECLiPSe 6.0 [9]. All tests have been performed on a non-dedicated server with an Intel Xeon Quad-Core i5 processor at 2.66GHz and 16GB RAM. In general, five to seven processes were launched in parallel to solve different problems, while external applications were active at the same time. Nevertheless, we strongly believe that these applications had a poor influence on the results presented in this section, since their use of CPU and memory resources was low and not continuous in time. In any case, so obtained CPU times are to be considered as approximated.

The hybrid VNS approach has been used to solve the 97 considered CVRP benchmark instances. In this case, distances have been rounded to integers, according to the specification included in the TSPLib [146]. This approach allows comparing the obtained results with those published in a wide range of references working over the same benchmark sets. However, this methodology is not restricted to work with integer distances, so it can be used without adding any modification to solve the same instances considering real costs. Although a good number of publications adopt this realistic approach, results presented in this section are aimed to be compared with best known integer solutions. In order to compare the obtained solutions with best known realistic ones, the algorithm should be run using real costs matrices, since rounding distances prior to solving make both instances different, and so are their corresponding solutions. This realistic approach is adopted for the Multi-Start VND approach, whose results are presented in section 5.1.2.

In all tests, *swapping* (see section 4.2.2 for movements description) has been set as the initial move both for shaking ($k = 1$) and local search ($l = 1$)

Table 5.1: Summary of results obtained by means of the hybrid VNS approach with a swapping/relocate configuration.

Class	Problems	Opt.	No opt.	Not solved	% Dev.	% Max	% Min
A	27	14 (51.85%)	13 (48.15%)	0 (0.00%)	0.65	2.14	0.14
B	23	12 (52.17%)	11 (47.83%)	0 (0.00%)	1.79	4.28	0.13
E	11	5 (45.45%)	6 (54.55%)	0 (0.00%)	0.68	1.59	0.41
F	3	1 (33.33%)	2 (66.67%)	0 (0.00%)	4.22	4.22	4.22
G	1	0 (0.00%)	1 (100.00%)	0 (0.00%)	0.44	0.44	0.44
M	5	1 (20.00%)	4 (80.00%)	0 (0.00%)	2.44	4.35	0.69
P	24	14 (58.33%)	8 (33.33%)	2 (8.33%)	0.88	3.30	0.15
TSPLib	3	1 (33.33%)	2 (66.67%)	0 (0.00%)	2.28	3.23	1.33
Totals	*97*	*48 (49.48%)*	*47 (48.45%)*	*2 (2.06%)*	*1.67*	*2.94*	*0.94*

processes. *Relocate*, *chain*, and *ejection chain* are used next whenever the previous solution is not improved. In the implemented approach, the relative percentage of customers modified by the ejection chain movement has been set to 40 %. In general, the number of customers assigned to each route is low for most CVRP instances, so a relatively high percentage of removed customers at the end of each route is required to avoid exploring a subset of the chain neighborhood. A summary of the obtained results with this neighborhood set is presented in Table 5.1. In all cases, the stopping criterion has been set to a maximum of 40 iterations.

Table 5.1 presents the number of problems successfully solved to optimality by using the hybrid VNS methodology, as well as the number of problems whose optimal value was not reached and those which could not be solved. Table 5.1 also shows the average (% Dev.), maximum (% Max), and minimum (% Min) deviation from the best known value for those problems that could not be solved to optimality. A low deviation is observed for most problem sets, comparable to the results obtained by means of other metaheuristics.

Another test has been done exchanging the swapping and relocate priorities, getting a similar performance. For this reason, only results from the first configuration are shown. Nevertheless, it is remarkable that, in general, the swapping/relocate configuration has a better performance when applied to class M problems, while relocate/swapping behaves better on class B instances. Probably, performance differences rely on customers distribution and the tightness of each problem, i.e. the ratio demand/capacity of each problem. In class B problems, the customers are clustered, so higher improvements are expected when applying the relocate movement first. On the other hand, class M instances are very tight problems. Thus, relocating customers may be difficult due to capacity constraints. In this case, swapping customers might be

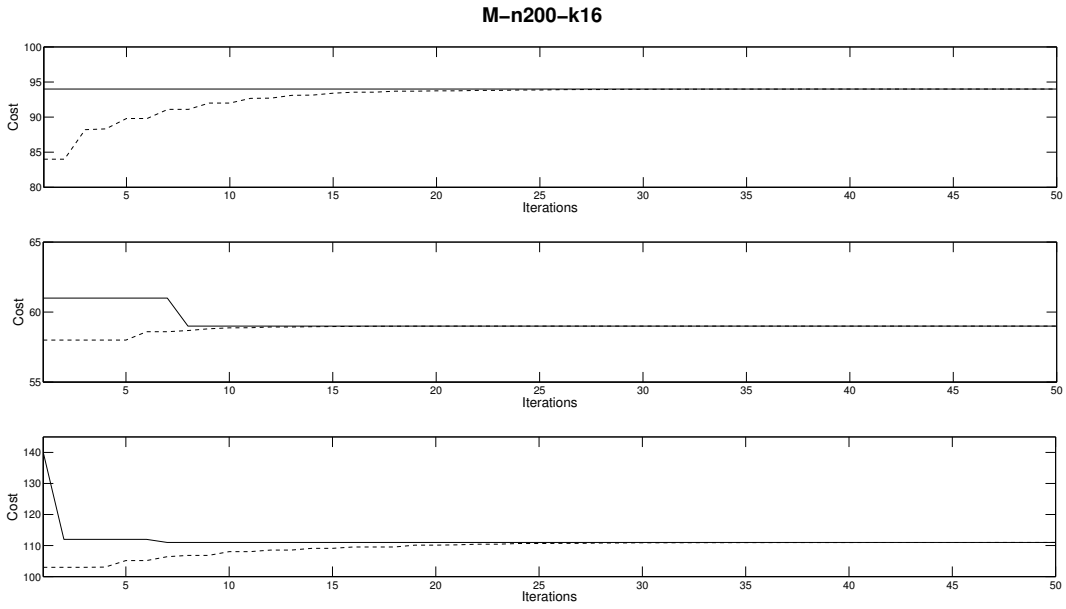

Figure 5.1: Convergence of the lower (dashed line) and the upper (solid line) bounds in three routes from problem *M-n200-k16*. Although LR maximum number of iterations is set to 300, it usually converges in less than 50 iterations for most problems.

a more efficient strategy to reduce the total cost.

As mentioned in section 4.2.3, the initial solution is obtained by solving separately capacity and routing problems. This approach is able to provide a low-quality quick solution, since both subproblems are easily solved but their variables are unlinked. However, this solution is highly improved at the first iteration. As an example, this approach may provide an initial feasible solution for larger problems, such as the *M-n200-k16*, in less than 6 seconds. After the first iteration, the cost of the current solution is usually close to the final result.

Furthermore, the use of LR ensures the partial optimality of all solutions from the routing perspective. The reason is that the tailored LR-based approach presented in section 4.2.1 can optimally solve all TSP instances resulting from the allocation of customers. As can be seen in Figure 5.1, the lower and upper bounds converge rapidly. For all problems, their gap is always located between 0 and 10^{-10}, guaranteeing so the solution optimality. Moreover,

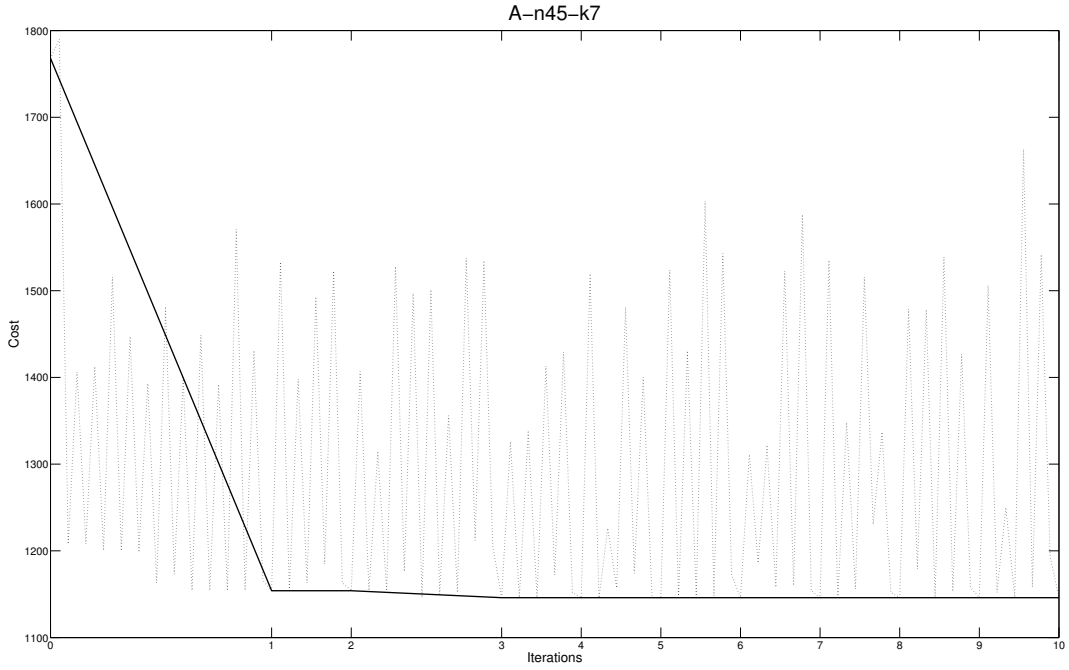

Figure 5.2: An example of the local search process on problem *A-n45-k7* for the first 10 iterations. The solid line corresponds to the final solution evolution, while the dotted line shows the shaking process and the local search behavior.

LR solves all routes in negligible times, due to the number of associated customers is always low. Thus, LR has demonstrated to be an efficient alternative for intra-route optimization processes.

The detailed behavior of the hybrid VNS method can be observed in Figures 5.2 and 5.3. In Figure 5.2, the solid line corresponds to the final solution evolution, while the dotted line shows algorithm's behavior at each iteration. Every shaking process draws a peak, while the local search process leads the algorithm to a solution with a lower cost. Figure 5.3 shows this behavior in more detail. It can be observed that, after some iterations, the local search process converges more often to the best solution found so far. As the algorithm evolves, so does solution's structure quality, getting closer to the optimum. Therefore, a more thorough shaking process would be needed to get the solution far enough so different regions from the solution space were explored. Nevertheless, it could cause the algorithm to diverge or to degrade into a multi-start process. As an example, when chain and ejection chain movements are

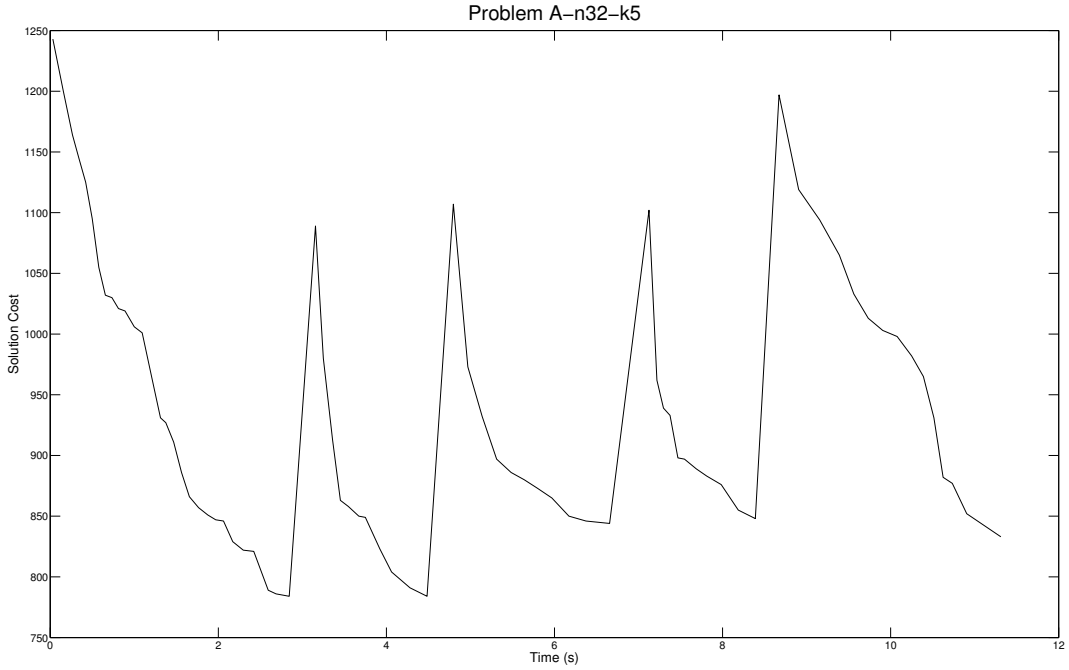

Figure 5.3: Detailed behavior of the hybrid VNS approach for the problem *A-n32-k5*. It can be observed how the shaking and local search processes affect the solution cost.

applied in the shaking process, the local search process either converges slowly or may not find the current best solution. Since both moves modify larger section than relocate or swapping, the exploration might be leaded to regions far from the current solution.

The hybrid VNS methodology performs similarly both for small and large instances. Thus, its applicability is not restricted. It is remarkable that the algorithm eventually reaches the optimal solution for smaller problems (50 customers or less), but it stops near the optimum for larger instances. Tables 5.2 and 5.3 show representative results obtained for classes A and P. These results are similar to those obtained for the remaining classes. These tables compare the obtained results to the best known integer solutions (columns BKS and $\#OV$). They show the initial (IS) and the final solution (OBS), computational times spent on calculating them, and the corresponding gap of the final solution. The computational time needed to reach a solution with a gap lower than 2 %, selected as a good quality solution threshold, has also been included in all tables. The last column presents the iteration at which the best-

Table 5.2: Results obtained for class A problems.

Problem	**BKS**	**IS**	CPU_{IS} **(s)**	**OBS**	CPU_{OBS} **(s)**	**Gap BKS-OBS (%)**	$CPU_{\leq 2\%}$ **(s)**	**#V/ #OV**	**Opt. It.**
A-n32-k5	784	1243	0.256	784	96.62	-	96.62	5 / 5	2
A-n33-k5	661	1185	0.018	661	42.67	-	28.05	5 / 5	2
A-n33-k6	742	1289	0.013	742	34.86	-	34.86	6 / 6	32
A-n34-k5	778	1259	0.014	778	27.88	-	27.88	5 / 5	1
A-n36-k5	799	1207	0.024	799	524.33	-	148.16	5 / 5	22
A-n37-k5	669	960	0.012	669	65.76	-	19.82	5 / 5	2
A-n37-k6	949	1393	0.020	949	59.67	-	27.10	6 / 6	14
A-n38-k5	730	1240	0.017	731	69.02	0.14	23.07	5 / 5	-
A-n39-k5	822	1291	0.010	822	277.41	-	39.54	5 / 5	8
A-n39-k6	831	1523	0.014	833	554.50	0.24	84.94	6 / 6	-
A-n44-k6	937	1547	0.020	942	149.39	0.53	60.68	6 / 6	-
A-n45-k6	944	1826	0.022	950	141.67	0.64	64.16	6 / 6	-
A-n45-k7	1146	1768	0.022	1146	207.30	-	77.42	7 / 7	3
A-n46-k7	914	1711	0.024	914	265.87	-	94.97	7 / 7	2
A-n48-k7	1073	1840	0.025	1084	295.10	1.03	178.55	7 / 7	-
A-n53-k7	1010	1841	0.030	1020	1291.31	0.99	122.65	7 / 7	-
A-n54-k7	1167	1883	0.051	1167	321.39	-	202.48	7 / 7	8
A-n55-k9	1073	2074	0.034	1073	1387.84	-	218.51	9 / 9	3
A-n60-k9	1354	2224	0.037	1354	1689.65	-	159.74	9 / 9	6
A-n61-k9	1034	2045	0.034	1037	1796.11	0.29	370.63	9 / 9	-
A-n62-k8	1288	2344	0.033	1290	3867.11	0.16	445.29	8 / 8	-
A-n63-k9	1616	2659	0.039	1629	1254.34	0.80	183.58	9 / 9	-
A-n63-k10	1314	2275	0.039	1318	3226.55	0.30	298.88	10 / 10	-
A-n64-k9	1401	2215	0.039	1431	386.78	2.14	-	9 / 9	-
A-n65-k9	1174	2331	0.045	1177	1577.88	0.26	552.54	9 / 9	-
A-n69-k9	1159	2463	0.068	1170	3752.35	0.95	1512.49	9 / 9	-
A-n80-k10	1763	3165	0.053	1763	9145.79	-	884.94	10 / 10	20

known value is reached or improved. As the maximum number of iterations has been fixed to 40, we have included the values obtained by the algorithm after these number of iterations for all problems not solved to optimality, but we have not specified any number in the last column.

Table 5.4 shows that this methodology is able to provide state-of-art results for large benchmark instances (100 customers or more) in terms of solution quality. As observed, final values are normally close to the best known integer solutions. It is also remarkable the result obtained for the largest selected test instance *G-n262-k25*, which stays slightly (0.44 %) over the best known value to the best of our knowledge, published by Hasle and Kloster [81].

Finally, two problems deserve special attention: *M-n200-k16* and *P-n55-k8*. The hybrid VNS methodology has been able to find a new best solution for the first and an alternative solution for the latter.

Table 5.3: Results obtained for class P problems. Improved solutions are marked in bold.

Problem	**BKS**	**IS**	CPU_{IS} **(s)**	**OBS**	CPU_{OBS} **(s)**	**Gap BKS-OBS (%)**	$CPU_{\leq 2\%}$ **(s)**	**#V/ #OV**	**Opt. It.**
P-n16-k8	450	534	0.020	453	0.62	0.67	0.62	8 / 8	-
P-n19-k2	212	267	0.156	219	0.57	3.30	-	2 / 2	-
P-n20-k2	216	266	0.003	216	0.95	-	0.95	2 / 2	1
P-n21-k2	211	276	0.003	211	2.75	-	2.75	2 / 2	1
P-n22-k2	216	278	0.003	216	4.68	-	4.68	2 / 2	1
P-n22-k8	603	687	0.009	603	1.20	-	1.20	8 / 8	1
P-n23-k8	529	642	0.016	529	2.43	-	2.43	8 / 8	8
P-n40-k5	458	773	0.014	458	39.43	-	39.43	5 / 5	1
P-n45-k5	510	827	0.015	510	111.05	-	111.05	5 / 5	1
P-n50-k7	554	1012	0.035	554	115.43	-	115.43	7 / 7	20
P-n50-k8	631	-	-	-	-	-	-	- / 8	-
P-n50-k10	696	1233	0.032	700	375.69	0.57	147.29	10 / 10	-
P-n51-k10	741	1248	0.031	741	427.25	-	181.10	10 / 10	5
P-n55-k7	568	1047	0.025	568	448.47	-	224.55	7 / 7	8
P-n55-k8 (a)	588	1093	0.029	**577**	247.69	-1.87	247.69	**7** / 8	1
P-n55-k8 (b)	588	1116	0.031	590	1379.75	0.34	68.99	8 / 8	-
P-n55-k10	694	1302	0.053	700	704.90	0.86	168.09	10 / 10	-
P-n55-k15	989	-	-	-	-	-	-	- / 15	-
P-n60-k10	744	1529	0.034	744	1671.03	-	321.79	10 / 10	12
P-n60-k15	968	1761	0.052	975	1023.88	0.72	178.30	15 / 15	-
P-n65-k10	792	1509	0.041	792	780.61	-	379.99	10 / 10	15
P-n70-k10	827	1586	0.055	842	3108.94	1.81	1293.94	10 / 10	-
P-n76-k4	593	1062	0.068	594	12328.96	0.17	3740.66	4 / 4	-
P-n76-k5	627	1177	0.042	628	10784.63	0.16	1167.90	5 / 5	-
P-n101-k4	681	1124	0.156	682	82818.18	0.15	16563.72	4 / 4	-

Table 5.4: Results obtained for large problems (100 or more customers). Improved solutions are marked in bold.

Problem	**BKS**	**IS**	CPU_{IS} **(s)**	**OBS**	CPU_{OBS} **(s)**	**Gap BKS-OBS (%)**	$CPU_{\leq 2\%}$ **(s)**	**#V/ #OV**	**Opt. It.**
E-n101-k8	817	1628	0.199	817	40888.91	-	10021.63	8 / 8	18
E-n101-k14	1067	2106	0.106	1084	4558.74	1.59	3655.71	14 / 14	-
M-n101-k10	820	1091	0.252	840	12897.92	2.44	-	10 / 10	-
M-n121-k7	1034	1227	0.193	1079	39383.90	4.35	-	7 / 7	-
M-n151-k12	1015	2481	0.287	1022	76933.89	0.69	70339.55	12 / 12	-
M-n200-k16	1371	3287	6.253	**1335**	109850.18	-2.63	12923.55	16 / 16	1
M-n200-k17	1275	3201	5.983	1304	195727.29	2.27	-	17 / 17	-
G-n262-k25	5685	14563	0.724	5710	443081.89	0.44	65442.80	25 / 25	-

For the test instance *M-n200-k16*, a new best solution with a cost of 1335 has been obtained. This solution is depicted in Figure 5.4. To the best of our knowledge, only one previous feasible solution with a cost of 1371 was known [81]. It was obtained by means of a VND algorithm embedded in the VRP software *SPIDER*. Taking into account the best known lower bound for this instance (1256.4), published by Baldacci et al. [15], the solution found by

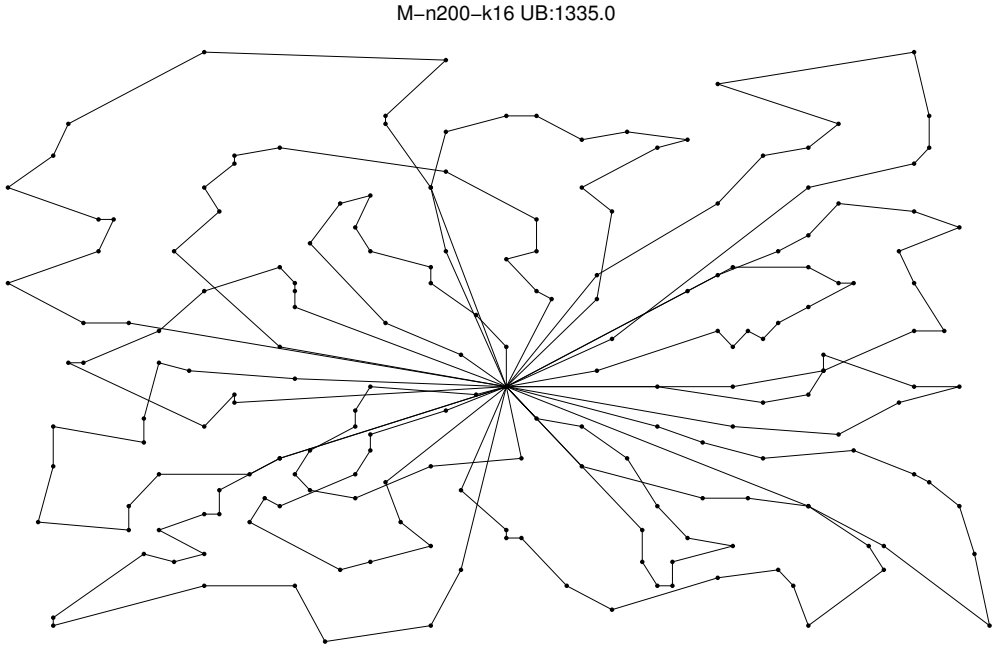

Figure 5.4: Proposed best solution for the test instance *M-n200-k16*.

means of the hybrid VNS algorithm reduces the gap between bounds from the previous value of 8.36 % to 5.89 %.

The solution found for the test instance *P-n55-k8* becomes an alternative to the published optimum [10]. For this case, a solution with a value of 577 has been found by using the swapping/relocate configuration, while a value of 576 has been obtained exchanging movements' priorities. This last solution is shown in Figure 5.5. Both solutions use 7 vehicles, instead of the 8 vehicles used in the published optimum. This solution has been marked as *P-n55-k8 (a)* in Table 5.3. *P-n55-k8 (b)* corresponds to the solution obtained when forcing the algorithm to use 8 vehicles. As far as we know, only two previous works, Alba and Dorronsoro [6] and Altinel and Öncan [7], have also presented this alternative value as the best known solution for this instance, while most authors keep the original cost of 588 using 8 vehicles as optimal.

In any case, optimality may not be guaranteed in the proposed solution for the instance *P-n55-k8*. Instances *P-n55-k7* and *P-n55-k8* share customers' distribution and demand, but the available vehicles have different capacities. Thus, both problems are critically different and the optimal cost of the instance

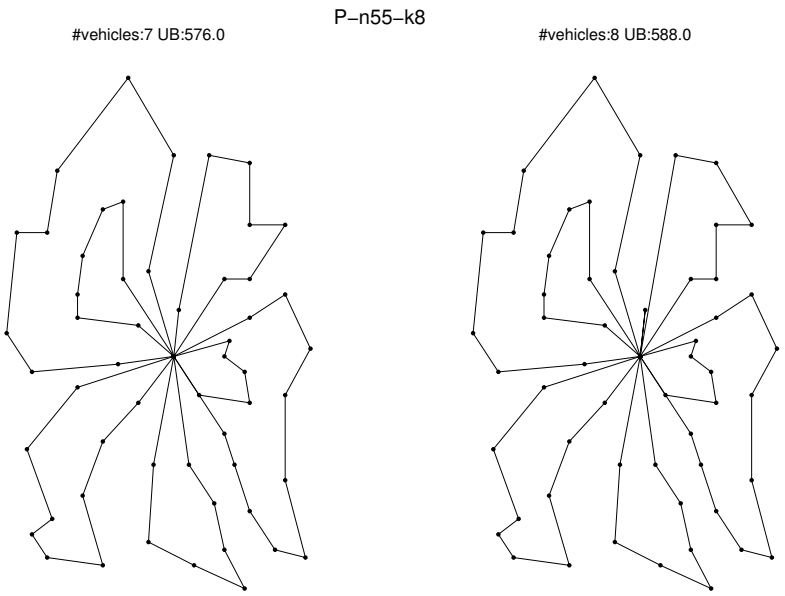

Figure 5.5: Alternative solution (left) to the published optimal configuration (right) for the test instance *P-n55-k8*. The proposed alternative solution has been obtained with a relocate/swapping configuration.

P-n55-k7 cannot be chosen as a reference. In fact, the solution of the instance *P-n55-k7* is not feasible according to *P-n55-k8* vehicles' capacity. The proposed solution has so the lowest known cost for this problem, but its optimality needs to be proved by means of an exact algorithm.

Computational complexity reduction of local search

The mechanisms for reducing the computational complexity of local search described in section 4.2.4 have been implemented in the hybrid VNS approach evaluated in this section. Some tests have been done in order to study the performance improvement obtained by means of these strategies.

The results obtained for the set A of benchmark instances are shown in Table 5.5. From these results, it can be stated that, in general, the strategies implemented to reduce the computational complexity of local search achieve the desired behavior. For most problems, the total calculation time is reduced

Table 5.5: Results obtained for class A problems comparing the original exhaustive local search method and a more efficient local search, including the computational complexity reduction strategies.

		Original local search			Efficient local search		
Problem	**BKS**	**OBS**	CPU_{OBS} **(s)**	**Gap BKS-OBS (%)**	**OBS**	CPU_{OBS} **(s)**	**Gap BKS-OBS (%)**
A-n32-k5	784	784	96.62	-	784	4.88	-
A-n33-k5	661	661	42.67	-	661	6.09	-
A-n33-k6	742	742	34.86	-	742	113.74	-
A-n34-k5	778	778	27.88	-	778	1.56	-
A-n36-k5	799	799	524.33	-	799	214.60	-
A-n37-k5	669	669	65.76	-	669	8.10	-
A-n37-k6	949	949	59.67	-	949	25.15	-
A-n38-k5	730	731	69.02	0.14	730	83.79	-
A-n39-k5	822	822	277.41	-	822	174.97	-
A-n39-k6	831	833	554.50	0.24	833	282.93	0.24
A-n44-k6	937	942	149.39	0.53	939	465.84	0.21
A-n45-k6	944	950	141.67	0.64	954	284.49	1.06
A-n45-k7	1146	1146	207.30	-	1158	479.01	1.05
A-n46-k7	914	914	265.87	-	914	112.23	-
A-n48-k7	1073	1084	295.10	1.03	1073	318.25	-
A-n53-k7	1010	1020	1291.31	0.99	1017	813.00	0.69
A-n54-k7	1167	1167	321.39	-	1172	861.31	0.43
A-n55-k9	1073	1073	1387.84	-	1074	739.85	0.09
A-n60-k9	1354	1354	1689.65	-	1354	675.40	-
A-n61-k9	1034	1037	1796.11	0.29	1037	886.55	0.29
A-n62-k8	1288	1290	3867.11	0.16	1302	1200.06	1.09
A-n63-k9	1616	1629	1254.34	0.80	1621	1248.39	0.31
A-n63-k10	1314	1318	3226.55	0.30	1323	1388.43	0.68
A-n64-k9	1401	1431	386.78	2.14	1419	1261.82	1.28
A-n65-k9	1174	1177	1577.88	0.26	1174	539.01	-
A-n69-k9	1159	1170	3752.35	0.95	1159	353.18	-
A-n80-k10	1763	1763	9145.79	-	1778	3203.25	0.85

while keeping the solution quality. For those problems from the set A not solved to optimality the average gap is 0.65 when using the original local search and 0.64 when the described strategies are applied. Therefore, these mechanisms help on getting a more competitive approach to the CVRP, although the times presented are still not competitive with most state-of-art heuristics.

Figure 5.6 compares the average time per iteration when using the former exhaustive local search and a more efficient one including the proposed strategies. Clearly, the introduced mechanisms get a linearithmic behavior on the instance size, as explained in section 4.2.4. Thus, algorithm's performance is significantly improved by considering these mechanisms, while final solution's quality remains unchanged.

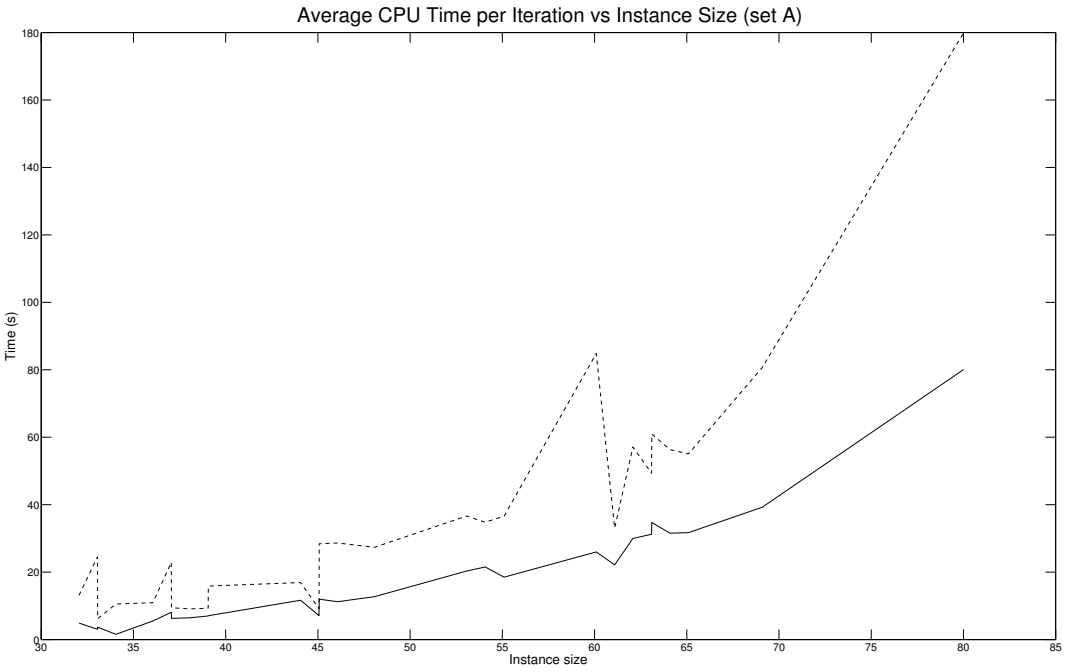

Figure 5.6: Average time per iteration when using the two different local search strategies. The dashed line represents the original exhaustive local search method, while the solid line depicts algorithm's behavior when applying the computational complexity reduction mechanisms.

Initial solution method

Although in all results presented in this section the initial solution has been obtained by solving separately capacity and routing problems, other methods may be used to get the initial solution. Table 5.6 presents a comparative of results for some problems from the set A using different techniques to obtain an initial solution.

First, it may be obtained using the proposed problem decomposition in a CP/LR schema (1), as mentioned. This approach is able to provide a low-quality quick solution, since both subproblems are easily solved but their variables are unlinked. This solution may be highly improved applying a VND method, providing an initial solution whose value is usually close to the final result (2). The initial solution may be also obtained by means of the RCWS algorithm (3) described in section 4.3.1. This algorithm provides a good initial solution in negligible times, but the maximum number of available vehicles

might not be always respected. For this reason, it may be forced to execute iteratively until a solution fulfilling this requirement is reached (4).

Table 5.6: Results obtained for some class A benchmark instances comparing different methods to get an initial solution: (1) CP/LR schema without VND, (2) CP/LR schema + VND, (3) RCWS and (4) RCWS forced to use the minimum number of required vehicles.

Problem	**BKS**	**Init.Sol.**				**CPU (s)**			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	784	1243	989	840	876	0.096	0.177	0.018	0.020
A-n36-k5	799	1207	815	838	859	0.010	0.011	0.004	0.005
A-n38-k5	730	1240	786	790	805	0.008	0.009	0.022	0.004
A-n39-k5	822	1291	906	870	856	0.011	0.010	0.004	0.004
A-n44-k6	937	1547	987	980	1040	0.013	0.013	0.003	0.104
A-n45-k7	1146	1768	1244	1267	1284	0.019	0.018	0.004	0.004
A-n53-k7	1010	1841	1105	1072	1057	0.022	0.021	0.005	0.005
A-n54-k7	1167	1883	1284	1238	1216	0.094	0.029	0.005	0.009
A-n55-k9	1073	2088	1177	1105	1118	0.026	0.030	0.005	0.005
A-n63-k9	1616	2659	1839	1657	1711	0.032	0.037	0.012	0.022
A-n63-k10	1288	2344	1323	1352	1360	0.026	0.027	0.008	0.004

Problem	**BKS**	**Fin.Sol.**				**CPU (s)**			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	784	784	784	784	784	5.36	4.88	3.22	6.23
A-n36-k5	799	799	799	799	799	74.17	214.60	34.82	163.41
A-n38-k5	730	730	730	730	730	172.60	83.79	53.84	18.92
A-n39-k5	822	825	822	822	822	122.42	174.97	33.14	253.68
A-n44-k6	937	942	939	942	939	37.62	66.70	40.77	434.59
A-n45-k7	1146	1146	1158	1146	1146	294.07	464.83	156.67	243.55
A-n53-k7	1010	1017	1017	1017	1017	313.78	535.10	244.51	497.05
A-n54-k7	1167	1167	1172	1167	1172	606.60	586.11	282.75	640.31
A-n55-k9	1073	1074	1074	1073	1073	104.05	354.78	631.93	519.22
A-n63-k9	1616	1635	1621	1621	1635	660.21	760.76	1002.37	1049.66
A-n63-k10	1288	1302	1302	1308	1308	965.23	197.05	71.21	484.29

Problem	**BKS**	**Gap BKS-FS (%)**				**# it**			
		(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	784	-	-	-	-	1	1	1	1
A-n36-k5	799	-	-	-	-	16	39	7	26
A-n38-k5	730	-	-	-	-	31	13	7	2
A-n39-k5	822	0.36	-	-	-	17	25	5	33
A-n44-k6	937	0.53	0.21	0.53	0.21	1	4	3	2
A-n45-k7	1146	-	1.05	-	-	22	39	9	17
A-n53-k7	1010	0.69	0.69	0.69	0.69	14	26	12	27
A-n54-k7	1167	-	0.43	-	0.43	29	26	12	30
A-n55-k9	1073	0.09	0.09	-	-	3	16	37	30
A-n63-k9	1616	1.18	0.31	0.31	1.18	22	24	34	32
A-n63-k10	1288	1.09	1.09	1.55	1.55	27	4	2	13

It can be stated from results presented in Table 5.6 that RCWS is clearly faster than the other methods to get a good initial solution. Even when it is forced to iterate to reach a given number of vehicles, it may provide a solution quickly. It is remarkable that this algorithm eventually finds a comparable, or even better, initial solution than the CP/LR + VND schema in a much lower time, becoming so a better alternative. For small problems, the RCWS combined with the hybrid VNS framework outperforms the same methodology using the CP/LR schema to provide the initial solution, reaching best known values in lower times. On the other hand, times are comparable for larger problems. For small problems, feasible points around a given solution may be usually low, while the solution space may grow dramatically as the problem size increases, and so does the time needed to explore it.

Using the RCWS algorithm to get an initial solution allows solving to the best known value 15 of 27 problems from the set A. Instead, the CP/LR + VND combination allows solving 14 problems. This method uses all four defined moves to reach a minimum. As they are also used in the hybrid VNS framework, the initial solution is a local optimum for all moves defined in its local search processes. Thus, a more thorough shaking would be mandatory to get a better algorithm's performance and avoid getting trapped in local minima. Moreover, the algorithm fails to successfully solve the same problems, regardless which method is used to get the initial solution. Again, a poor shaking might be the main reason for this behavior. Table 5.6 also shows the gap between the final solution and the best known value, when it is not reached. It can be observed that it is usually low when a good initial solution is provided.

Table 5.7 shows times needed to reach good quality solutions, i.e. with a gap lower or equal to 2 % from the best known solution, for different benchmark instances. It presents results comparing different methods to get an initial solution. It can be observed that, in average, using the RCWS algorithm to obtain the initial solution permits reaching a high quality solution in much lower times. This characteristic makes this method a better choice when running the algorithm for a fixed period of time, since it is more likely to provide a better final solution.

Finally, it is remarkable that combining the RCWS algorithm with the hybrid VNS methodology has been able to improve the best published result for the largest test instance *G-n262-k25*. To the best of our knowledge, the best integer result (5685) for this problem was published by Hasle and Kloster [81]. The result obtained by means of the proposed methodology reduces this

Table 5.7: Times required to reach a high quality solution for some class A benchmark instances when different methods to get an initial solution are used: (1) CP/LR schema without VND, (2) CP/LR schema + VND, (3) RCWS and (4) RCWS forced to use the minimum number of required vehicles.

Problem	$CPU_{\leq 2\%}$ (s)				$CPU_{\leq 1\%}$ (s)			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
A-n32-k5	5.36	4.89	3.22	6.23	5.36	4.89	3.22	6.23
A-n36-k5	27.09	26.66	15.49	12.26	74.17	63.42	34.82	12.26
A-n38-k5	16.79	80.71	18.72	7.27	16.79	83.79	30.13	18.92
A-n39-k5	13.56	18.32	5.71	9.16	13.56	18.32	18.28	30.54
A-n44-k6	37.62	26.05	28.35	33.41	37.62	26.05	28.35	33.41
A-n45-k7	44.47	53.58	16.64	38.41	44.47	-	47.81	62.68
A-n53-k7	193.28	139.10	37.83	14.89	313.78	535.10	37.83	497.05
A-n54-k7	115.70	156.42	17.65	50.25	115.70	438.14	282.75	557.86
A-n55-k9	57.42	93.72	29.98	17.10	104.05	192.08	29.98	43.32
A-n63-k9	127.56	196.37	29.79	135.26	-	196.37	182.11	-
A-n63-k10	130.23	130.42	53.74	474.66	819.65	130.42	53.74	474.66
Average:	*69.92*	*84.20*	*23.37*	*72.63*	*154.51*	*168.86*	*68.09*	*173.69*

value a 1.95 %, setting it to 5574. This result reduces the gap between the best known lower and upper bounds from 10.92 % to 9.15 %. Figure 5.7 shows the obtained solution for this problem.

5.1.2 Multi-Start Variable Neighborhood Search

The methodology described in section 4.3 has been implemented in Java and linked to the open-source CP software system ECLiPSe 6.0 [9]. All tests have been performed on a dedicated server with an Intel Xeon Quad-Core i5 processor at 2.66GHz and 16GB RAM. A total of 91 benchmark instances have been solved and used to test the efficiency of the described approach when dealing with this simple —in terms of constraints— but extensively tested scenario. The selected problems include 7 instances from Christofides et al. [44] (denoted in tables as *C1*-*C5*, *C11*, and *C12*). Although these problems are equivalent to some instances from the sets E and M, we also use this notation for further comparison with some recent metaheuristics.

In all tests, *relocate* (see section 4.2.2 for movements description) has been set as the initial move in local search processes ($k = 1$). Since the implemented algorithm is based on the VND approach, there is no shaking process and the diversification is obtained by means of the multi-start approach (see section 4.3). *Swapping*, *chain*, and *ejection chain* are used next whenever the previous solution is not improved. As in the previous methodology, the relative per-

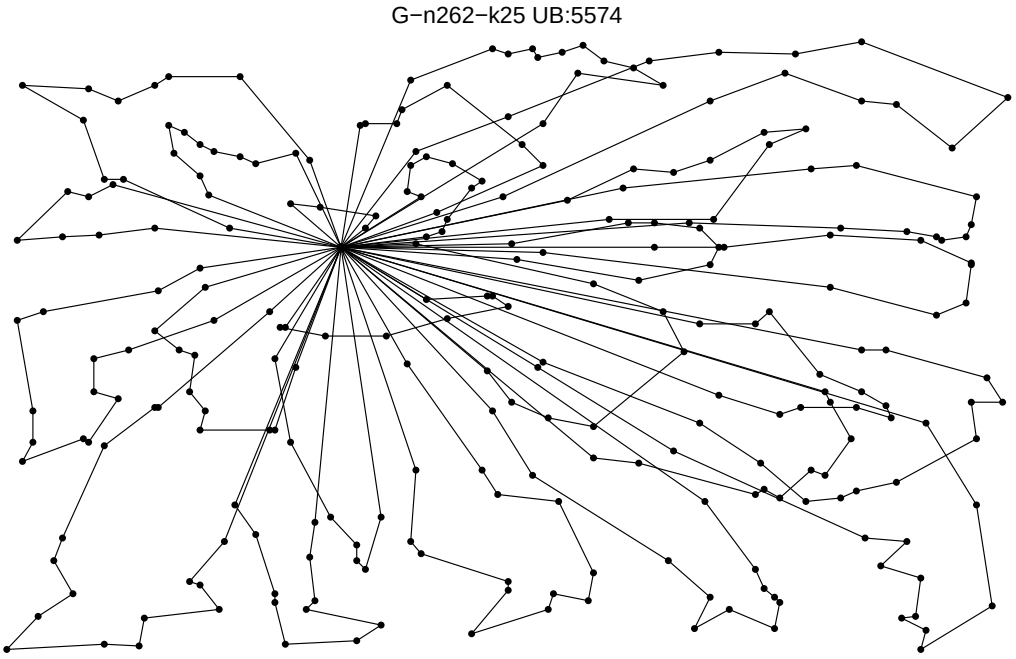

Figure 5.7: Proposed best solution for the test instance *G-n262-k25*.

centage of customers modified by the ejection chain movement has been set to 40 %.

As the algorithm has been designed to be run in a parallel computing environment, a test has been done over the set A of benchmark problems to determine the most suitable number of simultaneous threads. This parameter is to be fixed mainly according to computer's characteristics. The total number of threads (N_{total} in Algorithm 4.6) has been set up to 50, while the number of parallel threads (N_{max} in Algorithm 4.6) grows from 1 to 50. According to this test, the average time spent on calculating a pseudo-optimal solution increases smoothly while the number of parallel threads is below 8. This result was to be expected due to quad-core processors characteristics. Figure 5.8(b) represents the relation between the average time spent on calculating a solution and the number of concurrent threads, showing the described trend.

In the particular server used with this methodology, up to 4 threads may be executed in parallel in order to keep a reasonable computational efficiency. As shown in Figure 5.8(a), the lowest total computational times are obtained when running 4 to 7 threads simultaneously. Since the average time to calculate

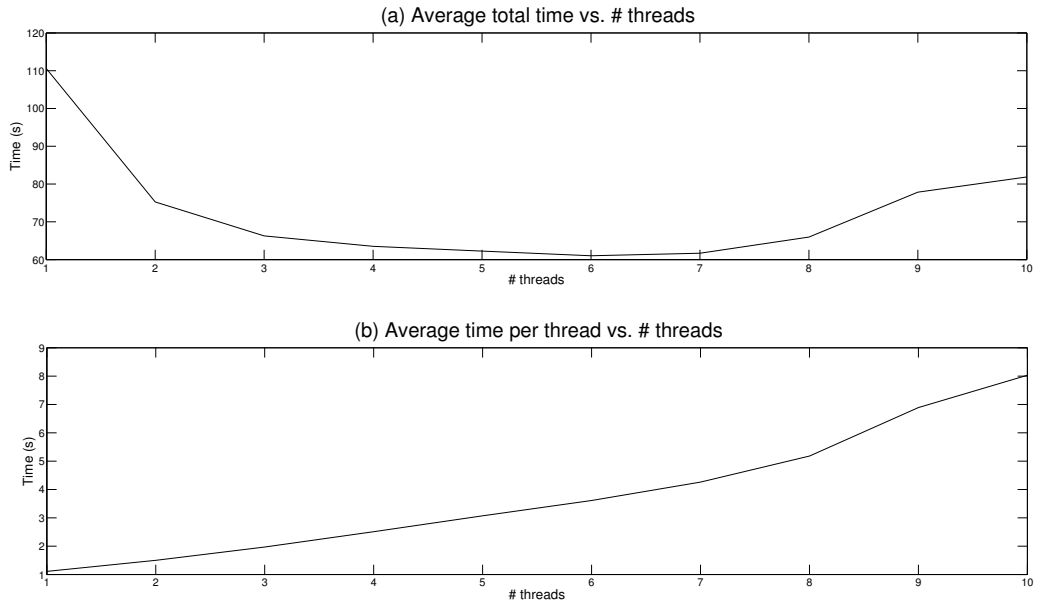

Figure 5.8: (a) Average total computation time and (b) average time per thread as the number of concurrent threads increases.

one pseudo-optimal solution increases with the number of concurrent threads, the best trade-off is found when 4 calculations are run in parallel. For this reason, all results presented in this section correspond to a Multi-Start VND implementation with 4 parallel processes, since this approach has demonstrated to keep a reasonable balance between the time spent on calculating one single solution and the total execution time. Adopting this parallelized approach permits reducing the total computation time significantly. In the performed test for problems from the set A, the total computation time is 41 % lower, in average, than the total time spent using a sequential approach.

Table 5.8 shows results obtained for some representative problems from the selected benchmark sets. Due to algorithm's probabilistic behavior, the final solutions' quality depends on the total number of threads. For this reason, 100 total tasks have been considered for each problem, i.e. 100 pseudo-optimal solutions have been generated for each benchmark instance. Table 5.8 summarizes information regarding the best solution found (OBS) for each problem, as well as the time required to reach this solution. These results are compared to the best known solutions (BKS) so far. As mentioned in the previous section,

Table 5.8: Results for 50 classical benchmark instances.

Problem	**BKS**	**OBS**	**Gap BKS-OBS (%)**	**#V/ #OV**	CPU_{OBS} **(s)**
A-n32-k5	787.81	787.08	-0.09	5 / 5	0.633
A-n33-k5	662.76	662.11	-0.10	5 / 5	0.842
A-n33-k6	742.83	742.69	-0.02	6 / 6	0.480
A-n37-k5	672.59	673.59	0.15	5 / 5	1.948
A-n37-k6	952.22	950.85	-0.14	6 / 6	1.631
A-n38-k5	734.18	733.95	-0.03	5 / 5	2.546
A-n45-k6	944.88	944.88	0.00	6 / 6	1.622
A-n46-k7	918.46	918.13	-0.04	7 / 7	2.062
A-n54-k7	1171.78	1171.78	0.00	7 / 7	4.007
A-n55-k9	1074.46	1076.85	0.22	9 / 9	5.544
A-n63-k9	1622.14	1622.14	0.00	9 / 9	8.073
B-n31-k5	676.76	676.09	-0.10	5 / 5	0.657
B-n34-k5	791.24	789.84	-0.18	5 / 5	0.497
B-n35-k5	956.29	958.94	0.28	5 / 5	1.174
B-n38-k6	809.45	809.45	0.00	6 / 6	1.211
B-n39-k5	553.27	553.16	-0.02	5 / 5	1.577
B-n43-k6	747.54	746.98	-0.07	6 / 6	1.520
B-n45-k5	755.43	753.96	-0.19	5 / 5	1.011
B-n50-k7	744.78	744.23	-0.07	7 / 7	1.721
B-n50-k8	1316.20	1319.53	0.25	8 / 8	7.069
B-n51-k7	1035.71	1037.54	0.18	7 / 7	597.915
B-n57-k9	1603.63	1604.88	0.08	9 / 9	7.653
B-n64-k9	869.32	868.31	-0.12	9 / 9	287.953
E-n22-k4	375.28	375.28	0.00	4 / 4	0.337
E-n23-k3	568.56	568.56	0.00	3 / 3	0.422
E-n33-k4	838.72	837.67	-0.13	4 / 4	0.819
E-n51-k5 (C1)	524.61	527.98	0.64	5 / 5	17.164
E-n76-k10 (C2)	835.26	843.49	0.99	10 / 10	28.941
E-n101-k8 (C3)	826.14	841.16	1.82	8 / 8	195.271
F-n45-k4	724.57	727.75	0.44	4 / 4	4.459
F-n135-k7	1170.65	1179.09	0.72	7 / 7	630.427
G-n262-k25	5685.00	5722.00	0.65	25 / 25	1651.360
M-n101-k10 (C12)	819.81	821.40	0.19	10 / 10	51.395
M-n121-k7 (C11)	1042.11	1045.14	0.29	7 / 7	137.553
M-n151-k12 (C4)	1028.42	1052.52	2.34	12 / 12	834.642
M-n200-k17 (C5)	1291.45	1324.91	2.59	17 / 17	243.789
P-n16-k8	451.95	451.95	0.00	8 / 8	0.019
P-n19-k2	212.66	212.66	0.00	2 / 2	0.243
P-n20-k2	217.42	217.42	0.00	2 / 2	0.148
P-n21-k2	212.71	212.71	0.00	2 / 2	0.275
P-n22-k2	217.85	217.85	0.00	2 / 2	0.277
P-n23-k8	531.17	531.17	0.00	8 / 8	1.447
P-n40-k5	461.73	461.73	0.00	5 / 5	6.189
P-n45-k5	512.79	512.79	0.00	5 / 5	10.016
P-n50-k7	559.86	560.15	0.05	7 / 7	5.155
P-n51-k10	742.48	742.36	-0.02	10 / 10	5.156
P-n55-k10	697.81	698.00	0.03	10 / 10	5.331
P-n55-k8	592.17	581.17	-1.86	7 / 8	14.703
P-n76-k5	635.04	633.32	-0.27	5 / 5	92.627
P-n101-k4	692.28	693.54	0.18	4 / 4	839.622
Average:			*0.17*		

most sources give these values as integer numbers, obtained by rounding costs, except for the problems from [44] where real values are usually given. To adopt a more realistic approach and comparing the obtained results to other methodologies, we have calculated the real costs from the detailed integer solutions. It should be remarked that the real cost of an integer optimal solution might not correspond to the optimal solution considering real costs. For this reason, negative gaps appear on this table. Thus, it can be deduced that the Multi-Start VND is able to match, and in many cases overcome, the real value associated to integer best known solutions. Concretely, the presented approach has been able to overcome 23 best known solutions, considering real costs, out of the 91 tested instances. In addition, the gap is kept reasonably low for all considered instances, being the average gap 0.65 %. It remains lower, 0.17 %, for the problems selected in Table 5.8, which include most of the largest instances.

Furthermore, it should be remarked that these results have been obtained in competitive times even for some large instances. As shown in Table 5.8, most solutions for small problems are obtained in less than a second, while larger instances require higher yet reasonable computational times. In most cases, higher times are closely related to higher quality solutions, i.e. solutions with a negative gap.

As a final remark, it can be observed that the lowest gap (-1.86 %) corresponds to the problem *P-n55-k8*, where a solution considering only 7 vehicles ($\#V$) has been obtained. As discussed for the previous methodology (see section 5.1.1), although the best known solution for this problem uses 8 vehicles, feasible solutions with 7 vehicles and lower costs may be reached, as the one obtained with this approach. However, if only 7 vehicles are considered, the Multi-Start VND has finished slightly over the value 580.96 (576 considering integer costs), presented in the previous section and published in [6] and [7].

5.1.3 Comparison between approaches and other heuristics

In this section, we provide a brief comparison of the results obtained for the hybrid VNS and the Multi-Start VND approaches, described in sections 4.2 and 4.3, respectively. Furthermore, we compare these values with the results published for some state-of-art metaheuristics.

Most works devoted to the CVRP present solutions only for those problems

contained in the benchmark set by Christofides et al. [44]. This set contains 14 instances: 7 CVRP with a side constraint limiting the total length of the tours and 7 corresponding to the classic CVRP. For this reason, only the latter 7 instances from this benchmark set have been considered in this thesis. As mentioned, they are equivalent to some problems from the sets E and M and have been denoted in Tables 5.9 and 5.10 as *C1-C5*, *C11*, and *C12*, as in the previous section.

Most sources give the solutions to these problems as real numbers, while integer solutions are usually reported for the other benchmark sets. Nevertheless, some approaches like those from Juan et al. [93] [92] present values based on real distances, and not on distances obtained by rounding initial costs. In order to clarify the comparison, we have calculated the real costs corresponding to the best known integer solutions —as already done in section 5.1.2. Moreover, we have calculated the real values corresponding to the solutions obtained by means of the hybrid VNS approach and presented in section 5.1.1. Again, it should be remarked that the real cost of an integer optimal solution might not correspond to the optimal solution considering real costs.

Tables 5.9 and 5.10 provide a comparison between the two hybrid methodologies presented in this thesis and some recent publications. The hybrid VNS approach presented in section 4.2 is denoted as *HVNS*. As mentioned, the solutions presented in Table 5.9 correspond to the real values calculated from the detailed integer solutions obtained by means of this methodology. Times presented for this approach (Table 5.10) improve those reported in section 5.1.1, since they include the strategies introduced to reduce the computational complexity of local search processes. Column *MS-VND* in Tables 5.9 and 5.10 presents some results obtained with the Multi-Start VND algorithm described in section 4.3.

As it may be observed, both presented methodologies are able to match —or even improve— most of the best known solutions for the selected problems. Although both approaches get similar pseudo-optimal solutions, the hybrid VNS algorithm is not competitive in terms of computational time. In this sense, the Multi-Start VND method obtains similar solutions in significantly lower times. Thus, the improvements introduced in the latter turns this approach into a more suitable methodology to tackle the CVRP than the former one.

Table 5.9: Obtained best solutions for the two hybrid methodologies proposed in this thesis and other approaches for some classical benchmark instances.

Problem	**BKS**	**SA-TS**	**HEMA**	**SR-2**	**SR-GCWS**	**SR-GCWS-CS**	**HVNS**	**MS-VND**
A-n32-k5	787.81	-	-	-	787.08	787.08	787.20	787.08
A-n33-k5	662.76	-	-	662.76	662.11	-	663.49	662.11
A-n33-k6	742.83	-	-	-	742.69	-	743.27	742.69
A-n37-k6	952.22	-	-	-	-	-	952.05	950.85
A-n38-k5	734.18	-	-	-	733.95	733.95	734.18	733.95
A-n45-k6	944.88	-	-	-	944.88	-	944.88	944.88
A-n46-k7	918.46	-	-	918.46	-	-	918.05	918.13
A-n63-k9	1622.14	-	-	-	1622.14	-	1629.69	1622.14
B-n31-k5	676.76	-	-	-	676.09	-	676.13	676.09
B-n39-k5	553.27	-	-	-	553.16	-	553.81	553.16
B-n45-k5	755.43	-	-	755.43	754.22	-	755.14	753.96
B-n50-k7	744.78	-	-	-	744.23	744.23	746.08	744.23
E-n33-k4	838.72	-	-	-	837.67	837.67	837.87	837.67
E-n51-k5 (C1)	524.61	524.61	524.61	524.61	524.61	524.61	524.61	527.98
E-n76-k10 (C2)	835.26	835.26	849.77	844.42	835.26	835.28	841.70	843.49
E-n101-k8 (C3)	826.14	826.14	844.72	829.40	-	-	828.99	841.16
M-n101-k10 (C12)	819.81	819.56	847.56	819.56	819.56	819.56	821.18	821.40
M-n121-k7 (C11)	1042.11	1045.50	1042.11	1052.34	1043.88	1043.88	1046.06	1045.14
M-n151-k12 (C4)	1028.42	1038.71	1059.03	1048.89	-	-	1035.80	1052.52
M-n200-k17 (C5)	1291.45	1311.70	1302.33	1323.89	-	-	1325.07	1324.91
P-n20-k2	217.42	-	-	-	217.42	-	217.42	217.42
P-n22-k2	217.85	-	-	-	217.85	-	217.85	217.85
P-n51-k10	742.48	-	-	-	741.50	-	742.48	742.36
P-n55-k8	592.17	-	-	-	-	-	580.96	581.17
P-n76-k5	635.04	-	-	-	633.32	633.32	635.04	633.32

In order to compare the methodologies described in this thesis with other approaches, we provide the results for some recent publications in Tables 5.9 and 5.10. Although a direct comparison between different algorithms may be a delicate issue —they are normally implemented in different languages and executed in different computing environments—, some conclusions may be inferred. For this work, we have selected five heuristic approaches to the CVRP. The first three metaheuristics are a hybrid algorithm of Simulated Annealing and Tabu Search (*SA-TS*) introduced by Lin et al. [113], a hybrid Electromagnetism-like heuristic (*HEMA*) proposed by Yurtkuran and Emel [179], and a Particle Swarm algorithm (*SR-2*) described by Jin Ai and Kachitvichyanukul [5]. Finally, we have included the randomized Clarke and Wright Savings (*SR-GCWS*) algorithm by Juan et al. [93] and its improved version using cache and splitting techniques (*SR-GCWS-CS*), introduced by the same authors in [92].

It may be observed that the results reported for the methodologies presented in this thesis are similar to those obtained by means of other state-of-art metaheuristics. The proposed approaches are comparable in terms of quality. The Multi-Start VND methodology is also competitive regarding computational efficiency. Times needed by this approach to reach a pseudo-optimal solution are in most cases lower than those required by means of the other algorithms.

It is remarkable that the Multi-Start VND approach clearly improves the efficiency of the two algorithms which form its basis: the hybrid VNS methodology and the randomized version of the Clarke and Wright Savings heuristic (*SR-GCWS*). For most problems, only the improved version of the SR-GCWS algorithm is able to reach similar computational times. Furthermore, the hybrid VNS and the Multi-Start VND methodologies provide the lowest average gaps among the selected metaheuristics, only beated by the SR-GCWS and the SR-GCWS-CS approaches. However, most of the higher gaps obtained with the methodologies proposed in this thesis correspond to some of the largest instances, whose results are not reported for the SR-GCWS and the SR-GCWS-CS algorithms.

Table 5.10: Computational times to get the best obtained solutions for the two hybrid methodologies proposed in this thesis and other approaches for some classical benchmark instances.

Problem	**SA-TS** (s)	**HEMA** (s)	**SR-2** (s)	**SR-GCWS** (s)	**SR-GCWS-CS** (s)	**HVNS** (s)	**MS-VND** (s)
A-n32-k5	-	-	-	6.00	1.00	4.88	0.63
A-n33-k5	-	-	13.00	2.00	-	6.09	0.84
A-n33-k6	-	-	-	3.00	-	6.42	0.48
A-n37-k6	-	-	-	-	-	25.15	1.63
A-n38-k5	-	-	-	7.00	1.00	11.58	2.55
A-n45-k6	-	-	-	31.00	-	169.42	1.62
A-n46-k7	-	-	23.00	-	-	112.23	2.06
A-n63-k9	-	-	-	145.00	-	1248.39	8.07
B-n31-k5	-	-	-	1.00	-	90.45	0.66
B-n39-k5	-	-	-	17.00	-	198.37	1.58
B-n45-k5	-	-	20.00	20.00	-	119.13	1.01
B-n50-k7	-	-	-	2.00	1.00	42.09	1.72
E-n33-k4	-	-	-	7.00	1.00	31.81	0.82
E-n51-k5 (C1)	38.14	13.00	24.00	32.00	1.00	1026.30	17.16
E-n76-k10 (C2)	118.27	19.00	57.00	21.71	231.00	3573.40	28.94
E-n101-k8 (C3)	293.25	41.00	101.00	-	-	11734.63	195.27
M-n101-k10 (C12)	316.02	190.00	88.00	338.00	0.00	408.29	51.40
M-n121-k7 (C11)	332.77	319.00	93.00	74.49	107.00	24513.66	137.55
M-n151-k12 (C4)	701.38	132.00	223.00	-	-	87924.44	834.64
M-n200-k17 (C5)	1844.34	201.00	413.00	-	-	223688.34	243.79
P-n20-k2	-	-	-	41.00	-	0.95	0.15
P-n22-k2	-	-	-	9.00	-	1.28	0.28
P-n51-k10	-	-	-	19.00	-	427.28	5.16
P-n55-k8	-	-	-	-	-	878.96	14.70
P-n76-k5	-	-	-	73.00	3.00	9129.08	92.63

5.2 Constraint Programming approach to the VRP

As mentioned in section 4.4.2, the application of exact CP techniques to solve VRP problems implies high computational times and therefore requires the definition of heuristic methods. Even though, the involved computational effort may be too high to be competitive with other state-of-art approaches. For this reason, we use the described approach to solve small CVRP instances. Furthermore, the methods introduced in section 4.4.2 are not able to cope with most of the VRPTW instances. Nevertheless, we can use the described methodology to solve these instances if soft time windows are considered. Since they correspond to relaxed variants of the original problems, our approach may be used to provide reasonable lower bounds. In any case, none of these applications are competitive with other approaches and further efforts need to be addressed in this direction. However, the aim of our methodology was to establish an initial framework as a first approach to richer VRP problems. A CP solver gains in performance as the problem gets more constrained, providing thus some advantages over other approaches.

The CP-based methodology described in section 4.4.2 has been implemented in the open-source CP software system ECLiPSe 6.0 [9]. All tests have been performed on a dedicated server with an Intel Xeon Quad-Core i5 processor at 2.66GHz and 16GB RAM.

In this particular implementation, the maximum number of pivots to be chosen is set to 10. Moreover, 5 neighboring visits are selected around each pivot. Thus, for larger VRPTW instances involving 100 customers, the RPOP operator is able to destroy up to 60 % of the current solution. Since it corresponds to a large part of the solution and the solver may require high computational times to re-optimize it, the branch-and-bound searches performed by the repair method are limited to 60 seconds. The total execution time is also set as the stopping condition for the VND-LNS algorithm.

Finally, it should be remarked that all initial solutions have been obtained by doing a simple labeling for all variables. This solution is later improved by means of a branch-and-bound algorithm limited to 60 seconds. We strongly believe that the use of other methods to get the initial solution, such as the RCWS algorithm or Solomon's heuristics [163], would provide better final results. Nevertheless, the tests presented in this section are aimed to validate the described formulation and methodology, which needs to be further improved

to be comparable with other approaches.

CVRP

As mentioned, the CP-based approach described in section 4.4.2 may be used to solve small CVRP instances. For this reason, we have selected instances with up to 30 customers from the CVRP benchmark sets E and P. In addition, the smallest instances from the sets A and B have also been included, with 32 and 31 visits respectively. In total, 11 instances ranging from 16 to 32 customers have been used in this test.

For a clearer comparison, integer distances have been considered, since most publications provide integer solutions for the selected instances. In addition, although CP methods are not restricted to work with finite integer domains, their performance is better when such domains are defined. Thus, using integer distances becomes a reasonable approach from a CP perspective.

In order to determine the efficiency of the Nearest Neighbor-based variable ordering heuristic introduced in section 4.4.3, a test has been done over the selected instances. To avoid the effects of the random behavior of the RPOP operator, a deterministic branch-and-bound algorithm has been used to solve the selected problems. In this test, the execution of the branch-and-bound algorithm was limited to 300 seconds. The obtained results are presented in Table 5.11. In this table, solutions reached by the algorithm (OS) are shown, as well as their corresponding gaps to the best known solutions (BKS), either using the Nearest Neighbor variable ordering heuristic or not.

Table 5.11: Results obtained applying a simple branch-and-bound algorithm and the same method combined with a Nearest Neighbor variable ordering heuristic during 300 seconds.

Problem	**BKS**	**BB**		**BB + heuristic**	
		OS	**Gap BKS OS (%)**	**OS**	**Gap BKS OS (%)**
A-n32-k5	784	1691	115.7	1337	70.5
B-n31-k5	672	1366	103.3	1163	73.1
E-n22-k4	375	491	30.9	488	30.1
E-n30-k3	534	960	79.8	807	51.1
P-n16-k8	450	495	10.0	495	10.0
P-n19-k2	212	331	56.1	310	46.2
P-n20-k2	216	350	62.0	307	42.1
P-n21-k2	211	366	73.5	300	42.2
P-n22-k2	216	379	75.5	309	43.1
P-n22-k8	603	663	10.0	663	10.0
P-n23-k8	529	636	20.2	636	20.2

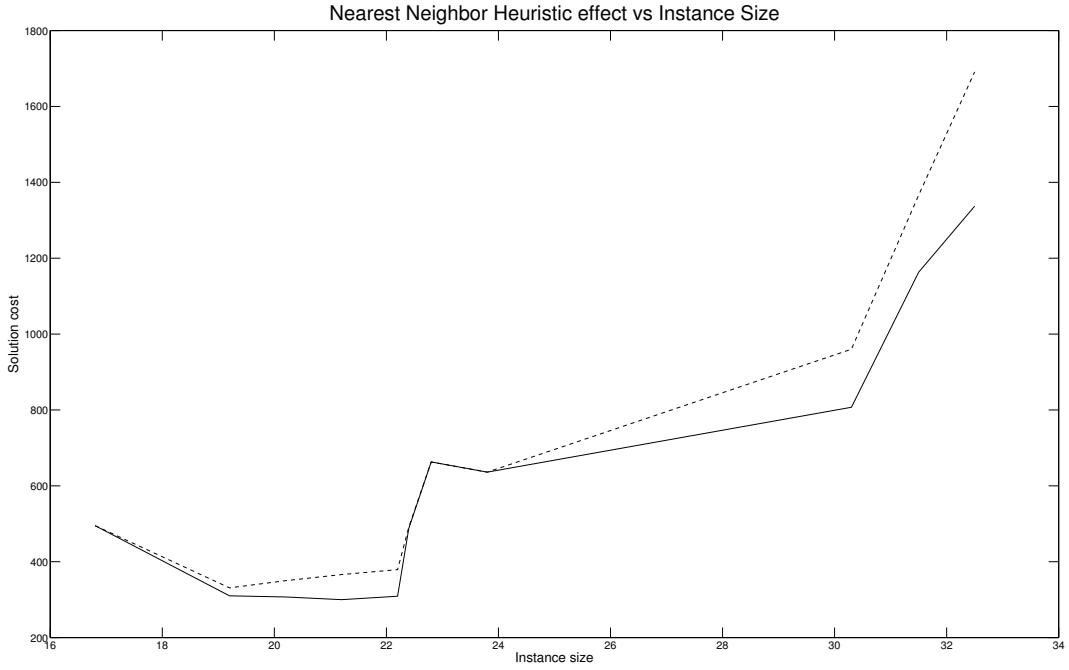

Figure 5.9: Nearest Neighbor-based variable ordering heuristic effect on the solution cost. The solid line represents the results for a branch-and-bound algorithm combined with this heuristic, while the dashed line depicts the behavior for the same algorithm without using it.

As expected, results in Table 5.11 show that solutions obtained with the variable ordering heuristic improve those obtained by means of a branch-and-bound algorithm without introducing any mechanism to guide the search. In all cases, the branch-and-bound algorithm combined with the Nearest Neighbor-based heuristic is able to match or overcome the results reached without this variable ordering method when run during the same time. Figure 5.9 depicts the effects of using this heuristic on the solution cost obtained after 300 seconds of execution. The dashed line represents the cost for the branch-and-bound algorithm, while the solid line shows the behavior of the same method combined with the variable ordering heuristic.

Moreover, the number of backtracks is significantly reduced. A CP-based search method backtracks any time a partial solution either violates any of the defined constraints or has a higher cost than the upper bound at that point of search. In both cases, backtracking is performed to the most recently instanti-

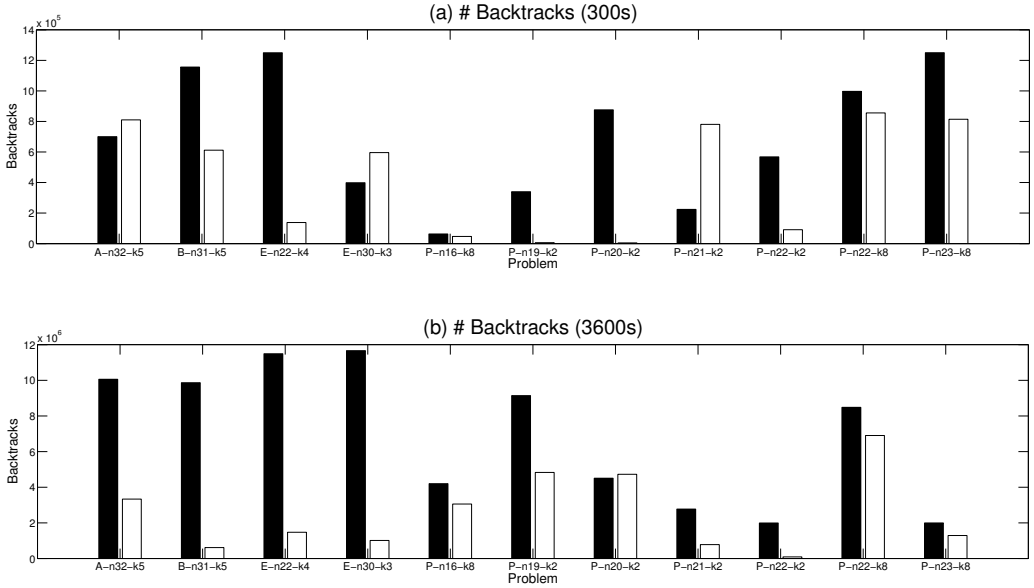

Figure 5.10: Number of backtracks for a branch-and-bound algorithm run for (a) 300 s and (b) 3600 s. Black bars show the number of backtracks performed without the Nearest Neighbor variable ordering heuristic, while white bars represent the number of backtracks considering it.

ated variable that still has available alternative values in its domain. Therefore, reducing the number of backtracks implies a more efficient exploration of the search tree. Figure 5.10 shows how the use of the Nearest Neighbor variable ordering heuristic reduces the number of backtracks for most selected problems. Black bars represent the number of backtracks performed during the search when a simple branch-and-bound algorithm is used. White bars indicate the number of backtracks obtained when the variable ordering heuristic is considered. As can be seen, differences in the number of backtracks increase as the algorithm is run for longer periods. For these reasons, it can be stated that using the Nearest Neighbor-based variable ordering heuristic helps on improving the search efficiency for the CVRP problem.

Table 5.12 shows the results obtained by means of the methodology described in section 4.4.2 for the selected instances. The execution time of the algorithm has been limited to 300 seconds for all problems. The results obtained

Table 5.12: Results obtained for some small CVRP instances using the CP-based VND-LNS methodology during 300 s.

Problem	BKS	BB + heuristic		VND-LNS		VND-LNS + heuristic	
		OS	Gap BKS OS (%)	OS	Gap BKS OS (%)	OS	Gap BKS OS (%)
A-n32-k5	784	1337	70.5	1313	67.5	1392	77.6
B-n31-k5	672	1163	73.1	1012	50.6	950	41.4
E-n22-k4	375	488	30.1	421	12.3	407	8.5
E-n30-k3	534	807	51.1	753	41.0	713	33.5
P-n16-k8	450	495	10.0	450	0.0	456	1.3
P-n19-k2	212	310	46.2	231	9.0	237	11.8
P-n20-k2	216	307	42.1	257	19.0	258	19.4
P-n21-k2	211	300	42.2	220	4.3	216	2.4
P-n22-k2	216	309	43.1	266	23.1	271	25.5
P-n22-k8	603	663	10.0	644	6.8	640	6.1
P-n23-k8	529	636	20.2	553	4.5	553	4.5

with the branch-and-bound algorithm combined with the Nearest Neighbor variable ordering heuristic have also been included for comparison. As expected, the described methodology is able to provide better solutions faster than the branch-and-bound algorithm. The VND-LNS method samples the neighborhoods, while the considered branch-and-bound algorithm performs a complete search. Although this sampling strategy might loose the global optimum, as other heuristic approaches, it is an efficient way of exploring the solutions space. Changing the neighborhood to be explored systematically helps on escaping from local minima and the surrounding valleys, while a branch-and-bound algorithm will spend time checking all solutions around them.

The effects of using the Nearest Neighbor variable ordering heuristic within the described methodology may also be observed in results presented in Table 5.12. Using this heuristic combined with the search methodology permits matching or overcoming the best results obtained by means of the VND-LNS method without using the heuristic for 6 of the 11 instances. Since the VND-LNS methodology is based on a probabilistic algorithm, it is difficult to trace the direct effect of the variable ordering heuristic on the overall behavior. Nevertheless, the described methodology uses repeatedly a repair method that consists on a branch-and-bound algorithm. As explained above, this method is significantly enhanced by using the Nearest Neighbor-based heuristic. For this reason, it is possible to infer that using this heuristic in the repair method will provide, in general, better quality solutions in lower times, and so it can be considered a significant enhancement of the proposed methodology.

Finally, it should be remarked that the results presented in Table 5.12 are far from other state-of-art approaches to the CVRP, both in terms of

quality and computational times. In this sense, the hybrid methodologies introduced in sections 4.2 and 4.3 are able to provide higher quality solutions. Moreover, the Multi-Start VND method reaches such solutions in competitive times. Nevertheless, the CP-based VND-LNS approach provides additional flexibility related to CP characteristics. Because of CP nature, it is expected to behave better as the problem gets more complex with the addition of some side constraints. Therefore, this approach is to be considered as a first step on tackling richer, more complex VRP problems.

VRPTW

The VRPTW is a much more difficult problem than the CVRP. For the VRPTW, even finding a feasible solution is NP-hard [158]. In such problems, the application of CP techniques may require high computational times, even for finding a feasible initial solution or solving the associated subproblems [132]. For this reason, a soft time windows constraints approach has been adopted to tackle the VRPTW. In this variant, time windows for each visit might be violated and a corresponding penalty term is added to the objective function.

In the VRPTW, the main objective is to minimize the total traveled distance, while fulfilling the side constraints, i.e. capacity and time windows constraints. As mentioned, in a soft time windows approach, customers may be visited out of their allowed service times, incurring in a penalization that is added to the objective function. Therefore, minimizing these time windows violations becomes a secondary objective of the problem.

For this test, we have selected the VRPTW benchmark set by Solomon [163]. It contains 56 instances featuring 100 customers which require several vehicles to service them while obeying the side constraints. The maximum number of available vehicles is set to 25, but the number of used vehicle should also be minimized. The problems are distributed in three sets, each of them having two subsets.

The C set features customers arranged in clusters, the R set contains uniformly distributed visits, and the RC set contains a mixture of both sets, that is, some customers are clustered and the rest are uniformly distributed. All sets have two different subsets: classes 1 and 2. The former subsets have tight time windows and, in consequence, few visits may be feasibly assigned to each route. The latter class accepts a higher number of visits per route, since wider time windows are defined. Each subset contains 8 to 12 instances, where different

time windows are defined for each customer. All results for these problems are usually published in real numbers and they only take into account the traveling distance (or time) between customers, excluding the depot. For this reason, we present the results obtained by means of the VND-LNS methodology using real costs. To simplify the search, the algorithm uses integer distances in all the assignments and the values are later converted to reals to provide the final result.

The implemented VND-LNS methodology corresponds to the one introduced in section 4.4.2 combined with the enhancement mechanisms described in section 4.4.3. Since the VRPTW is a more complex problem than the CVRP, the maximum execution time has been raised to 900 seconds for each problem. Although soft time windows are considered, the time windows preprocessing permits adding additional constraints that help on finding a better assignment of customers to routes.

Some authors refer to a simplification of Solomon's instances to test the proposed algorithms. In these cases, only the first 25, 50, or 75 customers from each instance are considered, while keeping the number of available vehicles. These problems, although still NP-hard, permit studying our algorithm's performance before tackling the larger 100-customer instances. In our test, we have selected the 25-customer variants of subset C1 instances. Results obtained by means of the CP-based VND-LNS methodology for these problems are presented in Table 5.13.

As can be observed in Table 5.13, for most 25-customer C1 instances the result obtained by means of the VND-LNS (OS) is over the value of the best known solution (BKS). This is due to the algorithm is stopped after the maximum allowed execution time, before it reaches a minimum. Since time windows violations are included in the objective function, the algorithm may find a bet-

Table 5.13: Results for 25-customers C1 instances.

Problem	**BKS**	**OS**	**# veh. OS / BKS**
c101-25	191.3	177.2	3 / 3
c102-25	190.3	203.2	3 / 3
c103-25	190.3	214.0	3 / 3
c104-25	186.9	206.6	3 / 3
c105-25	191.3	209.7	3 / 3
c106-25	191.3	215.4	3 / 3
c107-25	191.3	215.0	3 / 3
c108-25	191.3	213.2	3 / 3
c109-25	191.3	213.1	3 / 3

ter improving solution by reducing them, while the total traveled distance remains unchanged. Thus, because of the assigned weights in the objective function, the algorithm may focus on minimizing unfeasibility before tackling distance minimization. If enough running time is provided, the algorithm may eventually cancel these terms in the objective function and find a feasible solution for the hard time windows problem. In order to get an appropriate lower bound for the original instance, the penalty terms in the relaxed problem may be adjusted so the algorithm focuses on minimizing the total traveled distance. However, this adjustment is not straightforward, since it could provide a lower bound far below the optimal solution which would not be helpful for solving the original instance.

Table 5.14 presents the results obtained by applying the VND-LNS methodology to Solomon's benchmark instances. As may be observed, most results are far from the best known solution. The reason may be found on the fact that the algorithm is stopped after 900 seconds, not providing enough time to reach a minimum. Nevertheless, some observations may be inferred from these results.

In general, results for class 1 problems are better than those obtained for class 2 instances, except for the R set. The former subsets accept few customers per route because of the narrow time windows defined for each customer. As mentioned, a CP-based algorithm is expected to behave better as the problem gets tighter. In the case of looser problems, domains are larger and therefore the algorithm may try more values for each variable, growing the search tree size. Thus, for class 2 problems, the set of improving solutions found by adjusting the time windows violations is higher than for the class 1 instances. Since the branch-and-bound performs a complete search, it will spend more time exploring these solutions before moving to a different one with a lower traveling cost. For this reason, results for class 2 problems are worse in average, provided the execution time limitation.

It can be observed in the results in Table 5.14 that the VND-LNS method behaves better on those problems with clustered customers. Solutions for sets C and RC are closer to the best known solutions, in average, than those for instances with uniformly distributed customers from the R set. This behavior may be related to the use of the Nearest Neighbor-based variable ordering heuristic, which has a significantly higher influence on problems with clustered visits. Its use in R instances may lead to assign a set of nearby customers to the same route, even though it causes significant time windows violations. For these instances, probably the use of time variables ordering criteria would

Table 5.14: Results for VRPTW Solomon's benchmark instances.

Problem C	**BKS**	**OS**	**# veh. OS / BKS**	**Problem RC**	**BKS**	**OS**	**# veh. OS / BKS**	**Problem R**	**BKS**	**OS**	**# veh. OS / BKS**
c101	828.94	825.16	10 / 10	rc101	1696.94	1791.22	12 / 14	r101	1645.79	3761.48	23 / 19
c102	828.94	983.27	10 / 10	rc102	1554.75	2320.81	12 / 12	r102	1486.12	3338.30	19 / 17
c103	828.06	909.24	10 / 10	rc103	1261.67	2539.77	11 / 11	r103	1292.68	3012.42	15 / 13
c104	824.78	901.68	10 / 10	rc104	1135.48	2250.88	10 / 10	r104	1007.24	3051.70	10 / 9
c105	828.94	891.66	10 / 10	rc105	1629.44	2618.57	13 / 13	r105	1377.11	3384.64	11 / 14
c106	828.94	987.02	10 / 10	rc106	1424.73	1877.27	9 / 11	r106	1251.98	3643.81	11 / 12
c107	828.94	871.98	10 / 10	rc107	1230.48	2304.06	10 / 11	r107	1104.66	2875.14	10 / 10
c108	828.94	874.21	10 / 10	rc108	1139.82	2210.89	9 / 10	r108	960.88	2910.21	8 / 9
c109	828.94	870.17	10 / 10	rc201	1406.91	1779.23	2 / 4	r109	1194.73	3234.23	8 / 11
c201	591.56	1495.40	3 / 3	rc202	1365.65	1770.92	2 / 3	r110	1118.59	3481.27	8 / 10
c202	591.56	1547.48	3 / 3	rc203	1049.62	1739.41	2 / 3	r111	1096.72	3128.40	10 / 10
c203	591.17	1504.32	3 / 3	rc204	798.41	1685.08	2 / 3	r112	982.14	2592.09	8 / 9
c204	590.60	1539.61	3 / 3	rc205	1297.19	1700.29	2 / 4	r201	1252.37	1738.38	3 / 4
c205	588.88	1495.40	3 / 3	rc206	1146.32	1790.53	2 / 3	r202	1191.70	1725.44	3 / 3
c206	588.49	1470.03	3 / 3	rc207	1061.14	1919.76	2 / 3	r203	939.50	1741.13	2 / 3
c207	588.29	1422.37	3 / 3	rc208	828.14	1919.76	2 / 3	r204	825.52	1607.93	2 / 2
c208	588.32	1449.08	3 / 3					r205	994.42	1624.68	2 / 3
								r206	906.14	1677.65	2 / 3
								r207	890.61	1627.87	2 / 2
								r208	726.75	1607.93	2 / 2
								r209	909.16	1608.12	2 / 3
								r210	939.34	1656.66	2 / 3
								r211	885.71	1612.41	2 / 2

perform better if applied without the distance-based ordering heuristics.

As can be inferred from the presented results, the application of the described VND-LNS methodology to the VRPTW requires further efforts to be addressed in the future in order to get a competitive approach. For example, its performance departing from a good initial solution obtained by means of an efficient constructive heuristic is to be analyzed. As mentioned above, the influence of the different defined variable ordering heuristics should also be studied. In addition, several heuristics to control the branch-and-bound repair method may be introduced to improve its performance. However, the VND-LNS method described in section 4.4.2 was aimed to develop an initial approach to complex VRP problems, difficult to tackle from a pure CP perspective. In this sense, the obtained results justify the application of the proposed methodology and provide a starting point for further developments.

Chapter 6

Conclusions and Future Research

In the present thesis, we have introduced three different yet related methodologies to tackle the *Vehicle Routing Problem*. Their structures make use of several metaheuristics frameworks from the *Variable Neighborhood Search* family and combine *Constraint Programming* and *Lagrangian Relaxation* paradigms, as well as probabilistic constructive heuristics.

In the first two hybrid approaches, presented in sections 4.2 and 4.3, the *Capacitated VRP* problem has been decomposed into two separate subproblems. The first one is aimed to assign customers to vehicles in terms of capacity, while the second is used to optimize the corresponding routes. This approach allows reducing the computational time, since the problems to be solved are far less complex than the original CVRP.

In both subproblems, two well-known paradigms aimed to solve combinatorial problems, CP and LR, have been applied. The used LR-based method permits reducing the calculation times due to its improved convergence with respect to the Subgradient Optimization classical algorithm. It also provides optimal routes when the number of customers is relatively small, as it is for all CVRP benchmark instances. Combining these characteristics with the adopted decomposition allows reducing the total computation time. On the one hand, the selected decomposition makes LR only necessary to recalculate two routes at each iteration. On the other hand, the LR-based method is faster and simpler than other routing post-optimization processes, since no intra-route movements are to be defined. At the same time, the adopted CP approach to customers' allocation has demonstrated to be efficient both for solving the capacity subproblem and for checking feasibility at runtime.

The mentioned paradigms have been embedded into a general VNS metaheuristic framework in the first presented methodology, introduced in section 4.2. In this approach, the described decomposition is combined with CP and LR to get a quick initial solution for the CVRP, even for larger instances. Although solutions quality is usually low, it is rapidly improved after the first iterations. This methodology has provided state-of-art results, in terms of quality, for most CVRP benchmark instances. It has been able to reach —or, in some cases, overcome— best-known or optimal solutions in few iterations. Otherwise, the gap between the obtained solution and the best-known one is usually low (under 1 % for most benchmark instances).

Although this hybrid methodology has demonstrated to be effective for solving the CVRP, the required computational time is not competitive with other state-of-art heuristics. Some mechanisms have been introduced to enhance its performance. First, some strategies have been implemented to reduce the computational complexity of neighborhoods exploration. Since only a small part of the solution is changed any time a feasible move is applied, the algorithm may focus on exploring this region at next iteration. This approach does not modify the solutions space. In addition, if all feasible moves are recorded, the independent moves may be applied at once in order to improve algorithm's convergence to a local optimum. Results show the desired reduction of the required computational time after implementing these strategies. On the other hand, departing from a good initial solution may yield to better results faster than the implemented method. For this reason, we have studied the influence of initial solution's quality. In this test, the obtained solutions and computational times show that the randomized version of the *Clarke and Wright Savings* heuristic provides a useful mechanism to get the initial solution for this hybrid methodology.

The RCWS heuristic has been used to feed the multi-start schema presented as the second methodology, described in section 4.3. This approach consists of a *Variable Neighborhood Descent* framework which embeds CP and LR to cope with the described CVRP decomposition, while the RCWS algorithm is used to provide high-quality initial solutions. This schema has been parallelized and a multi-start approach has been adopted.

Because of the techniques we have used, it is remarkable that both methodologies are robust, due to the light requirements for fine tuning. The tailored LR-based algorithm does not require any specific adjustment since all the convergence parameters are self-tuned. The CP-based subproblem depends just on the quality of the defined constraint model to properly describe the feasi-

ble solutions. The RCWS does not require any adaptation either. Only the movements used in the VND and VNS frameworks could require a different prioritization depending on the problem being solved in order to get a better solution quality.

The efficiency of the Multi-Start VND approach is supported by the obtained results. As discussed, this methodology is able to match the best known solutions for CVRP benchmark problems of different sizes in reasonable computational times. The provided comparison proves that its efficiency is similar to other state-of-art metaheuristics, both in terms of computational time and solutions quality.

It should be noticed that, due to their modular design, the hybrid VNS and the Multi-Start VND methodologies provide a moderate flexibility to be adapted to solve other VRP with additional constraints. Both approaches benefit from the CP capabilities to model different operational constraints. This constraints are present in most of the real application cases and, in general, affect to the allocation decisions. CP facilitates the representation of these allocation constraints without requiring any specific action on the solving method. Nevertheless, facing other relevant VRP variants, such as those involving pick-up and delivery locations or time windows, would imply the modification of the LR-based method and the implementation of new neighborhoods in the VND and VNS metaheuristics. The RCWS algorithm should be also adapted and would require a proof of its efficiency in the new scenario.

In order to cope with such VRP variants, a CP-based methodology has been introduced in section 4.4. First, an extension of the CP model for the VRP introduced by Kilby and Shaw [99] has been described. This formulation has been implemented as a first step for defining a VRP library, providing a unified model able to tackle different VRP classes. This model can be easily extended, since CP permits adding side constraints that would be difficult to implement using other approaches.

For problems like the VRP, the application of CP exact methods to explore the search space is limited and the definition of heuristic methods becomes mandatory. The heuristic methodology described in section 4.4.2 combines CP search methods within VND and *Large Neighborhood Search* frameworks. After an initial solution is provided, the two operators *SPLIT* and *RPOP* are iteratively used to partially destroy and re-optimize the current solution in a LNS fashion. A VND metaheuristic guides this search process. Some strategies have been defined to enhance the performance of this CP-based VND-LNS methodology in section 4.4.3.

The VND-LNS methodology has been tested on several CVRP and VRPTW benchmark instances. Due to the CP techniques used to re-optimize the solutions, the required computational times are high and far from state-of-art methods. As may be observed in the reported results, the presented methodology is only able to tackle small CVRP instances, as well as soft time windows instances in the VRPTW case. However, the main goal was to establish an initial approach to the VRP using CP techniques, suitable to be extended to richer VRP variants and enhanced by the application of different heuristic methods. Hence, the presented methodology constitutes the starting point for further developments, achieving thus the pursued objective.

6.1 Contributions of this work

The contributions of this thesis, discussed in the previous section, can be summarized in the following list:

- The CVRP has been decomposed and formulated according to CP and LR paradigms. These paradigms have been used to solve the decomposed problem efficiently.

- The CP and LR methods have been embedded in a VNS metaheuristic framework to obtain better quality solutions and to escape from local minima. Two different implementations of this algorithm's family, the general VNS and the VND, have been developed.

- The local search efficiency has been improved by choosing which partial solutions should be checked, developing the corresponding data structures to manage this information, and speeding up algorithm's convergence to a local optimum because of the application of several independent local moves.

- Several strategies to obtain the initial solution for the algorithm have been tested. Their influence in the quality of the final solution has also been analyzed. One of these strategies is a randomized version of the CWS heuristic, capable of getting nearly optimal initial solutions in very low times.

- The hybrid VND approach has been parallelized in a multi-start schema. The RCWS heuristic is used to feed this parallelized algorithm. This approach allows reducing the computational times significantly while improving solutions' quality, becoming a competitive alternative to other state-of-art methods.

- The CP model presented by Kilby and Shaw [99] has been extended, considering additional constraints in order to cope with more complex VRP variants.

- A CP search method based on the VND and LNS heuristics has been implemented. Two different strategies have been defined to partially destroy the solutions, while a CP exact method is used for re-optimization.

- Two specific variable ordering heuristics have been implemented to improve the CP search method performance when tackling the CVRP and VRPTW problems.

Publications

The publications related in this section are to be considered as part of the contributions of this work.

The work presented in this thesis has been partially published in the following journal articles:

- D. Guimarans, R. Herrero, D. Riera, A.A. Juan, and J.J. Ramos. Combining probabilistic algorithms, Constraint Programming and Lagrangian Relaxation to solve the Vehicle Routing Problem. *Annals of Mathematics and Artificial Inetlligence*, 62(3): 299–315, 2011.

- D. Guimarans, R. Herrero, J.J. Ramos, and S. Padrón. Solving Vehicle Routing Problems using Constraint Programming and Lagrangean Relaxation in a Metaheuristics Framework. *International Journal of Information Systems and Supply Chain Management*, 4(2): 61–81, 2011.

Some parts of this work have also been presented in several international conferences and published in the following related articles:

- R. Herrero, J.J. Ramos, and D. Guimarans. Lagrangean Metaheuristic for the Travelling Salesman Problem. *Operational Research Conference 52 (OR-52)*. Surrey, UK. September 2010.

- J.J. Ramos, S. Padrón, L. Guillén, M.A. Piera, D. Guimarans, and R. Herrero. Intelligent platform for sustainable routing. *XV Summer School Francesco Turco*. Porto Giardino, Italy. September 2010.

- R. Herrero, D. Guimarans, J.J. Ramos, and S. Padrón. A Variable Neighbourhood Search combining Constraint Programming and Lagrangean Relaxation for solving routing problems. *Summer Computer Simulation Conference (SCSC)*. Ottawa, Canada. July 2010.

- D. Guimarans, R. Herrero, D. Riera, A.A. Juan, and J.J. Ramos. Combining Constraint Programming, Lagrangian Relaxation and probabilistic algorithms to solve the Vehicle Routing Problem. *RCRA International Workshop, CP-AI-OR*. Bologna, Italy. June 2010.

- R. Herrero, D. Guimarans, and J.J. Ramos. Solving the Travelling Salesman Problem with Time Windows by Lagrangean Relaxation. *European Modeling and Simulation Symposium (EMSS)*. Tenerife, Spain. September, 2009.

- D. Riera, A.A. Juan, D. Guimarans, and E. Pagans. A Constraint Programming-based library for the Vehicle Routing Problem. *European Modeling and Simulation Symposium (EMSS)*. Tenerife, Spain. September, 2009.

- D. Guimarans, J.J. Ramos, M.G. Wallace, and D. Riera. A hybrid Constraint Programming / Local Seach approach to the Pick-up and Delivery Problem with Time Windows. *European Modeling and Simulation Symposium (EMSS)*. Tenerife, Spain. September, 2009.

Finally, the hybrid VNS methodology described in section 4.2 has been registered with the following software license:

- D. Guimarans, R. Herrero, J.J. Ramos, and M.A. Piera. *ITSLogiSim Optimization Suite 1.0*. Registered at *Universitat Autònoma de Barcelona*, 30^{th} June 2010.

Currently, the company *Digital Aeronautics Engineering Services* is using this software license under a commercial agreement. Combined with a clustering algorithm, it is being applied to solve problems up to 22,000 customers and more than 5,000 planned routes.

6.2 Future Research

Several fields for future research are open for the three methodologies presented in this thesis. We present a short review of some of these lines, separating them according to the presented methodologies. In any case, since the three methodologies are strongly related, most of the proposed lines are to be considered transverse.

Regarding the hybrid VNS and the Multi-Start VND methodologies, the following lines for future research have been identified:

- Different VNS schemes are to be studied, such as the *Variable Neighborhood Decomposition Search* (see Section 3.3.4), whose shaking process can be improved by embedding CP techniques. The definition of new VNS approaches may also be considered. In this context, two algorithms have already been defined, although further efforts need to be addressed in this direction. The first approach hybridizes the general VNS algorithm with a Simulated Annealing solutions' acceptance strategy. Thus, at first iterations the algorithm is more likely to accept some non-improving solutions to escape from local minima and diversify the search. This probability decreases as the exploration evolves, intensifying the search. The second approach assigns probabilities to each movement and applies a roulette wheel selection to decide which neighborhood to explore at each iteration. The weights are adjusted in runtime according to algorithm's performance exploring each neighborhood for the problem at hand, becoming an adaptive approach.

- New movements are to be defined, as well as designing heuristics to efficiently explore the corresponding neighborhoods. The described strategies for reducing the computational complexity of local search are to be evaluated and adapted to these new neighborhoods.

- The methodologies are to be adapted to different VRP variants, especially those including time windows and pick-up and delivery side constraints. To achieve this goal, the LR-based method is to be adapted

so it can handle the additional constraints. In addition, new movements are to be defined to explore the feasible regions of these problems' solution space. Eventually, the RCWS algorithm should also be adapted, or substituted by other efficient approaches for each VRP variant.

As for the CP-based VND-LNS methodology, the following lines for future research have been identified:

- The presented approach is to be combined with the two previous hybrid methodologies for checking solutions' feasibility and re-optimizing small parts of the VRP at hand. This approach would permit the hybrid methodology to guide the search, while the CP-based method would be used as a solver. For example, the relocate movement could be redefined as a mechanism to choose a pivot customer and subsequently apply the CP-based approach. This approach would modify a larger part of the current solution and so the algorithm's convergence to a local optimum would be improved. Moreover, since CP handles side constraints in a natural manner, it would permit tackling VRP problems with additional restrictions with little adjustments in the algorithm.

- New destroy methods are to be defined for the VRP. Some of these destroy strategies could include, but are not limited to, unfixing one complete route, unassigning pairs of neighboring routes, choosing customers to remove according to their time windows violations, or choosing visits to remove following the relatedness criterion defined by Shaw [161].

- The repair methods are to be enhanced by using heuristics on the search. Furthermore, tailored CP search methods may be studied and applied to improve algorithm's performance for specific problems.

- The CP model is to be extended to other VRP variants, such as the PDTW or the *Rich VRP*. In the first case, the model may be easily adapted by adding a corresponding constraint on the time variable for each pick-up/delivery pair. Nevertheless, propagation over time variables may not be efficient and it would be preferable to add a new sequence variable so propagation is improved. Similar modifications are to be studied in order to tackle more complex VRP problems.

Finally, the adaptation of the three methodologies presented in this thesis to other combinatorial problems is to be considered.

Bibliography

[1] Branch and cut - http://www.branchandcut.org, 2003.

[2] *ILOG Solver 6.3 Reference Manual*. ILOG, France, 2006.

[3] R.K. Ahuja, O. Ergun, J.B. Orlin, and A. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.

[4] T.J. Ai and V. Kachitvichyanukul. A particle swarm optimisation for vehicle routing problem with time windows. *International Journal of Operational Research*, 6(4):519–537, 2009.

[5] T.J. Ai and V. Kachitvichyanukul. Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem. *Computers and Industrial Engineering*, 56(1):380–387, 2009.

[6] E. Alba and B. Dorronsoro. A hybrid cellular genetic algorithm for the capacitated vehicle routing problem. In A. Abraham, C. Grosan, and W. Pedrycz, editors, *Engineering Evolutionary Intelligent Systems*, volume 82 of *Studies in Computational Intelligence*, pages 379–422. Springer-Verlag, Berlin Heidelberg, 2008.

[7] I. Altinel and T. Öncan. A new enhancement of the clarke and wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 56:954–961, 2005.

[8] K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[9] K. Apt and M. Wallace. *Constraint Logic Programming Using ECLiPSe*. Cambridge University Press, 2007.

[10] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical report 1 RR949-M, ARTEMIS-IMAG, Grenoble, France, 1995.

[11] B. De Backer, V. Furnon, P. Kilby, P. Prosser, and P. Shaw. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6:501–523, 2000.

[12] B. De Backer, V. Furnon, P. Prosser, P. Kilby, and P. Shaw. Local search in constraint programming: application to the vehicle routing problem. In *Proceedings of the CP-97 Workshop Industrial Constraint-Directed Scheduling*, pages 1–15, Schloss Hagenberg, Austria, 1997.

[13] P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research C*, 5:109–122, 1997.

[14] N. Balakrishnan. Simple heuristics for the vehicle routing problem with soft time windows. *Journal of the Operational Research Society*, 44:279–287, 1993.

[15] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming Series A*, 115(2):351–385, 2008.

[16] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52:723–738, 2004.

[17] R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

[18] R. Baldacci, A. Mingozzi, and R. Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, doi:10.1016/j.ejor.2011.07.037, 2011.

[19] R. Baldacci, P. Toth, and D. Vigo. Exact algorithms for routing problems under vehicle capacity constraints. *Annals of Operations Research*, 175:213–245, 2010.

[20] M. Balinski and R. Quandt. On an integer program for a delivery problem. *Operations Research*, 12:300–304, 1964.

[21] N. Beldiceanu, M. Carlsson, and J.X. Rampon. Global constraint catalog. Technical report 2005:08, Swedish Institute of Computer Science (SICS), Kista, Sweden, 2005.

[22] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of ACM*, 1:61–63, 1962.

[23] R. Bent and P. van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

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

[25] J. Berger and M. Barkaoui. A hybrid genetic algorithm for the capacitated vehicle routing problem. In *Proceedings of the International Genetic and Evolutionary Computation Conference*, pages 646–656, Chicago, IL, 2003. Springer-Verlag.

[26] J. Berger, M. Barkaoui, and O. Bräysy. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *Information Systems and Operational Research*, 41:179–194, 2003.

[27] J. Berger, M. Salois, and R. Begin. A hybrid genetic algorithm for the vehicle routing problem with time windows. *Lecture Notes on Artificial Intelligence*, 1418:114–127, 1998.

[28] C. Bessiere. *Constraint Propagation*, volume Handbook of Constraint Programming, chapter 3, pages 29–83. Elsevier, 2006.

[29] J.L. Blanton and R.L. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459, San Francisco, CA, 1993. Morgan Kaufmann Publishing.

[30] M. Boschetti and V. Maniezzo. Benders decomposition, lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15:283–312, 2009.

[31] A. Le Bouthillier and T.G. Crainic. A cooperative parallel metaheuristic for vehicle routing with time windows. *Computers and Operations Research*, 32:1685–1708, 2005.

[32] J. Brandão. *Metaheuristics - Advances and trends in local search paradigms for optimization*, chapter Metaheuristic for the vehicle routing problem with time windows, pages 19–36. Kluwer Academic, Boston, MA, 1999.

[33] O. Bräysy. A reactive variable neighborhood search for the vehicle routing problem with time windows. *INFORMS Journal of Computing*, 15:347–368, 2003.

[34] O. Bräysy, W. Dullaert, and M. Gendreau. Evolutionary algorithms for the vehicle routing problem with time windows. *Journal of Heuristics*, 10:587–611, 2004.

[35] O. Bräysy and M. Gendreau. Vehicle routing problems with time windows, part i: route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[36] O. Bräysy and M. Gendreau. Vehicle routing problems with time windows, part ii: metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[37] O. Bräysy, G. Hasle, and W. Dullaert. A multi-start local search algorithm for the vehicle routing problem with time windows. *European Journal of Operational Research*, 159:586–605, 2004.

[38] G. Buxey. The vehicle scheduling problem and monte carlo simulation. *Journal of Operational Research*, 30:563–573, 1979.

[39] W.B. Carlton. *A tabu search approach to the general vehicle routing problem.* PhD thesis, University of Texas, Austin, Texas, 1995.

[40] A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers and Operations Research*, 33:2972–2990, 2006.

[41] C.-H. Chen and C.J. Ting. An improved ant colony system algorithm for the vehicle routing problem. *Journal of the Chinese Institute of Industrial Engineers*, 23(2):115–126, 2006.

[42] W.C. Chiang and R. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal of Computing*, 9:417–430, 1997.

[43] W.C. Chiang and R.A. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63:3–27, 1996.

[44] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In *Combinatorial Optimization*, pages 315–338. Wiley, 1979.

[45] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282, 1981.

[46] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivering points. *Operations Research*, 12:568–581, 1964.

[47] S.A. Cook. The complexity of theorem-proving procedures. In *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[48] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (darp): variants, modeling issues and algorithms. *4OR: A Quarterly Journal of Operations Research*, 1(2):89–101, 2003.

[49] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

[50] E. Danna and C. Le Pape. *Accelerating branch-and-price with local search: a case study on the vehicle routing problem with time windows*, chapter Column generation, pages 90–130. Springer-Verlag, Berlin Heidelberg, 2005.

[51] G. Dantzig and J. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[52] P. Fernández de Córdoba, L. García, A. Mayado, and J. Sanchís. A real delivery problem dealt with monte carlo techniques. *TOP*, 8:57–71, 2000.

[53] H.C. Brandão de Oliveira and G.C. Vasconcelos. A hybrid search method for the vehicle routing problem with time windows. *Annals of Operations Research*, 180:125–144, 2010.

[54] G. Desaulniers, F. Lessard, and A. Hadjar. Tabu search, partial elementary, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, 2008.

[55] M. Desrochers, J. Desrosiers, and M.M. Solomon. A new optimization algorithm for the vehicle-routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

[56] M. Dorigo and T. Stützle. *Handbook of Metaheuristics*, chapter Ant Colony Optimization: overview and recent advances, pages 227–264. Kluwer Academic, Boston, MA, 2003.

[57] W. Dullaert. Impact of relative route length on the choice of time insertion criteria for insertion heuristics for the vehicle routing problem with time windows. In B. Maurizio, editor, *Proceedings of Rome Jubilee 2000 Conference on Improving Knowledge Tools Transportation Logistics*, pages 153–156, Rome, Italy, 2000.

[58] W. Dullaert and O. Bräysy. Routing with relatively few customers per route. *TOP*, 11:325–336, 2003.

[59] J. Faulin and A.A. Juan. The algacea-1 method for the capacitated vehicle routing problem. *International Transactions in Operational Research*, 15:1–23, 2008.

[60] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44:216–229, 2004.

[61] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[62] P. Festa and M. Resende. An annotated bibliography of grasp - part i: Algorithms. *International Transactions in Operational Research*, 16:1–24, 2009.

[63] G. Finke, A. Claus, and E. Gunn. A two-commodity network flow approach to the traveling salesman problem. *Numerantium*, 41:167–178, 1984.

[64] M. Fisher. The lagrangean relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.

[65] L. Fortnow. The status of the p versus np problem. *Communications of the ACM*, 52(9):78–86, 2009.

[66] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming Series A*, 106:491–511, 2006.

[67] L.M. Gambardella, E. Taillard, and G. Agazzi. *New Ideas in Optimization*, chapter MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows, pages 63–76. McGraw-Hill, London, UK, 1999.

[68] B.-L. Garcia, J.-Y. Potvin, and J.-M. Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers and Operations Research*, 21:1025–1033, 1994.

[69] H. Gehring and J. Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In K. Miettinen, M. Mäkelä, and J. Toivanen, editors, *Proceedings of EUROGEN99*, pages 57–64, Finland, 1999. Univeristy of Jyväskylä.

[70] H. Gehring and J. Homberger. Parallelization of a two-phase metaheuristic for routing problems with time windows. *Asia-Pacific Journal of Operations Research*, 18:35–47, 2001.

[71] M. Gendreau. *Handbook of Metaheuristics*, chapter An introduction to Tabu Search, pages 37–54. Kluwer Academic, Boston, MA, 2003.

[72] M. Gendreau, A. Hertz, and G. Laporte. A new insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1093, 1992.

[73] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[74] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46:330–335, 1998.

[75] B. Gillett and L.R. Miller. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

[76] F. Glover. *Computer Science and Operations Research: New Developments in Their Interfaces*, chapter New ejection chain and alternating path methods for traveling salesman problems, pages 449–509. Pergamon Press, Oxford, UK, 1992.

[77] M. Guignard. Lagrangean relaxation. *TOP*, 11(2):151–228, 2003.

[78] D. Guimarans, R. Herrero, J.J. Ramos, and S. Padrón. Solving vehicle routing problems using constraint programming and lagrangian relaxation in a metaheuristics framework. *International Journal of Information Systems and Supply Chain Management*, 4(2):61–81, 2011.

[79] P. Hansen and N. Mladenovic. A tutorial on variable neighbourhood search. Technical report G-2003-46, Les Cahiers du GERAD, HEC Montreal and GERAD, Montreal, Quebec, Canada, 2003.

[80] P. Hansen, N. Mladenovic, and J.A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175:367–407, 2010.

[81] G. Hasle and O. Kloster. *Geometric Modelling, Numerical Simulation, and Optimization*, chapter Industrial vehicle routing, pages 397–435. Springer-Verlag, Berlin Heidelberg, 2007.

[82] M. Held and R.M. Karp. The travelling salesman problem and minimum spanning trees: part ii. *Mathematical Programming*, 1:6–25, 1971.

[83] M. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[84] R. Herrero, J.J. Ramos, and D. Guimarans. Lagrangean metaheuristic for the travelling salesman problem. In *Extended Abstracts of Operational Research Conference 52 (OR-52)*, Royal Holloway, University of London, 2010. Operational Research Society.

[85] J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *Information Systems and Operational Research*, 37:297–318, 1999.

[86] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162:220–238, 2005.

[87] G. Ioannou, M. Kritikos, and G. Prastacos. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52:523–537, 2001.

[88] S. Irnich and D. Villeneuve. The shortest-path problem with resource constraints and k-cycle elimination for k>=3. *INFORMS Journal of Computing*, 18(3):391–406, 2006.

[89] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

[90] L. Jourdan, M. Basseur, and E.G. Talbi. Hybridizing exact methods and metaheuristics: a taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.

[91] A.A. Juan, J. Faulin, J. Jorba, S. Grasman, and B. Barrios. Sr-2: a hybrid intelligent algorithm for the vehicle routing problem. In *Proceedings of the 8th International Conference on Hybrid Intelligent Systems*, pages 78–83, Barcelona, 2008. IEEE Computer Society.

[92] A.A. Juan, J. Faulin, J. Jorba, D. Riera, D. Masip, and B. Barrios. On the use of monte carlo simulation, cache and splitting techniques to improve the clarke and wright savings heuristics. *Journal of the Operational Research Society*, 62:1085–1097, 2011.

[93] A.A. Juan, J. Faulin, R. Ruiz, B. Barrios, and S. Caballe. The sr-gcws hybrid algorithm for solving the capacitated vehicle routing problem. *Applied Soft Computing*, 10(1):215–224, 2010.

[94] A.A. Juan, J. Faulin, R. Ruiz, B. Barrios, M. Gilibert, and X. Vilajosana. *Operations Research and Cyber-Infrastructure*, volume 47 of *Operations Research / Computer Science Interfaces Series*, chapter Using oriented random search to provide a set of alternative solutions to the capacitated vehicle routing problem, pages 331–346. Springer-Verlag, Berlin Heidelberg, 2009.

[95] S. Jung and B.-R. Moon. A hybrid genetic algorithm for the vehicle routing problem with time windows. In *Proceedings of the Genetic and Evolutionary Computing Conference*, pages 1309–1316, San Francisco, CA, 2002. Morgan Kaufmann Publishing.

[96] B. Kallehauge. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers and Operations Research*, 35(7):2307–2330, 2008.

[97] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, 1995. IEEE Computer Society.

[98] P. Kilby, P. Prosser, and P. Shaw. *Metaheuristics - Advances and trends in local search paradigms for optimization*, chapter Guided local search for the vehicle routing problem with time windows, pages 473–486. Kluwer Academic, Boston, MA, 1999.

[99] P. Kilby and P. Shaw. *Vehicle routing*, volume Handbook of Constraint Programming, chapter 23, pages 801–836. Elsevier, 2006.

[100] H. Wee Kit, J. Chin, and A. Lim. A hybrid search algorithm for the vehicle routing problem with time windows. *International Journal of Artificial Intelligence Tools*, 10:431–449, 2001.

[101] N. Kohl, J. Desrosiers, O.B.G. Madsen, M.M. Solomon, and F. Sounis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[102] G.A. Kontoravdis and J.F. Bard. A grasp for the vehicle routing problem with time windows. *INFORMS Journal of Computing*, 7:10–23, 1995.

[103] G. Laporte. The traveling salesman problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1992.

[104] G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54:811–819, 2007.

[105] G. Laporte, Y. Nobert, and M. Desrochers. Optimal routing under capacity and distance restrictions. *Operations Research*, 33:1058–1073, 1985.

[106] H.C. Lau, Y.F. Lim, and Q. Liu. Diversification of neighborhood via constraint-based local search and its application to vrptw. In *Proceedings of CP-AI-OR 2001 Workshop*, Kent, UK, 2001.

[107] H.C. Lau, M. Sim, and K.M. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148:559–568, 2003.

[108] A. Law. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 2007.

[109] J.K. Lenstra and A.H.G.R. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[110] H. Li, A. Lim, and J. Huang. Local search with annealing-like restarts to solve the vrptw. *European Journal of Operational Research*, 150:115–127, 2003.

[111] A. Lim and X. Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal of Computing*, 19:443–457, 2007.

[112] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[113] S. Lin, Z. Lee, K. Ying, and C. Lee. Applying hybrid metaheuristics for capacitated vehicle routing problem. *Expert Systems and Applications*, 2(36):1505–1512, 2009.

[114] C.Y. Liong, I. Wan Rosmanira, O. Khairuddin, and M. Zirour. Vehicle routing problem: models and solutions. *Journal of Quality Measurement and Analysis*, 4(1):205–218, 2008.

[115] J. Lysgaard. Reachability cuts for the vehicle routing problem with time windows. *European Journal of Operational Research*, 175(1):210–223, 2006.

[116] J. Lysgaard, A.N. Letchford, and R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming Series A*, 100:423–445, 2004.

[117] S. Masrom, S.Z.Z. Abidin, A.M. Nasir, and A.S.A. Rahman. Hybrid particle swarm optimization for vehicle routing problem with time windows. In *Proceedings of the International Conference on Recent Researches in Computational Techniques, Non-Linear Systems and Control*, pages 142–147, Romania, 2011.

[118] D. Mester. An evolutionary strategies algorithm for large scale vehicle routing problem with capacitate and time window restrictions. Working paper, Institute of Evolution, University of Haifa, Israel, 2002.

[119] D. Mester and O. Bräysy. Active-guided evolution strategies for the large-scale capacitated vehicle routing problems. *Computers and Operations Research*, 34:2964–2975, 2007.

[120] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of ACM*, 7(4):326–329, 1960.

[121] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[122] L. Moccia, J.-F. Cordeau, and G. Laporte. An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society*, doi:10.1057/jors.2011.25, 2011.

[123] U. Montanary. Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[124] P. Moscato and C. Cotta. *Handbook of Metaheuristics*, chapter A modern introduction to Memetic Algorithms, pages 141–183. Kluwer Academic, Boston, MA, 2003.

[125] D. Naddef and G. Rinaldi. *The Vehicle Routing Problem*, volume 9 of *Monographs on Discrete Mathematics and Applications*, chapter Branch-and-cut algorithms for the capacitated VRP, pages 53–81. SIAM, Philadelphia, PA, 2002.

[126] Y. Nagata. Edge assembly crossover for the capacitated vehicle routing problem. In C. Cotta and J. van Hemert, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 142–153, Berlin Heidelberg, 2007. Springer-Verlag.

[127] Y. Nagata and O. Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5):333–338, 2009.

[128] Y. Nagata, O. Bräysy, and W. Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers and Operations Research*, 37:724–737, 2010.

[129] A.G. Nikolaev and S.H. Jacobson. *Handbook of Metaheuristics*, chapter Simulated Annealing, pages 1–40. Kluwer Academic, Boston, MA, 2003.

[130] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41:421–452, 1993.

[131] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In M. Wallace, editor, *10th International Conference on Principles and Practice of Constraint Programming (CP-04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 468–481, Berlin Heidelberg, 2004. Springer-Verlag.

[132] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An exact contraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32:12–29, 1998.

[133] G. Pesant, M. Gendreau, and J.-M. Rousseau. Genius-cp: a generic vehicle routing algorithm. In *3rd International Conference on Principles and Practice of Constraint Programming (CP-97)*, Lecture Notes in Computer Science, pages 420–434, New York, 1997. Springer-Verlag.

[134] D. Pisinger and S. Ropke. *Handbook of Metaheuristics*, chapter Large Neighborhood Search, pages 399–420. Kluwer Academic, Boston, MA, 2003.

[135] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34(8):2403–2435, 2007.

[136] J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows, part ii: Genetic search. *INFORMS Journal of Computing*, 8:165–172, 1996.

[137] J.-Y. Potvin, T. Kervahut, B.L. Garcia, and J.-M. Rousseau. The vehicle routing problem with time windows, part i: Tabu search. *INFORMS Journal of Computing*, 8:157–164, 1996.

[138] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66:331–340, 1993.

[139] E. Prescott-Gagnon, G. Desaulniers, and L.M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204, 2009.

[140] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31:1985–2002, 2004.

[141] P. Prosser and P. Shaw. Study of greedy search with multiple improvement heuristics for vehicle routing problems. Working paper, University of Strathclyde, Glasgow, Scotland, 1996.

[142] C. Qi and Y. Sun. An improved ant colony algorithm for vrptw. In *Proccedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 455–458, Wuhan, China, 2008. IEEE Computer Society.

[143] T. Ralphs, L. Kopman, W. Pulleyblank, and L. Trotter. On the capacitated vehicle routing problem. *Mathematical Programming Series B*, 94:343–359, 2003.

[144] C. Reeves. *Handbook of Metaheuristics*, chapter Genetic Algorithms, pages 55–82. Kluwer Academic, Boston, MA, 2003.

[145] G. Reinelt. *The Travelling Salesman: Computational solutions for TSP Applications*. Springer-Verlag, Berlin Heidelberg, 1994.

[146] G. Reinelt. Tsplib - http://comopt.ifi.uni-heidelberg.de/software/tsplib95/, 2008.

[147] M. Resende. *Tutorials in Operations Research*, chapter Metaheuristic hybridization with Greedy Randomized Adaptive Search Procedures, pages 295–319. INFORMS, 2008.

[148] D. Riera, A.A. Juan, D. Guimarans, and E. Pagans. A constraint programming-based library for the vehicle routing problem. In *Proceedings of the European Modeling and Simulation Symposium*, pages 105–110, Tenerife, Spain, 2009.

[149] G. Righini and M. Salani. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

[150] Y. Rochat and E. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995.

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

[152] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171:750–775, 2006.

[153] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. Technical report ACT-AI-222-89, MCC, Austin, Texas, 1989.

[154] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[155] L.M. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8:43–58, 2002.

[156] R. Russell. An effective heuristic for the m-tour traveling salesman problem with some side conditions. *Operations Research*, 25:517–524, 1977.

[157] R. Russell. Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29:156–166, 1995.

[158] M. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[159] J. Schulze and T. Fahle. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86:585–607, 1999.

[160] P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. Working paper, University of Strathclyde, Glasgow, Scotland.

[161] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *4th International Conference on Principles and Practice of Constraint Programming (CP-98)*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431, Berlin Heidelberg, 1998. Springer-Verlag.

[162] M.M. Solomon. On the worst case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16:161–174, 1986.

[163] M.M. Solomon. Algorithms for the vehicle routing problem and scheduling problems with time window constraints. *Operations Research*, 35(254-265), 1987.

[164] R.K. Yang Soo and Y.H. Tay. A survey on the progress of research on vehicle routing problem with time window constraints. In *Symposium on Progress in Information and Communication Technology*, pages 142–146, 2009.

[165] E. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[166] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, pages 170–186, 31.

[167] E.G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[168] N. Tamura. Cream version 1.2 programmers guide. Available online: http://bach.istc.kobe-u.ac.jp/cream/ (last accessed 5th Jan. 2012), 2004.

[169] K.C. Tan, L.H. Lee, and K. Ou. Hybrid genetic algorithms in solving vehicle routing problems with time window constraints. *Asia-Pacific Journal of Operations Research*, 18:121–130, 2001.

[170] K.C. Tan, L.H. Lee, and K.Q. Zhu. Heuristic methods for vehicle routing problem with time windows. In *Proceedings of the 6th International Symposium on Artificial Intelligence and Mathematics*, Ft. Lauderdale, FL, 2000.

[171] C. Tarantilis and C. Kiranoudis. Boneroute: an adaptative memory-based method for effective fleet management. *Annals of Operations Research*, 115:227–241, 2002.

[172] S.R. Thangiah. *Application Handbook of Genetic Algorithms: New Frontiers*, volume II, chapter Vehicle routing with time windows using genetic algorithms, pages 253–277. CRC Press, Boca Raton, FL, 1995.

[173] S.R. Thangiah, K.E. Nygard, and P.L. Juell. Gideon: a genetic algorithm system for vehicle routing with time windows. In *Proceedings of the 7th IEEE Conference on Artificial Intelligence and Applications*, pages 322–328, Los Alamitos, CA, 1991. IEEE Computer Society Press.

[174] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, 2002.

[175] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal of Computing*, 15:333–346, 2003.

[176] P. van Hentenryck and L. Michel. *Constraint-based local search*. The MIT Press, 2009.

[177] C. Voudouris and E. Tsang. *Handbook of Metaheuristics*, chapter Guided local search, pages 185–218. Kluwer Academic, Boston, MA, 2003.

[178] L.A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, 1998.

[179] A. Yurtkuran and E. Emel. A new hybrid electromagnetism-like algorithm for capacitated routing problems. *Expert Systems and Applications*, 37(4):3427–3433, 2010.

[180] E.E. Zachariadis and C.T. Kiranoudis. A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers and Operations Research*, 37:2089–2105, 2010.

[181] R. Zamani and S.K. Lau. Embedding learning capability in lagrangean relaxation: an application to the travelling salesman problem. *European Journal of Operational Research*, 201:82–88, 2010.

[182] Y. Zhong and X. Pan. A hybrid optimization solution to vrptw based on simulated annealing. In *Proceedings of the IEEE International Conference on Automation and Logistics*, pages 3113–3117, Jinan, China, 2007. IEEE Computer Society.