UNIVERSITY OF MACEDONIA

MASTER OF SCIENCE

DEPARTMENT OF APPLIED INFORMATICS

Cooperative Vs Non-Cooperative Parallelization Strategies for Variable Neighborhood Search metaheuristic: A case study on the Capacitated Vehicle Routing Problem

An MSc thesis

by

Panagiotis Kalatzantonakis

B.S. Computer Science, HOU, 2016

Submitted to the University of Macedonia School of Information Sciences and the department of Applied Informatics in partial fulfillment of the requirements for the degree of

Master of Science in Applied Informatics

Thessaloniki, 30 Oct 2019

COOPERATIVE VS NON-COOPERATIVE PARALLELIZATION STRATEGIES FOR VNS: A CASE STUDY ON THE CAPACITATED VEHICLE ROUTING PROBLEM

Panos Kalatzantonakis

B.S. Computer Science, HOU, 2016

A thesis submitted to the University of Macedonia School of Information Sciences
and the department of Applied Informatics
in partial fulfillment of the requirements for the degree of
Master of Science in Applied Informatics

Supervisor
Angelo Sifaleras

Approved by the three-member committee on 30/10/2019

Angelo Sifaleras	Nikolaos Samaras	Hristu - Varsakelis Dimitrios
....	....	....

Panagiotis Kalatzantonakis

....

ABSTRACT

The Capacitated Vehicle Routing Problem (CVRP) is a well-known NP-hard combinatorial optimization problem with numerous real-world applications in logistics. In this work, we present a literature review with recent successful parallel implementations of Variable Neighborhood Search regarding different variants of vehicle routing problems. We conduct an experimental study for the CVRP using well-known benchmark instances, and we present and investigate three parallelization strategies that coordinate the communication of the multiple processors. We experimentally evaluate a non-cooperative and two novel cooperation models, the managed cooperative and the parameterized cooperative strategies. Our results constitute a first proof-of-concept for the benefits of this new parameterized cooperative approach, especially in computationally hard instances.

Keywords: Combinatorial Optimization, Parallel Metaheuristics, Parallel VNS, Metaheuristics, Heuristics, Algorithm Design, General Variable Neighborhood Search, GVNS, Parallel Computing, Capacitated Vehicle Routing Problem, CVRP

List of publications derived from this thesis:

- Kalatzantonakis P., Sifaleras A., and Samaras N., "Cooperative Vs Non-Cooperative Parallel Variable Neighborhood Search Strategies: A case study on the Capacitated Vehicle Routing Problem", re-submitted to *Journal of Global Optimization*, Springer, 2019 (minor revision).
- Kalatzantonakis P., Sifaleras A., and Samaras N., "On a cooperative VNS parallelization strategy for the capacitated vehicle routing problem", to appear in N. Matsatsinis, Y. Marinakis, and P. M. Pardalos (Eds.), *Learning and Intelligent Optimization.* LION13. Lecture Notes in Computer Science, 27-31 May, Chania, Crete, Greece, 2019.
- Kalatzantonakis P., Sifaleras A., Samaras N., Migdalas Ath., "Parallel Variable Neighborhood Search for the Capacitated Vehicle Routing Problem", presented at the 6th *International Conference on Variable Neighborhood Search.* ICVNS, Sithonia, Halkidiki, Greece, October 4-7, 2018, p. 32 (book of abstracts).

To my wife Anna, my son Theseus and my parents.

May you grow up to be righteous
May you grow up to be true
May you always know the truth
And see the lights surrounding you
May you always be courageous
Stand upright and be strong
May you stay forever young

ACKNOWLEDGEMENTS

There are many individuals I would like to acknowledge and thank for the time and effort they dedicated in supporting me through this project and thesis.

Thank you, Angelo, for all of your time, guidance, and support over the past years. I have learned a great amount under your guidance. It has truly been a pleasure working together. To my advisors, Samaras Nikolaos and Hristu - Varsakelis Dimitrios, I owe you my gratitude. I appreciate the guidance and support. I would also like to acknowledge the Operations Research Center of UoM for providing amazing opportunities.

It is a pleasure for me to thank my friends and colleagues who have provided support and company, with a special mention to my friends Dr. John Daoutidis, Elpida Klepkou, Vladimir Augustin, Kyriakos Souvatzoglou, Dr. Nikolaos Ploskas, Dr. Geranis Georgios and my colleagues Alexandros Kokozidis, Vasilis Markopoulos, Georgia Kozari, Aggelos Pentelas, Chrysa Fotoglou, Anastasia Kyriakidou. Finally, I would like to thank my family, for their genuine, unconditional love and their unwavering support for my academic pursuit.

Lastly, I'd like to thank my wife Anna. If it wasn't for Anna, I would never be here to begin with. I dedicate this thesis to my son Theseus, my wife Anna and my family.

CONTENTS

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Combinatorial or discrete optimization problems have significant importance to many industrial applications, with a vast number of uses and can be described as the effort to find an optimal solution from a finite number of alternative solutions. The principal goal in optimization problems is to maximize benefit, mostly defined by something such as profit, time or quality. In real-life applications, money and time are limited resources, and this is the reason several constraints are applied in order to minimize resources utilization.

All optimization problems can be reduced into mathematical formulas and can be formally described as maximizing or minimizing a function by methodically choosing input values from a predefined set. A generic form of optimization problems is as follows:

$$\begin{aligned} &minimize\ f(x) \\ &subject\,to \begin{cases} c(x) = 0 \\ g(x) \leq 0 \end{cases} \end{aligned} \tag{1}$$

where $f(x)$ is the objective or cost function. The equality $c(x)$ and the inequality $g(x)$ are called constraints. Of course if we flip the inequality we will form a maximization problem.

In real-world applications, we usually encounter more than one objective function, and we expect conflicts between them since the improvement of one objective produces a disadvantage to another. In the industry, on the one hand, problems are usually extremely complex and time-consuming, with multiple constraints and many objectives. On the other hand, there are few computer resources, and there is a lack of computation time.

This thesis focuses on those combinatorial optimization problems that arise in the functional area of logistics and transportation management, specifically transport planning. The task of servicing multiple client demands, subject to several constraints, while optimizing predefined criteria is known in the Operational Research literature as the Vehicle Routing Problem.

The Vehicle Routing Problem and all its variants are NP-hard and therefore unlikely to be solvable in polynomial time (Lenstra and Kan, 1981). As with many hard optimization problems, metaheuristics are often used with vehicle routing problems in order to find optimal or nearly optimal solutions, within a reasonable amount of time. In many real-life problems, approaching the optimal solution is a time-consuming process. The requirements of this class of problems, stretch and frequently transcend what regular off-the-shelf computing systems can offer.

In order to reduce the computational time, the application of parallelism to a metaheuristic algorithm is a common practice. Apart from reducing the computational time, parallelization can increase the exploration space and offer solutions with better quality. One of the major problems that arise from the parallelization process concerns the communication between the model components. Communication affects the performance and the scalability of parallel algorithms.

1.1 Thesis Overview and Hypothesis

1.1.1 Objectives

The objectives of this thesis are as follows:

1. To survey recent successful parallel implementations of the Variable Neighborhood Search metaheuristic regarding several variants of vehicle routing problems.
2. To present three parallelization strategies using the General Variable Neighborhood Search variant to tackle the Capacitated Vehicle Routing Problem, in which we examine different approaches of exchanging solutions among parallel executions.
3. To experimentally evaluate how the level of cooperation can affect the performance between non-cooperative and cooperative models.
4. To improve the quality of the solutions by creating a new parallel model that filters communication based on the tests performed.

1.1.2 Approach

Experimental research was carried for the Capacitated Vehicle Routing Problem using a parallel version of the General Variable Neighborhood Search metaheuristic. Several parallel models were used into well-known benchmark instances. Afterwards, a qualitative investigation of the results was carried. This statistical analysis was done in order to compare the communication strategies of the parallel models, under several simple and complex instances.

A second analysis followed, concerning the performance of the parallel models. The interpretation of this study provided the necessary information in order to make various modifications and apply filters to coordinate the communication between the processors in order to increase performance and create a parallel model that will carry the benefits from both worlds.

1.1.3 Structure of this work

This thesis is organized as follows: In Chapter 2 we describe the VRP, its extensions and the relevant scientific literature. In this chapter we mainly focus on the CVRP extension and we define its mathematical formulation in subsection 2.2.1. In Chapter 3, we outline the solution methods for the VRP and their efficiency. In Chapter 4, we describe the methodology and the configuration of the GVNS approach. In Chapter 5, we describe the theory behind the parallelization of VNS and we present related works in parallel VNS for several VRP variants. In Chapter 6, we present three parallel GVNS models for the solution of the CVRP and discuss the cooperation strategies and the benefits and drawbacks of communication. The methodology and the findings from this computational study are presented in depth in Chapter 7. Chapter 7 also includes a study of the communication overhead among processes, solution generation and exchange and a study of the benchmark instance parameters in subsection 7.3.2. Finally, the conclusions and prospects are summarized in Chapter 8.

CHAPTER 2

Introduction to the Vehicle Routing Problem

2.1 The Vehicle Routing Problem

The Vehicle Routing Problem (henceforth referred to as the VRP) is one of the most well-studied combinatorial optimization problems, and it emerges when one seeks an optimal route or a combination of routes for a fleet of vehicles to accommodate a set of clients, given a set of constraints. The standard VRP is known to be NP-hard since it generalizes the most influential and well-known combinatorial optimization problems, the Travelling Salesman Problem (TSP) and the Bin Packing Problem (BPP) which are both popular NP-hard problems (Garey, 1979). It can also be considered as a Set Packing (SP) problem if the cost of each set is viewed as the distance of the associated optimal TSP tour (Balinski and Quandt, 1964). A comprehensive definition of VRPs is:

> ***"Vehicle Routing Problems are concerned with the delivery of some commodities from one or more depots to a number of customer locations with known demand. Such problems arise in many physical systems dealing with distribution networks. For example, delivery of commodities such as mail, food, newspapers, etc. The specific problem which arises is dependent upon the type of constraints and management objective."***
>
> Achuthan et al. (1997)

The formal VRP representation is a directed graph $G(E, V)$, where $V = 0, 1, ..., n$ depicts the set of nodes and E is the set of arcs. The depot is the first node marked with zero, the rest of the nodes $i = 1, 2, \dots, n$ represent the clients. The demand of each client is $d_i > 0$. Every arc depicts a route from node i to node j. The weight of every arc $C_{ij} > 0$ represents the cost of moving from node i to node j. When $C_{ij} = C_{ji}$ then the VRP is symmetric, oppositely if $C_{ij} \neq C_{ji}$, the VRP is asymmetric.

In view of those definitions, Vehicle Routing Problem is a general name for a wider class of problems that can be described with the following:

- the items are conveyed from a depot to a client,
- drivers are assigned to a set of vehicles,
- given is the road network
- given is a set of clients where the goods are delivered.

2.2 Extensions of the Vehicle Routing Problem

The fact that numerous VRP variants exist with several applications in the industry, made the VRP popular in the academic literature. The number of VRP variants have increased over the years. The most common variants of VRP are as follows:

2.2.1 The Capacitated VRP variant (CVRP)

The CVRP is the most studied version of the VRP and it is the problem being studied in this research. In this problem, the distribution of goods starts from a single depot and the goods are conveyed by a homogeneous fleet of vehicles, meaning that vehicles that are available at the depot all have the same capacity. An additional restriction is introduced, the vehicle capacity is considered an extra constraint.

It should be noted that, in spite of the fact that there are several variants of VRP that attracted substantial research efforts, according to an analysis of the Scopus scientific database, the CVRP proves to be the most prevalent variant of the VRP. From the total number of articles, about 43% is about the CVRP Gayialis et al. (2019).

Basic concepts and definitions

The CVRP, which was initially introduced by Dantzig and Ramser in 1959 (Dantzig and Ramser, 1959), belongs to the category of location routing problems and is a variation of the VRP, in which a fleet of homogeneous delivery vehicles of limited carrying capacity must service known customer demands from a depot, at a minimum transit cost. The figure **2.2.1** shows the graphic representation of a CVRP example.

The CVRP is an NP-hard problem with significant impact on the fields of transportation, distribution, and logistics, since transportation is usually a significant component of the cost of a product. One of the major concerns of the industry has always been the minimization of the product cost, essential both for the achievement of a more substantial profit as for the maintenance of a vantage over the competition. In addition, a growing interest in reducing the environmental impact of their products and services is also cultivated among companies, thus creating a trend towards greener management of modern supply chain (Skouri et al., 2018).

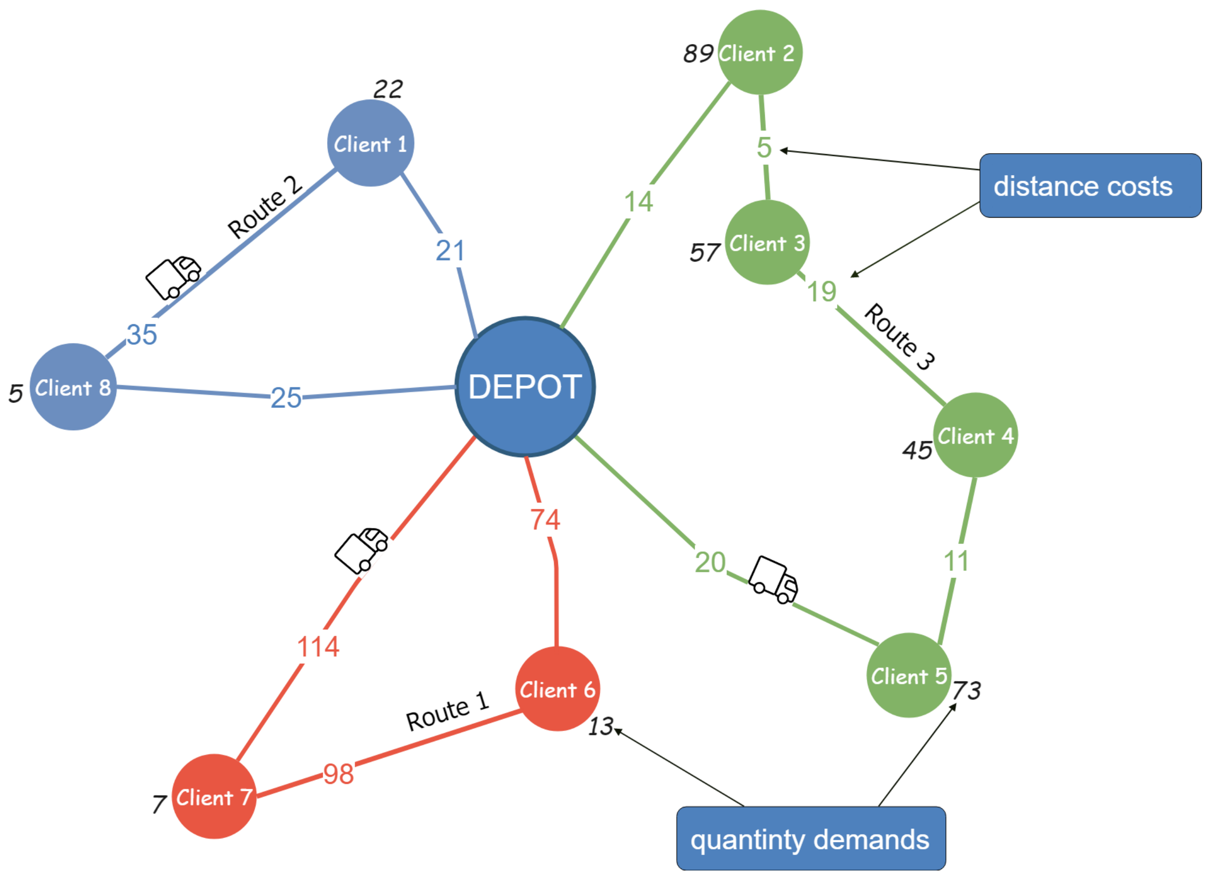

Figure **2.2.1** A depiction of the Capacitated Vehicle Routing Problem.

Finding an optimal solution for the CVRP is generally a computationally difficult problem. Exact algorithms exist, but those are not considered efficient due to the computational nature of all NP-hard problems, which leads to markedly long periods of computation time when in the process of solving problems with many customers. Many different exact algorithms are proposed in the literature for the CVRP (Baldacci et al., 2010). Laporte and Nobert (Laporte and Nobert, 1987) gave a survey covering early exact methods for the CVRP. Toth and Vigo (Toth and Vigo, 2002) delivered a complete overview of exact methods for the CVRP. For more insight into the literature of exact methods, the interested reader is referred to the work by Cordeau et al. (Cordeau et al., 2007).

CVRP mathematical formulation

Several mathematical formulations have been proposed for the VRP (O. et al., 2015), such as the Vehicle Flow Formulation (VFF), the Commodity Flow Formulation (CFF), and the Set Partitioning Formulation (SPF).

This study works on the simplest version of the CVRP problem, with the assumption of a homogeneous vehicle fleet (same capacity). VFF is employed in order to formulate the CVRP, under the assumption that the number of vehicles is unlimited and the goal is to obtain a solution that minimizes the total travel cost. VFF uses discrete variables associated with each arc that count the number of times that the edge is traversed by a vehicle. The number of variables is polynomially bounded, and the number of constraints is exponential (Toth and Vigo, 2002).

Let $G = (V, E)$ be an undirected weighted graph where $V = \{0, n\}$ and $i = 0$ the depot and $i = 1, \ldots, n$ the customers. By E we denote the set of edges, i.e., the roadways between the customers. V is the set of vertices, K is the available number of vehicles, and S the set of clients. Each edge (i, j) is associated with a non-negative cost c_{ij} and a binary variable x_{ij} (whether it is traversed or not). The CVRP formulation is as follows:

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}$$

s.t.

$$\sum_{i \in V} x_{ij} = 1 \quad \forall i \in V \backslash \{0\} \tag{1}$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall j \in V \backslash \{0\} \tag{2}$$

$$\sum_{i \in V} x_{i0} = K \tag{3}$$

$$\sum_{j \in V} x_{0j} = K \tag{4}$$

$$\sum_{i \notin V} \sum_{j \in V} x_{ij} \geq r\left(S\right) \quad \forall S \subseteq V \backslash \{0\}, S \neq 0 \tag{5}$$

$$x_{ij} \in (0, 1) \tag{6}$$

Constraints 1 and 2 impose the in-degree and out-degree constraints so that precisely one edge enters and leaves each vertex associated with a customer, respectively.

Constraints 1 and 2 impose the in-degree and out-degree constraints so that precisely one edge enters and leaves each vertex associated with a client, respectively. Constraints 3 and 4 impose client degree requirements for the depot. Constraint 5 is called the Capacity-Cut Constraint (CCC) and imposes both the connectivity of the solution and the vehicle capacity requirements. Capacity-Cut Constraints (CCCs) specify that each cut specified by vertex sets (S, S') is crossed by a number of edges not less than $r(S)$, where $r(S)$ stands for the minimum number of vehicles needed to serve set S. Constraint 6 describes the binary nature of variables x_{ij}, in other words, value 1 is assigned if edge $(i, j) \in E$ belongs to the optimal solution and 0 otherwise.

2.2.2 The Green VRP variant (G-VRP)

Green logistics or environmentally friendly logistics is an active research area that comes to focus when corporations are trying to decrease their carbon footprint voluntarily in response to a constantly increasing concern about climate change, or to adopt new environmental regulations. Occasionally, there are difficulties to overcome when there is insufficient refueling support for the vehicles, or limited vehicle driving range. Furthermore, ever so often the concern for global warming leads to new investments in environment-friendly transportation, that promote R&D activities.

2.2.3 The VRP with Backhauls (VRPB)

The VRP with Backhauls, also known as the linehaul-backhaul problem, pertains to the general class of delivery and collection VRP. In this class of problems, the clients are divided into two subsets, the linehaul and backhaul clients. In the first category, linehaul clients need goods to be delivered from the depot. In the second category, the backhaul clients have a given quantity of goods to be picked up and conveyed to the depot. In real-world problems, separate distribution and collection services on the same route are very popular, and that is the reason why VRPB is so common.

2.2.4 The Multi-Depot VRP variant (MDVRP)

The Multi-Depot Vehicle Routing Problem (MDVRP) is a prevalent problem with many real-life applications. The differentiation with VRP is that the set of customers will be served by a fleet of vehicles originating from multiple depots. This means that the vehicle has no obligation to return to its initial depot after serving the clients; preferably, it can pick the closest one. The optimization criterion of Multi-Depot VRP is to minimize the distance traveled.

2.2.5 The Multi-Echelon VRP variant (MEVRP)

The Multi-Echelon Vehicle Routing Problem (MEVRP) is an extension of the Capacitated VRP, where the transport from a single warehouse to the clients is accomplished by routing and connecting the vehicle through central warehouses called satellites while minimizing the overall transportation cost. MEVRP has widespread real-life applications because it seeks to utilize existing resources other than optimizing the whole distribution process.

2.2.6 The Multi-Trip VRP variant (MTVRP)

The Multi-Trip VRP (MTVRP) also known as VRP with Multiple-use of vehicles (VRPM) is a popular problem in the context of real-world applications. The main differentiation from the VRP is that rather than using a vehicle for only one route, vehicles may perform several routes. Specifically, if there are constraints that impose a small number of clients per route (small capacity or limited fleet size), then the quality of services can only be achieved when vehicles are reused. A new constraint is introduced, the sum of the duration of the tours ascribed to a vehicle should not surpass a given trip duration. It should be noted that not all vehicles are necessarily used. Also, in some papers, MTVRP has slight differences in the definition Cattaruzza et al. (2016).

2.2.7 The Open VRP variant (OVRP)

The Open VRP (OVRP) is a variant of VRP in which each route is a sequence of customers. The main differentiation from the classical VRP is that vehicles are not required to return to the depot after the end of the route, or likewise, the return trip to the depot is not charged. The Open vehicle routing problem has not gained much attention in the literature compared to other VRP variants Gurpreet and Vijay (2014).

2.2.8 The Time Dependent VRP variant (TDVRP)

The Time-Dependent VRP variant (TDVRP) was first introduced in 1992 by Malandraki and Daskin (Malandraki and Daskin, 1992) and by Hill and Benton (Hill and Benton, 1992). In this VRP extension the added constraint is the total time spent on the routes by the vehicles. Time must be minimized. The trip time that the vehicle spends between two clients or between a client and the depot also becomes an important factor. Instead of relying only on the distance between the clients, the TDVRP also factors the time of the day. On many occasions time-windows may also be given, as well as the maximum allowable duration of each route.

The TDVRP can be divided into 3 categories (Gendreau et al., 2015) depending on the quality and the progression of the information related to travel times, namely:

- Static and deterministic time-dependent VRP,
- Static and stochastic time-dependent VRP,
- Dynamic time-dependent VRP

2.2.9 The VRP with Satellite Facilities variant (VRPSF)

In the VRP with Satellite Facilities (VRPSF), satellites are utilized in order to refill the vehicle during a tour, allowing drivers to make deliveries to the clients until the end of their shift, without necessarily returning to the central depot (Gayialis et al., 2018; Bard et al., 1998).

2.2.10 The VRP with Time Windows variant (VRPTW)

The VRP with Time-Windows follows the CVRP in popularity. It accumulates a 37% of the total research in the Scopus scientific database. The VRPTW objective is to find a collection of routes of minimal total traveling cost, much like the classical VRP, but constrained by a time interval within which the clients that have to be served are prioritized.

VRPs with time window constraint can be divided into two different categories:

i. VRPs with Hard Time Window (VRPHTW). The time-window constraint must be satisfied,

ii. VRPs with Soft Time Window (VRPSTW). Failure to conform to the time-window constraint does not affect the feasibility of the solution. The solution that violates the Time-Window constraint receives a penalty in the objective function, thus increasing the solution cost.

2.2.11 Other VRP variants

Several other variants of the VRP that are derivatives of the above exist, such as:

- Multi-Trip VRP with Backhauls (MTVRPB) (Salhi et al., 2014),
- Fleet Size and Mix VRP with Backhauls (FSMVRPB) (Salhi et al., 2013),
- VRP with Backhauls and Time Windows (VRPBTW) (Zhong and Cole, 2005),
- Dynamic VRP (DVRP) (Pillac et al., 2013).
- The Stochastic Vehicle Routing Problem (Cordeau et al., 2007).
- The single VRP with deliveries and selective pickups (SVRPDSP) (Gribkovskaia et al., 2008).

The continued research of the VRP notwithstanding, there is no algorithm that optimally solves every problem, not even a specific VRP variant. Many diverse algorithms and methods have been proposed to this end for certain VRP classes, but they tackle only a particular issue in order to reduce the number of variables under consideration. In real-world applications, heuristic or metaheuristic algorithms are used as solutions.

CHAPTER 3

Solution methodology for VRP problems

As mentioned in section in 2.1, VRP and all its extensions are NP-hard. There are various methods and procedures to solve a VRP or approximate a solution in a decent running time. The vast majority fall into 3 main categories. Those categories are Exact, Heuristic, Metaheuristic, Approximation and Simulation. For the sake of completeness, in this section, we briefly review all available approaches.

3.1 Mathematical modelling for the VRP (Exact algorithms)

Exact solutions guarantee that the exact optimal solution is found, but as a consequence of the fact that the VRP belongs to NP-Hard class, there are limitations to what this method can offer — the time needed to calculate an exact solution increases when there are many customers to visit. Consequently, for real-world problems of the VRP, heuristics are used. Exact algorithms can be divided into the following categories (Laporte, 1992):

(a) **Direct tree search (DTS)**
   - (i) Assignment lower bound
   - (ii) k-degree centre tree

(b) **Dynamic programming (DP)** (Burrows et al., 1972)

(c) **Integer linear programming (ILP)**
   - (i) Vehicle flow formulation
   - (ii) Set partition & column generation

Focusing on CVRP problems, Branch-and-Cut algorithms (ILP) were the best performing algorithms until the early 2000s (Naddef and Rinaldi, 2002). Up until then, there were instances from the literature with only 50 nodes that could not be solved to optimality, while the most challenging CVRP instance that was solved had 134 nodes (Augerat et al., 1995).

It was Fukasawa Fukasawa et al. (2005) who at that moment in time demonstrated his Branch-and-Cut-and-Price algorithm (BCP) and proved that the combination of cut and column generation is much more effective than each of those techniques taken alone (Poggi and Uchoa, 2014). Fukasawa solved all instances from sets A,B,E,F,M,P with up to 100 customers, as well as instances M-n121-k7 and F-n135-k7. Only instances M-n151-k12, M-n200-k16 and M-n200-k17 remained unsolved. BCP is a hybrid of branch and bound and column generation. In 2007, the biggest instance for the CVRP that has been solved to optimality contained 135 nodes (Pisinger and Ropke, 2007).

In 2016, a new milestone was set by consistently solving CVRP instances up to 199 nodes using BCP algorithm (Pecin et al., 2016), while the bigger instance that was solved had 360 nodes (instance: Golden19, 45 hours). Even though breaking the barrier of 200 nodes was a significant advancement upon the previous record of 150 nodes, there still is a big obstacle that has to be surpassed when using exact algorithms. The effort involved to synthesize, code, and implement those extremely refined, sophisticated BCPs in a simple problem requires a significant effort. A skilled team could need several months of work to devise these algorithms for a specific task and those often correspond to only one variant of the VRP.

Pessoa et al. in 2009, proposed a generic BCP solver that encircles a large class of VRPs. Those authors (Pessoa et al., 2019) showed that the overall performance of their generic BCP in several problems produced better solutions than the best existing specialized algorithms. They managed to solve to optimality five instances from the X set for the first time. The largest instance had 548 nodes (instance: X-n548-k50). The maximum time needed to solve some of those instances was sixteen days, while the minimum was two days.

The size of the instances that were solved in optimality across the years can be seen in **3.1**. Denoted with an asterisk are individual instances that the researcher managed to solve and not a batch of instances.

Table **3.1** Exact algorithm performance on CVRP through the years

Year	Instance details	Customers	Algorithm	Hours
1995	-	50	Branch-and-Cut	
2003	-	80	Branch-and-Cut	
2006	A,B,E,F,M,P, M-n121-k7 and F-n135-k7	$\leq$ 100 on sets, 135	BCP	-
2016	M-n200-k16	199	BCP	7.95
2016	Golden_19	360*	BCP	45.11
2019	X-n284-k15	283	Generic BCP	264
2019	X-n322-k28	322	Generic BCP	134.4
2019	X-n393-k38	392	Generic BCP	139.2
2019	X-n469-k138	468	Generic BCP	364.8
2019	X-n548-k50	547	Generic BCP	48
2019	X-n548-k50	547	Generic BCP	48

The latest set with benchmark instances for the CVRP contains instances that range from 6000 to 30000 (Arnold et al., 2019). Based on the above, this set is currently beyond the capabilities of any exact algorithm, when there is a need for it to be solved fast.

Even the X set Uchoa et al. (2017a) with nodes ranging from 100 to 1000 is considered very hard if not impractical to craft and use exact algorithms on the harder instances. These new instances constituted a unique challenge for the research community. Through the years, feasible solutions for all those instances are provided by heuristics and metaheuristics methods.

At the conference of the German Operations Research Society in Hamburg in 2016, a statement by Sörensen jokingly referred to as "Sörensen's conjecture" reads as follows:

"In the real world, solving optimization problems using exact methods is a waste of resources."

Sörensen et al. (2016)

Popular integer or constraint programming solvers are principally based on Tree Search techniques (branch-and-bound, branch-and-cut, branch-cut-price). These methods explore the solution space by recursively instantiating variables forming a solution vector. Operating in exponential time, the main disadvantage is the restriction to small and medium-scale problems. If tree search techniques execute without termination, they offer no more guarantee on the solution quality than any heuristic approach. Numerous real-life VRPs require thousands of 0-1 decisions variables, which are out of tree search scope (Ibrahim et al., 2019).

On the contrary, commercial robust general-purpose heuristic solvers dealing with nonlinear 0-1 models, produce high-quality solutions in extremely short running times without any tuning. One good example of a commercial heuristic solver is LocalSolver (Benoist et al., 2011).

3.2 Approximation algorithms for the VRP

In real-world problems, it is reasonable to expect a solution for an essential task or a minimal instance of about 200 customers, but it is likely to get that solution within several days or weeks. In contrast, there is a special type of algorithm called approximation algorithms that are time efficient and designed to produce provably good quality solutions, close to the optimal one, in provably polynomial computation time. An approximation algorithm has an approximation ratio of a(n), if for any input of the problem, the cost C of its solution is within factor ρ of the cost of the optimal solution C^*, i.e.

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right)$$

This definition works for minimization and maximization problems. It would be perfect if the approximation ratio ρ is a small constant, but for various hard problems, the best-known approximation ratio ρ depends either or both on the input size n and some other parameters linked to the problem (Vazirani, 2003).

3.3 Simulation methods for the VRP

In VRP problems, the reliability of a route can be defined as the probability that the route does not suffer a failure. This probability aspect makes it possible to get an acceptable solution by using simulation methods. The use of simulation tools in VRPs is scarce and uncommon (Juan et al., 2013). When simulation methods are used, researchers use them in order to obtain estimates of the cost and the reliability of each solution. Simulation methods use what-if analyses for producing various scenarios, picking the best one, and applying these in the principal algorithms (Alexander et al., 2014). One way to achieve the generation of those scenarios is to use a Monte Carlo Simulation.

Monte Carlo Simulation (MCS) can be described as a set of methods that make use of random number generation to solve specific stochastic or deterministic problems. Heuristic methods for the CVRP can be combined with Simulation methods. Fernández de Córdoba et al. applied MCS along with the Clarke and Wright heuristic for tackling the CVRP. Those authors (de Córdoba et al., 2000) confirmed that Monte Carlo simulations are useful in implementing heuristic algorithms for different Routing Problems. Two crucial advantages of MCS are that (i) they are easy to implement, and (ii) they can adapt quickly to any problem.

3.4 Heuristics for the VRP

Frequently, in real-world problems, truly optimal solutions are rarely found, and it is considered natural to compromise the quality of the solution in order to acquire a solution in a reasonable time. The diversity and randomness, which is intrinsic in real-world tasks can be dealt with heuristic solutions.

Heuristic methods can be divided into two main categories:

- Construction heuristics,
- Improvement heuristics

These two methods will be discussed in detail in the next sections.

3.4.1 Construction Heuristics

A construction heuristic is an algorithm that constructs a solution incrementally according to some rules. In CVRP problems, in every iteration, a new tour is attached to the solution consecutively, and parts previously created remain unchanged. Each route is an individual sub-problem and can be built sequentially or in parallel. The main construction heuristics for the CVRP fall into the following categories:

(a) **TSP based heuristics** can be used for solving VRPs, in order to construct multiple tours, apply the problem constraints and then filter out unwanted tours. Tours may be built by adding nearest nodes, farthest nodes or random nodes.

(b) **Insertion heuristics** begin with a sub-tour and then extend this tour by inserting the remaining nodes consecutively until all nodes have been included. Insertion rules may vary. A simple greedy strategy is to insert a node in the cheapest possible route repeatedly.

(c) **Cluster-First Route-Second** (CFRS) method is used to build clusters of nodes and then determine a feasible route per cluster. The following algorithms use Cluster-First Route-Second method:

(c.i) The **Sweep algorithm** (Gillett and Miller, 1974; Wren and Holliday, 1972; Lampkin and Wren, 1972), can only be implemented in planar VRPs. Sweep algorithm scans the surface in Euclidean space. Nodes are combined in order to form a cluster by rotating a ray centered at the depot. While increasing the polar angle centered at the depot, nodes are added to the route in the order they appear. If a node is feasible, it is included in the route; on the other hand, if a node is not feasible, a new route is created. The implementation of Gillette and Miller (Gillett and Miller, 1974) can do some post-optimization steps by switching nodes.

(c.ii) The **Petal algorithm** (Ryan et al., 1993), aims to construct a subset of feasible routes and then solve a set partitioning problem. It is an extension of the Sweep algorithm. Every route is called a petal. It is often observed that optimal solutions for several problems exhibit petal-shaped geometric structures (Ryan et al., 1993). Clustering methods that promote such geometric features (figure **3.4.1**) often achieve near-optimal solutions for practical-sized problems.

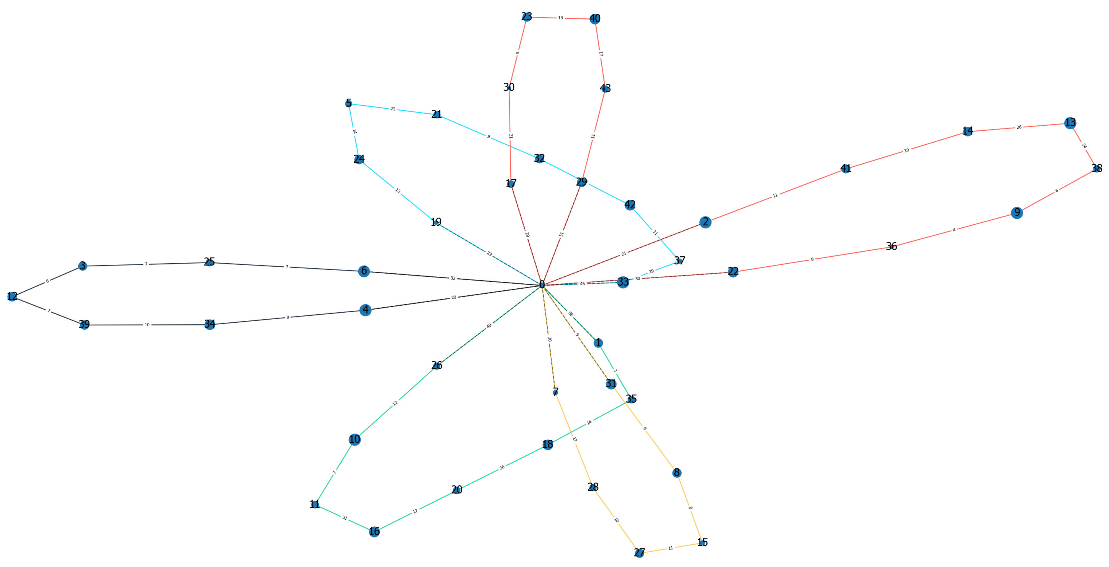

Figure **3.4.1**: Optimal solution of A-n44-k6 using PCM GVNS. The solution exhibits petal-shaped structure.

(c.iii) **Fisher and Jaikumar** algorithm (FJ) solves a Generalized Assignment Problem (GAP) to form the clusters (Fisher and Jaikumar, 1981). This algorithm was introduced for the capacitated vehicle routing problem. A cluster indicates a collection of customers or a city, visited by a vehicle (Hashimoto et al., 2010). FJ algorithm was modified by Koskosidis et al. (Koskosidis et al., 1992) in order to also work with the CVRPTW.

(c.iv) **Taillard algorithm** (Taillard, 1993a) also works only for planar VRPs. The main differentiation is that it decomposes the main problems into sub-problems by formulating regularly separated clusters. It assumes that the depot is centered and the clusters are around the depot. Taillard's algorithm determines neighborhoods by using the $\lambda-interchange$ generation mechanism (Osman, 1993).

(d) Unlike CFRS, **Route-First, Cluster-Second** algorithms (RFCS) (Newton and Thomas, 1974; Bodin and Berman, 1979; Beasley, 1983) create several capacitated VRP routes from a single tour. RFCS runs in two different phases. In the first phase, a large route that serves all customers is formed, similar to a TSP solution, while ignoring all side constraints. In the second phase, this large tour is sectioned into several sub-routes from the depot, that honor constraints. RFSC algorithm is a general and flexible method but is seldom used because, as stated above (c.ii), clustering-first methods often achieve near-optimal solutions due to the geometric features.

(e) In 1964, the famous **Clarke and Wright Saving** algorithm (Clarke and Wright, 1964) were developed. The number of vehicles in VRP in which CW is used is unfixed and is considered as a decision variable. CW savings algorithm is applied to both directed and undirected problems VRPs. In directed VRPs, CW produces a lower quality of solutions (Vigo, 1996). This heuristic method relies on iteratively linking routes in order to create a single route that produces the maximum cost saving. In the figure **3.4.2**, the first step of the CW algorithm is shown.

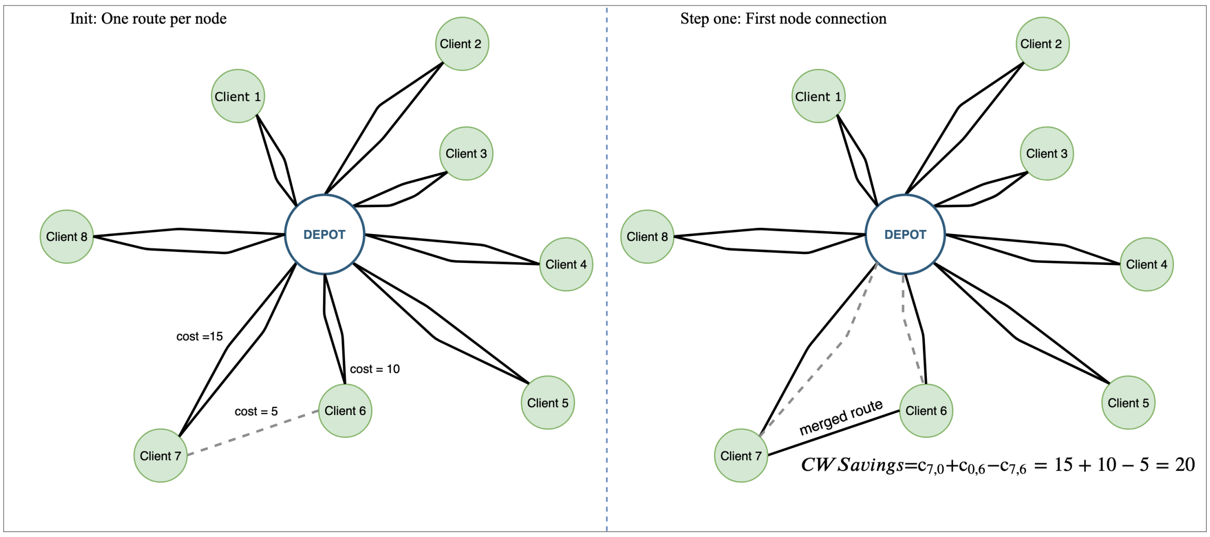

Figure **3.4.2** Savings algorithm. Initial connections with a depot and first iteration of CW.

The algorithm starts with one route for every customer, and in every iteration, the number of routes is reduced by merging two routes that give max savings (S) (Figures **3.4.3**). The classical Savings formula (S_{ij}) is depicted in Equation 1. In this equation, i and j represent a pair of customers, D is the depot and the function $dist(x, y)$ calculates the distance between two given points.

$$S_{ij} = dist(D, i) + dist(D, j) - dist(i, j) \tag{1}$$

Algorithm 1: Pseudo code for the Clark Write Savings algorithm (**CW**)

```
1 $initialiseRoutes()$;
2 $l \leftarrow CalcAndSortSavings()$;
3 for $i \leftarrow 1$ to $l_{max}$ do
4     $R^{l[i][0]}, R^{l[i][1]} \leftarrow findRoutes(l_{ij})$;
5     if $feasibleMerge(R^i, R^j)$ then
6         $combineRoute(R^i, R^j)$ ;
7     end
8 end
```

The algorithm 1 operates in the following way:

1 Construct n routes: $c_0 \to c_1 \to c_0$, for each $i \geq 1$. Calculate the savings which is given by equation 1 then sort the savings list in descending order.

2 A parallel and a sequential version of the algorithm are available. In contrast with the parallel version, the sequential version can only construct one route at a time.

   i. **Parallel version**: In essence, the parallel version searches for the best feasible merge. Given a saving s_{ij} from the top of the savings list, determine if two routes exist that can be feasible when merged. One of the two routes should contain the edge $(i, 0)$ and the other the edge $(0, j)$. In order to merge them delete $(0, j)$ and $(i, 0)$ and insert (i, j).

   ii. **Sequential version**: For every route $(0, i, \dots, j, 0)$ Find the first merge S_{ki} or S_{jl} that can be feasibly merged into the current route with another route containing $(k, 0)$ or $(0, l)$. Apply the merge and repeat the process on the same route. If no other merge exists, proceed to the next route.

3 Repeat step 2 until no additional savings can be achieved.

According to Laporte and Semet (2002) the parallel version of the savings method definitely dominates the sequential one. The same authors point out a disadvantage of Clarke and Wright's saving algorithm. In spite of the fact that CW produces good routes at the beginning, it tends to generate less satisfying routes towards the end, including some that are problematic. In order to address this issue Gaskell (1967) and Yellow (1970) proposed a route shape parameter λ. The generalized savings is of the form:

$$S_{ij} = dist(D, i) + dist(D, j) - \lambda * dist(i, j) \tag{2}$$

The λ parameter is putting more emphasis on the distance between the nodes to be connected. It has been reported that $\lambda = 0.4$ or $\lambda = 1$ yields good results (Golden et al., 1977).

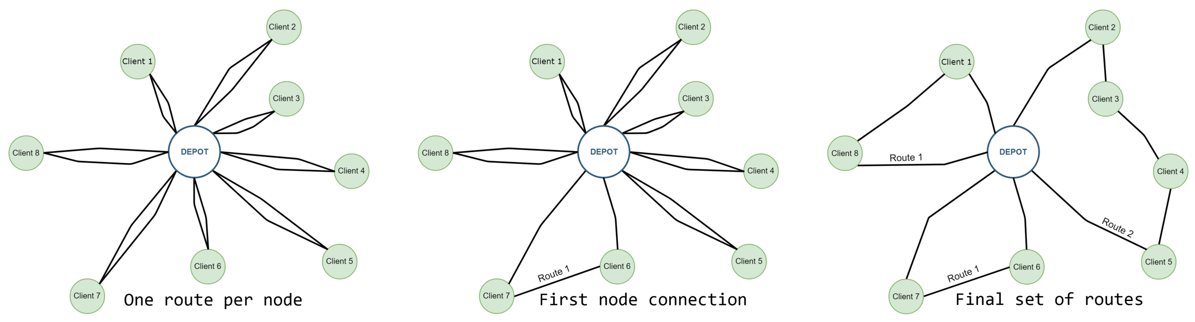

Figure **3.4.3** Savings algorithm. Start, first iteration, and last iteration.

3.4.2 Improvement Heuristics

In contrast with Construction heuristics, Improvement Heuristics are algorithms that, in every iteration, apply an improvement to the initial solution. The initial solution can be generated randomly or by using one of the construction heuristics mentioned above. The improvements in the initial solution can be to exchange clients between two routes, placing a client to a different route, swap two or more clients within the same route in order to alter the visiting sequence. In order for those improvements to happen, a local search is employed as a mean to produce an acceptable move from one solution to another inside a neighborhood structure. A neighborhood is a collection of all the possible solutions that can be reached from one solution by performing only one move. Common neighborhood structures for the CVRP are:

A. Intra-route Neighborhoods,

- i 2-opt (figure **3.4.4**),
- ii 3-opt,
- iii Or-opt (2H-opt)

B. Inter-route Neighborhoods

- i λ-interchange,
- ii relocate (figure **3.4.4**),
- iii swap (figure **3.4.4**),
- iv cross,
- v generalized insertion

The process of improvement ends when a predefined stopping criterion is met. Standard stopping criteria are (i) time, (ii) number of iterations and (iii) stagnation of improvements through a certain number of iterations.

In CVRP the neighborhood structures that are usually employed are Relocate, Swap and 2-opt, as well as their generalizations. These neighborhoods are ample to ensure that most solutions emerging from a local search include optimal routes (Toffolo et al., 2019).

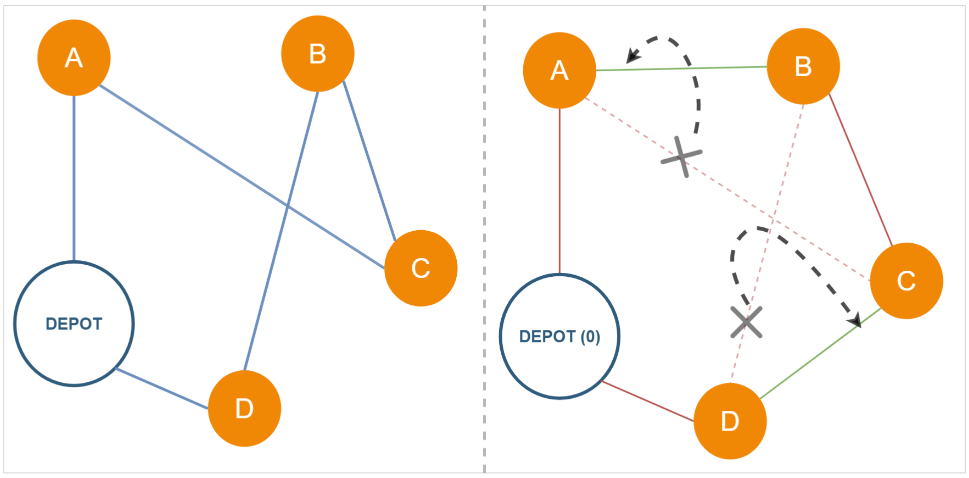

Figure **3.4.4** In a 2-opt, when removing two edges, there is only one alternative possible solution.

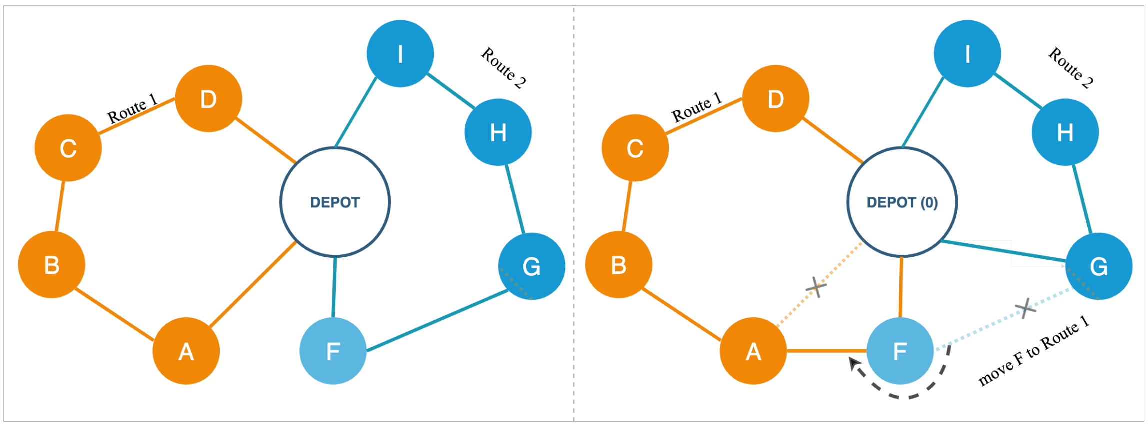

Figure **3.4.5** In relocate, a node is removed from a route and placed into another route.

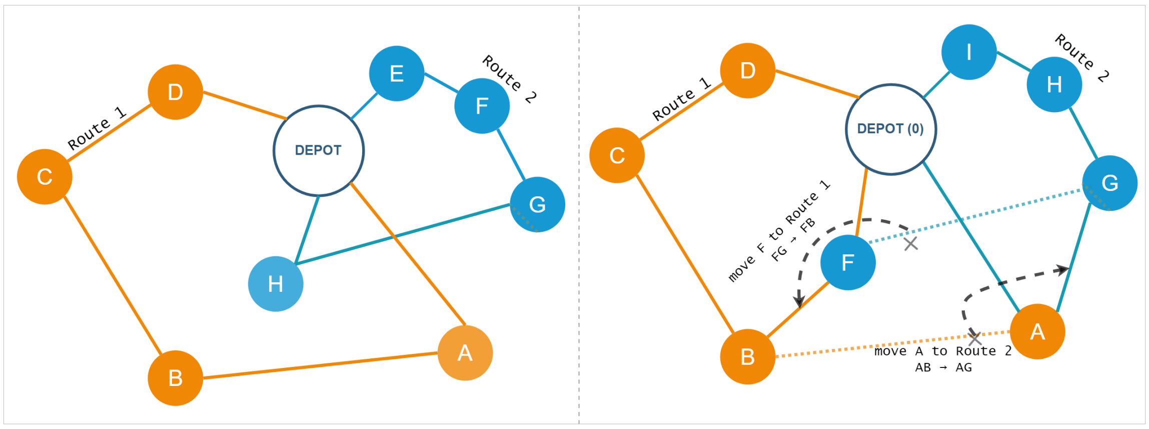

Figure **3.4.6** In swap (exchange), two nodes from different routes swap places.

Improvement heuristics frequently fail to obtain a global optimum because they regularly get trapped in a local optimal solution. This condition has led to the development of global optimization heuristics, also called metaheuristics. In order to produce a new solution out of the current one, metaheuristics use a heuristic which guides the search approach and can temporarily accept a less optimal solution to avoid local minima.

3.5 Metaheuristics

Metaheuristics can find applications in complex problem-solving in a wide array of fields ranging from finance to production management and engineering and pose as the most promising and practical solution methods for the VRP (Gendreau et al., 2007). Like heuristics, metaheuristic optimization methods may not ensure that an optimal solution is found, but they produce a satisfactory solution, in less time (Rothlauf, 2011). A metaheuristic is considered to be successful on a given optimization problem if it can achieve a balance between the exploitation of the collected search experience and the exploration of the unexplored search space to identify regions with high-quality solutions.

A rudimentary way to describe the metaheuristics is heuristics guiding other heuristics. As suggested by the Greek prefix "meta", metaheuristics are higher-level heuristics, in contrast with problem-specific heuristics. In 2003, Glover and Kochenberger issued a handbook that provided a comprehensive definition of metaheuristics:

> ***"Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes."***
>
> Glover and Kochenberger (2003)

One might argue that an accurate and more precise definition of metaheuristics is still debated. A more recent definition of metaheuristics is:

> ***"A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework."***
>
> Sörensen and Glover (2013)

3.5.1 Classification of Metaheuristics

Metaheuristics are part of a very diverse group of algorithms including: Variable Neighborhood Search (VNS), Tabu Search (TS), Genetic Algorithms (GA), Greedy Randomized Adaptive Search Procedures (GRASP), Ant Colony Optimization (ACO), Simulated Annealing (SA), Swarm Intelligence (SI), Large Neighborhood Search (LNS), Iterated Local Search (ILS), Scatter Search (SS), Metaheuristic Hybrids and others (Gendreau and Potvin, 2010). Variable Neighborhood Search is the method used in this thesis and will be discussed further in section 3.6.

Based on several of their distinctive attributes and aspects, a systematic classification for metaheuristics is possible. A thorough study has been made by Birrattari et al. (2001); Nesmachnow (2014); Jaradat et al. (2016) in order to describe and classify them. They generally fall into two main categories:

A. **Population-based vs. single-point search**,
Population-based search uses a population of search points instead of one single search point.
Population based: ACO, GA

B. **Memory usage vs. memory-less methods**,
The use of the search experience (memory) is another distinctive characteristic.
Memory is used in TS and ACO and partially in GA and ILS

C. **One vs. various neighborhood structures**,
Multiple neighborhoods are used in VNS and ILS and partially in GA.

D. **Single trajectory vs discontinuous trajectory methods**,
Trajectory-based metaheuristics can be separated into two groups. Single trajectory methods improve a single solution over and over again in an iterative fashion, improving only one solution in its neighborhood and forming a search trajectory in the neighborhood solution space. Discontinuous trajectory methods perform large jumps in the search space (discontinuous walk). Single trajectory methods include SA and TS. Discontinuous trajectory methods include VNS, GA, ACO, ILS, GRASP.

E. **Nature-inspired vs. non-nature inspiration**,
SA, GA and ACO are nature-inspired.

3.6 Variable Neighborhood Search metaheuristic

VNS is a metaheuristic, or framework for building heuristics, proposed by Nenad Mladenović and Pierre Hansen in 1997 (Mladenović and Hansen, 1997). VNS and its variants are generally perceived as a very efficient approach for various hard optimization problems. VNS and several VNS hybrids have been used in order to solve standard versions of the vehicle routing problem along with practical applications of routing problems (Hansen et al., 2009).

The key concept of VNS is systematically changing the neighborhood for a given solution. This process is accomplished in two phases. The first is a descent phase that attempts to find a local or a global optimum and the second is a perturbation phase, that is designed to disrupt the process in order to escape the area of the corresponding valley (Hansen et al., 2008).

VNS follows a discontinued trajectory (thus classified as a discontinuous method). The repetitive utilization of local searches structures facilitates distant jumps from one neighborhood structure to another. Neighborhoods $(N_1, N_2, \ldots, N_{n-1}, N_n)$ can be arbitrarily chosen, but it is a good practice not to overlap $(N_1 \in N_2 \in \ldots \in N_{n-1} \in N_n)$ because it may generate an ineffective search resulting in a large number of solutions that can be revisited (Blum and Roli, 2003).

Following the generation of the initial solution, the process of VNS starts. The VNS process can be analyzed into three steps: shaking, local search and move. Shaking step randomly selects a solution S' that belongs in the same neighborhood of the current solution S using the same neighborhood operator. Local search generates a solution S'' from S' using any neighborhood structure. When local search ends, if S'' is better than S' then S'' replaces S' and the iteration through the neighborhood structures begins again. If there is no improvement, the algorithm moves to the next neighborhood $n + 1$ and a new shaking phase begins using this neighborhood structure.

After choosing an initial solution s, a descent starts in order to find a better solution. When there is no direction of descent, the local search heuristic stops. The steepest direction of descent is called best improvement (Algorithm 2) and is commonly used.

Finding the best solution within a neighborhood structure may be time-consuming, an alternative is to use the first descent heuristic where a move is made as soon as a direction for the descent is found. This is illustrated in Algorithm 3. The neighborhood change exists in all schemes. It is a central part of the underlying logic of the VNS method and is responsible for the solution move within a VNS framework. Neighborhood change is illustrated in Algorithm 4.

VNS method, illustrated in Algorithm 5, is based on three simple facts:

Fact A. A local minimum with respect to one neighbourhood structure is not necessarily a local minimum for another neighbourhood structure

Fact B. A global minimum is a local minimum with respect to all possible neighbourhood structures

Fact C. For many problems, local minima with respect to one or several neighbourhoods are relatively close to each other.

Algorithm 2: Pseudo code for the **Best Improvement** function

```
1 repeat
2     $x' \leftarrow x$;
3     // Explore all neighborhoods and return the solution with the lowest/maximum
        objective function value
4     $x \leftarrow \mathrm{argmin}_{\mathrm{y \in N(x)}} f(y)$;
5 until $f(x) \geq f(x')$;
```

While the first two proposals are apparent, the third proposal is not so obvious and emerges as an empirical result. It is often observed in practice that regularly a local optimum gives some information

Algorithm 3: Pseudo code for the **First Improvement** function

```
repeat
    $x' \leftarrow x$;
    // Reset neighborhood counter
    $i \leftarrow 0$;
    repeat
        // Move to next neighborhood structure
        $i \leftarrow i + 1$;
        // Find the best neighbor in $N(x)$
        $x \leftarrow argmin(f(x), f(x_i)),\ x \in N(x)$;
    until ( $f(x) < f(x'_i)$ or $i = |N(x)|$ );
until $f(x) \geq f(x')$;
```

Algorithm 4: Pseudo code for the **Neighborhood Change** function based on best improvement

```
input  : $x$,$x'$,$k$
output: $x$,$k$
if $f(x') < f(x)$ then
    $x \leftarrow x'$
    // Move back to the first neighborhood
    $k \leftarrow 1$
else
    $k \leftarrow k + 1$
    // Next neighborhood
end
```

regarding the global optimum, in the way that in two different solutions some variables possibly have the same values. Though, no information is given on what these variables are (Hansen et al., 2009).

In section 3.7, all VNS schemes are analyzed. The basic schemes of VNS stem from the different way of utilizing the three fundamental propositions mentioned above. The different ways of use are:

i. deterministic,

ii. stochastic,

iii. both deterministic and stochastic to each other.

Algorithm 5: Pseudo code for the Variable Neighborhood Search algorithm (**VNS**)

```
input   : s:, A random initial solution, $t_{max}$: Max execution time,
          $k_{max}$: Max neighborhoods to be explored from a set of neighborhoods
output  : The best solution
while stopping criteria are not met do
    $k \leftarrow 1$
    while $k < k_{max}$ do
        // Select a random solution $s'$ from the $k^{th}$ neighborhood $N_k(S)$
        $s' \leftarrow Shake\,(s, N_k)$;
        // Apply a local search starting from $s'$ to get a solution $s''$
        $s'' \leftarrow LocalSearch(s')$
        if $s''$ is better than $s'$ then
            $s \leftarrow s''$
            // Move back to the first neighborhood (Best Improvement strategy)
            $k \leftarrow 1$
        else
            // Move to the next neighborhood
            $k \leftarrow k + 1$
        end
    end
end
```

3.7 Basic VNS schemes and extensions

The basic schemes of VNS are the following:

A. **Variable Neighborhood Descent (VND)**

The VND method (Algorithm 6) is achieved if the change of neighborhoods is performed in a deterministic way. On the contrary, VNS is a stochastic version of Variable Neighborhood Descent (Talbi, 2009).
In step 5, best improvement is used. If the first improvement is used instead, then we obtain the Sequential (piped) VND method. Sequential VND often performs substantially better when dealing with very large neighborhoods.

B. **Reduced VNS (RVNS)**

In order to obtain the reduced VNS (RVNS) method, a random point (solution) is chosen from the neighborhood $Nk_{(x)}$. Instead of starting a descent from this point, the value of the new point is compared with the current solution and an update occurs in the case of improvement, as seen in Algorithm 7. The two parameters t_{max} and k_{max}, used in all schemes, represent the maximum CPU time allowed and the maximum number of iterations between two improvements. The maximum number of iterations between two improvements is regularly employed as a stopping condition. It has been observed that the best value for the parameter k_{max} is often 2.

The RVNS is similar to a Monte-Carlo method, but it is more systematic and it is beneficial in very large instances, for which local search is costly (Mladenović, 2011).

C. **Basic VNS (BVNS)**

A combination of deterministic and stochastic changes of neighbourhood yields the Basic VNS (BVNS) method (Mladenović and Hansen, 1997).

The BVNS method is shown in Algorithm 8. A common practice is to use successive nested neighborhoods N_k. At the beginning, the solution is produced randomly (step 5). The first improvement local search, in step 7, is most commonly used but it can be replaced with best improvement.

D. **General VNS (GVNS)**

When VND is applied as a local search procedure in VNS then a more generic VNS emerges, called General Variable Neighborhood Descent (GVNS), as illustrated in Algorithm 9. Employing this General VNS (VNS/VND) approach has led to the most successful applications reported (Hansen et al., 2009).

E. **Skewed VNS**

The skewed VNS (SVNS) method tackles the problem of exploring valleys far from the current best solution.

Admittedly, immediately after the best solution in a large region has been found, it is essential to go some way to obtain an improved one. When the search for a new solution -picked at random- is in a distant neighborhood, then the new solution may deviate substantially from the current best. VNS then degenerates into the Multistart heuristic which is known not to be very efficient (Hansen et al., 2009).

Considering this problem, Skewed VNS is proposed (Algorithm 11). In step 7, the best among two solutions is kept. When the neighborhood changes a function $\alpha\rho\ (x,\ x'')$ is used. $\rho\ (x,\ x'')$ is used to estimate the distance between the current solution s and the local optimum found s''.

The α parameter that feeds the $NeighborhoodChangeS$ function, must be selected in such a way that it is possible to explore areas well spaced from the s region. When $\rho(x, x'')$ is small, then a large value for α should be selected in order to avoid frequent transitions to solutions near the current one.

Two important extensions are:

A. **Variable Neighborhood Decomposition Search (VNDS)** VNDS is an extension of the BVNS (Algorithm 12). In particular, this method is based on the logic of decomposing the problem in problems with much smaller size.

The parameter t_d, represents the time allowed to solve each sub-problem. Considering that the whole solution s is a set of properties, then the set y is the one assigned to the properties of randomly generated s', which are not s properties. BVNS is called in step 7, with s' as the initial solution so that obtaining a local optimal y' is achieved in the

y domain. Then, s'' is defined as the area in $s'' \leftarrow (s'/y) \cup y'$. VNDS can be time-consuming but it is remarkable that the exploitation of certain marginal situations can lead to a significant improvement in the solution.

B. **Parallel VNS (PVNS)**

Another extension of VNS is the Parallel VNS (PVNS) method. There are numerous approaches to parallelize VNS. The first attempt to parallelize the VNS was presented in García-López et al. (2002) some of which have been proposed for solving the p-Median problem. VNS parallelization will be discussed in detail in Chapter 5.

Algorithm 6: Pseudo code for the standard Variable Neighborhood Descent algorithm (**VND**)

```
input  : $x$,$k_{max}$
Init   : Choose at random an initial solution $s$
         Select a set on Neighborhoods $N_n,\ n = 1, \dots, n_{\max}$.
output: The best solution
1  while stopping criteria is not satisfied do
2      $k \leftarrow 1$
3      while $k < k_{max}$ do
4          // Standard VNS: best improvement.  Sequential VND: first improvement.
5          Select the best solution $S'$ from the $n^{th}$ neighborhood $N_n(S)$ of s
6          if $s'$ is better than $s$ then
7              $s' \leftarrow s$
8              $k \leftarrow 1$
9          else
10             $k \leftarrow k + 1$
11         end
12     end
13 end
```

Algorithm 7: Pseudo code for the Reduced Variable Neighborhood Search (**RVNS**) algorithm

```
input  : $s$, $_{kmax}$, $t_{max}$
output: $s$
1  repeat
2      $k \leftarrow 1$;
3      repeat
4          $s' \leftarrow Shake\,(s, k)$;
5          $s, k \leftarrow NeighborhoodChange\,(s, s', k)$;
6      until $k = k_{max}$;
7      $t \leftarrow CpuTime()$;
8  until $t \geq t_{max}$;
```

Algorithm 8: Pseudo code for the Basic Variable Neighborhood Search (**BVNS**) algorithm

```
input  : $s$, $_{kmax}$, $t_{max}$
output: the best solution
repeat
    $k \leftarrow 1$;
    repeat
        // Select a random solution $s'$ from the $n^{th}$ neighbourhood
        $s' \leftarrow Shake\,(s, k)$;
        // Local search
        $s'' \leftarrow FirstImprovement\,(s')$;
        // Change neighbourhood
        $NeighborhoodChange\,(s, s'', k)$;
    until $k = k_{max}$;
    $t \leftarrow CpuTime()$;
until $t \geq t_{max}$;
```

Algorithm 9: Pseudo code for the General Variable Neighborhood Search algorithm (**GVNS**)

```
input  : $s$, $l_{max}$, $_{kmax}$, $t_{max}$
output: the best solution
while $t < t_{max}$ do
    $k \leftarrow 1$;
    while $k < k_{max}$ do
        // Select a random solution $s'$ from the $n^{th}$ neighbourhood
        $s' \leftarrow Shake\,(s,\ k)$;
        // Execute VND on the random solution
        $s'' \leftarrow VND\,(s',\ l_{max})$;
        $s,\ k \leftarrow NeighborhoodChange\,(s,\ s'',\ k)$;
    end
    $t \leftarrow CpuTime()$;
end
```

Algorithm 10: Pseudo code for the Skewed VNS neighborhood change

```
input  : $s$,$s''$,$k$,$a$
output: $s$,$k$
if $f(s') - a\rho(s'', s) < f(s)$ then
    // Move to better solution
    $s \leftarrow s''$
    // Initial neighborhood
    $k \leftarrow 1$
else
    $k \leftarrow k + 1$
    // Next neighborhood
end
```

Algorithm 11: Pseudo code for the Skewed VNS algorithm (**SVNS**)

```
input  : $s$, $k_{kmax}$, $t_{max}$, $\alpha$
output: the best solution
while $t < t_{max}$ do
    $k \leftarrow 1$;
    while $k < k_{max}$ do
        // Select a random solution $s'$ from the $n^{th}$ neighbourhood
        $s' \leftarrow Shake\,(s,\ k)$;
        $s'' \leftarrow FirstImprovement\,(s')$;
        $KeepBest\,(s_{best},\ s)$;
        $NeighborhoodChange\,(s,\ s'',\ k,\ \alpha)$;
    end
    $s \leftarrow s_{best}$;
    $t \leftarrow CpuTime()$;
end
```

Algorithm 12: Pseudo code for Variable Neighborhood Decomposition Search (**VNDS**)

```
input  : $s$, $k_{kmax}$, $t_{max}$, $t_d$
output: the best solution
while $t < t_{max}$ do
    $k \leftarrow 1$;
    while $k < k_{max}$ do
        // Select a random solution $s'$ from the $n^{th}$ neighbourhood
        $s' \leftarrow Shake\,(s,\ k)$;
        $y \leftarrow s'/s$;
        $y' \leftarrow BVNS(y,\ k,\ t_d)$;
        $s'' \leftarrow (s'/y)\ \cup\ y'$;
        $s''' \leftarrow FirstImprovement\,(s'')$;
        $s,\ k \leftarrow NeighborhoodChange\,(s,\ s''',\ k)$;
    end
    $t \leftarrow CpuTime()$;
end
```

CHAPTER 4

The proposed GVNS solution approach

4.1 Initialization Phase

In order to find a feasible initial solution, the Savings algorithm of Clarke and Wright (1964) has been implemented as described in section 3.4.1 (Figures **3.4.2** and **3.4.3**). This construction heuristic provided the initial solution for all parallel GVNS models, as it is one of the best and the most widely known heuristic approach to solve the VRP.

4.2 Improvement Phase

4.2.1 Neighborhood structures used in the GVNS models

All models are using the same neighborhood structures, consisting of three widely used inter and intra-route operators (Figure **4.2.1**):

- 2-opt (Intra-route)
  performing 2-interchange moves.
- Swap (Inter-route).
  This neighborhood consists of swapping the positions of two clients (a and b) from different routes ($Route_a$ and $Route_b$).
- Relocate (Inter-route)
  In this local search operator, the customer is transferred from his current route R_a to a new route R_b.

The GVNS method (Hansen et al., 2010), described in Algorithm 9, is used to improve the initial solution given by the Clarke and Wright algorithm for all the proposed methods. The Variable Neighborhood Descent (VND) in line 5 is described in Algorithm 6. By t_{max}, k_{max}, and l_{max} we denote

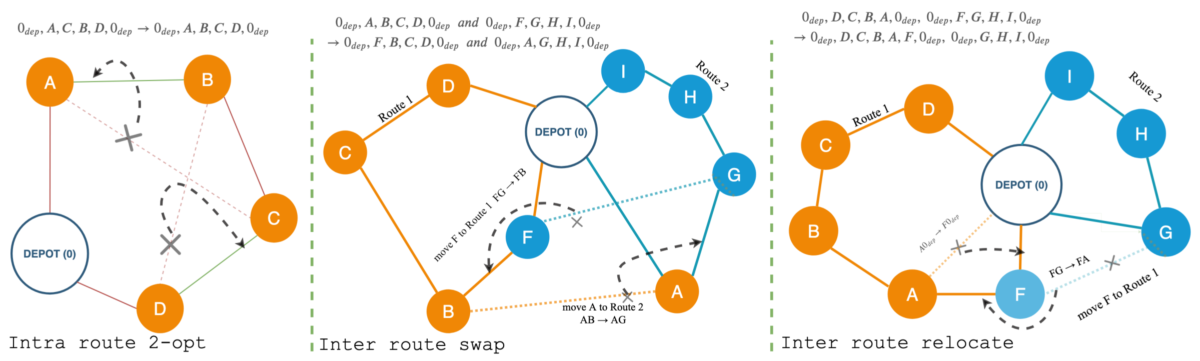
$0_{dep}, A, C, B, D, 0_{dep} \rightarrow 0_{dep}, A, B, C, D, 0_{dep}$
A
B
C
D
DEPOT (0)
Intra route 2-opt
$0_{dep}, A, B, C, D, 0_{dep}$ and $0_{dep}, F, G, H, I, 0_{dep}$
$\rightarrow 0_{dep}, F, B, C, D, 0_{dep}$ and $0_{dep}, A, G, H, I, 0_{dep}$
Route 1
Route 2
DEPOT (0)
move F to Route 1 FG → FB
move A to Route 2
AB → AG
Inter route swap
$0_{dep}, D, C, B, A, 0_{dep}, 0_{dep}, F, G, H, I, 0_{dep}$
$\rightarrow 0_{dep}, D, C, B, A, F, 0_{dep}, 0_{dep}, G, H, I, 0_{dep}$
Route 1
Route 2
DEPOT (0)
$A0_{dep} \rightarrow F0_{dep}$
FG → FA
move F to Route 1
Inter route relocate

Figure **4.2.1** Three neighborhood structures were used in all parallel models.

the maximum CPU time allowed before termination, the maximum number of shaking iterations, and the number of neighborhood structures.

4.2.2 Best vs First Improvement

Best and first improvement are strategies that provide the means to decide the trajectory path. The first strategy will accept only the best possible improvement within a neighborhood structure; thus, the trajectory will take the steepest descent. On the contrary, the latter strategy referred to as the first improvement immediately picks the first descent that improves the solution.

There are many examples of both in the literature. Usually, the first improvement strategy is selected when neighborhoods are vast because finding the best solution within a large neighborhood structure may be time-consuming. Experience seems to indicate that if the initial solution is chosen at random, the first improvement rule yields better results, but if a constructive heuristic is used, then the best improvement generates better solutions (Hansen et al., 2009).

In the proposed GVNS models, the steepest descent (best improvement) was applied. The decision of adopting this strategy was based on several tests and prior experience from the literature.

4.2.3 Shaking

GVNS uses the shaking procedure based on stochastic transformations of the best solution, in order to escape from local optimum solutions. In each iteration, the shaking procedure applies a number of random jumps in a randomly selected neighborhood from a predetermined collection of neighborhoods. The pseudo-code is presented in Algorithm 13. The maximum number of random jumps is set to six, ($k_{max} \simeq 6$). After applying a randomly selected neighborhood k times (from a set of three neighborhoods) in the incumbent solution, a new solution S' is returned.

Algorithm 13: Pseudo code for **Shaking** algorithm

input : S -incumbent solution, k: number of random jumps, l_{max}: max number of neighborhoods
output: new solution S'

```
// Randomly selected neighborhood (from the total $l_{max}$ = 3 neighborhoods)
$l \leftarrow randominteger(1, l_{max})$;
// Execute k times ($1 < k < k_{max}$)
for $i = 1$ to $k$ do
    $switchcase(l)$;
    $case(1)$;
    $S' \leftarrow Intra2Opt(S)$;
    $case(2)$;
    $S' \leftarrow InterSwap(S)$;
    $case(3)$;
    $S' \leftarrow InterRelocate(S)$;
    end switch
end
$return \leftarrow S$;
```

CHAPTER 5

Parallelization of VNS

5.1 Parallel strategies for the VNS

Recently, the rise of multi-core processors along with cluster and grid computing popularity attracted several researchers to design and develop new parallel metaheuristics. Parallel processing is considered to be a cost-effective method for the fast solution of computationally hard problems. Except for the availability of computing resources, the use of parallel metaheuristics has also increased since, as it has been demonstrated by Bouthillier and Crainic (Le Bouthillier and Crainic, 2005), such parallel algorithms are capable of both speeding-up the search and improving the robustness as well as the quality of the solutions obtained.

The parallelization of the VNS serves the purpose of either obtaining an improvement in performance or an increase of exploration space. Parallelization strategies for trajectory-based metaheuristics (section 3.5.1) may be classified into one of the two categories (Shi and Zhang, 2018):

a. Low-level parallelism

The goal of this approach is to accelerate a sequential metaheuristic by executing in parallel computing-intensive tasks. Such tasks frequently include the fitness evaluation of each solution (Figure **5.1.1**) and neighbor evaluation, using a thread per neighborhood (Figure **5.1.2**) or even multiple threads per neighborhood, thus dividing possible solutions of the neighborhood among the available processors to look for the best one (Algorithm 14).

This strategy is usually realized using a master-slave topology. The sequential method is executed on one processor (the master) while tasks are being dispatched to the other processors (the slaves). The master process assembles and combines the results yielded by the slave processes and then resumes the sequential algorithm. This approach barely aims to speed up computations, without any effort to achieve a better exploration. The parallelization of the local search in the sequential VNS is denoted as Synchronous Parallel VNS (SPVNS) as seen in Algorithm 14 given by Moreno-Pérez et al. (2004).

b. High-level parallelism or **multiple search strategy**

In this strategy, parallelism is achieved by utilizing various concurrent explorations of the solution space. Several independent search processes are used. Every process defines a unique trajectory in the search space from the same or a different initial solution point (Figure **5.1.3**).

Search processes may communicate during the search to exchange useful information or at the end to identify the best overall solution. When processes communicate during the search, the method is called cooperative multi-search method. Communications may be performed synchronously or asynchronously and may be event-driven or executed at predetermined or dynamically decided moments.

Algorithm 14: Pseudo code for the Synchronous Parallel VNS (**SPVNS**)

```
output: the best solution
initialize(best_sol);
$k \leftarrow 0$;
while $k < k_{max}$ do
    $k \leftarrow k + 1$;
    $cur\_sol \leftarrow Shake(best\_sol,\ k)$;
    // Parallelize local search to balance load among the processors
    par_loc_search(cur_sol);
    if $improved(cur\_sol, best\_sol)$ then
        $best\_sol \leftarrow cur\_sol$;
        $k \leftarrow 0$
    end
end
```

Algorithm 15: Pseudo code for the Parallel Local Search (**PLS**)

```
input  : $sol$, $cur\_sol$
output: the best solution
$init\_sol \leftarrow cur\_sol$;
while ($improved()$) do
    // Create chunks of work based on the cardinality of solutions
    $load \leftarrow \frac{n-p}{numproc}$;
    // parallel $(pr \leftarrow 0; pr < numproc; pr++)$
    $tmp\_sol(pr) \leftarrow init\_sol$;
    $low \leftarrow pr * load\ high \leftarrow low + load$;
    while $i <$ $high$ do
        $i \leftarrow i + 1$; while $j < p$ do
            $j \leftarrow j + 1$;
            $exchange(initsol, newsol, i, j)$
            if improve(new_sol,   tmp_sol(pr)) then
                $tmp\_sol(pr) \leftarrow new\_sol$
            end
        end
        // critical
        if improve(tmp_sol(pr),   cur_sol) then
            $tmp\_sol(pr) \leftarrow new\_sol$
        end
    end
    // End parallel
end
```

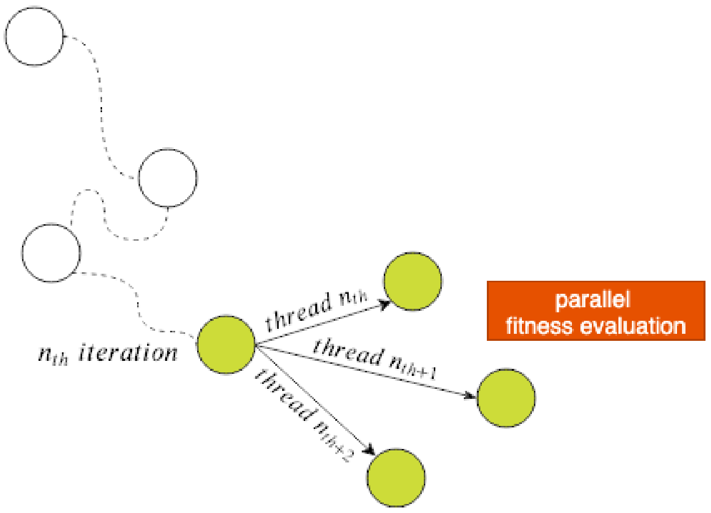

Figure **5.1.1** Parallel fitness evaluation of a solution

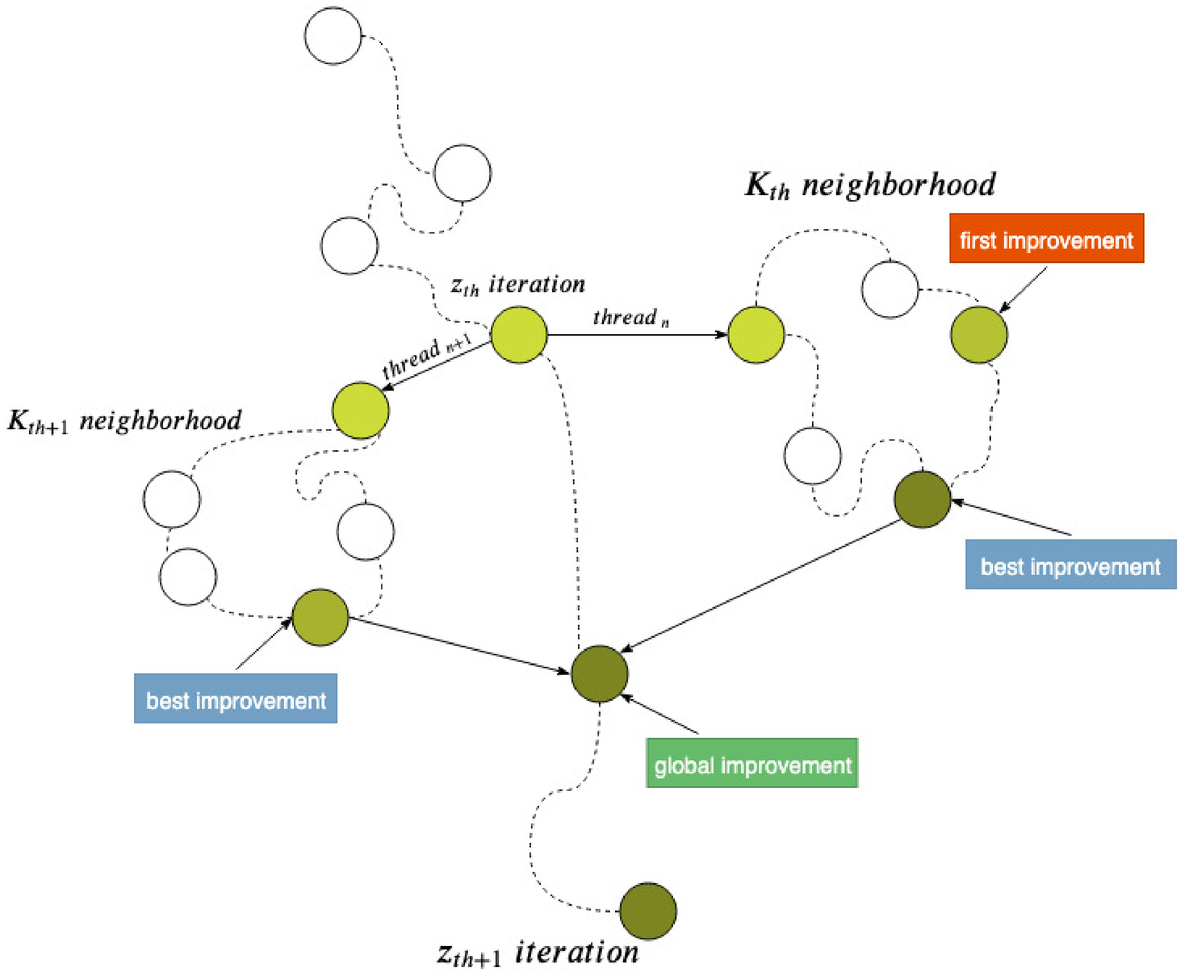

Figure **5.1.2** Parallel neighborhood exploration. In this example best improvement is used.

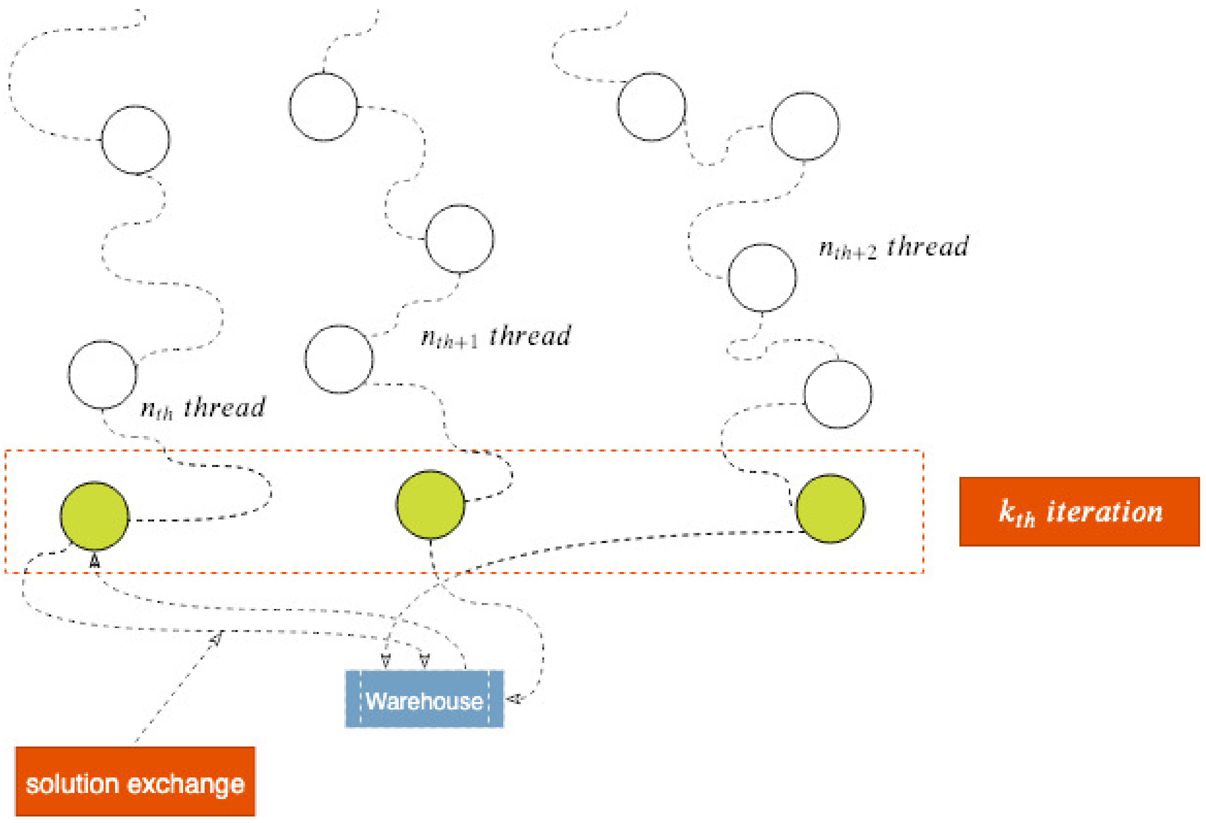

Figure **5.1.3** Various concurrent explorations of the solution space

5.2 Related works in parallel VNS for the VRP

The VNS algorithm is a trajectory-based metaheuristic. The evaluation of constraints and objective components for each solution in the neighborhood in these kinds of algorithms is an embarrassingly parallel task (Schulz et al., 2013). Thus, several strategies for parallelizing a VNS algorithm have been already proposed and analyzed in the literature (Pérez et al., 2005). Generally, contributions to the VRP using parallel metaheuristics published before the year 2000 are not as numerous as for other combinatorial optimization problems, as seen in Table **5.1**. Besides the fact that the VNS is an embarrassingly parallel task, one is surprised to realize that even fewer works have targeted the VRP using VNS, as seen in Table **5.2**. For more insight into the literature of solving VRP using additional metaheuristics methods, the interested reader is referred to the work by Crainic (Crainic, 2008).

Table **5.1** Parallel metaheuristics for VRP problem

Related work	Parallel Metaheuristic	VRP Problem
(Bouthillier and Crainic, 2005)	Tabu Search, E.A., Post-optimization	VRPTW
Bouthillier et al. (2005)	Tabu Search, E.A., Post-optimization, Pattern identification	VRPTW
Polacek et al. (2008)	VNS	MDVRPTW
Subramanian et al. (2010)	ILS - RVND	VRPSPD
Cordeau and Maischberger (2012)	ILS - Tabu Search	VRP
Coelho et al. (2012)	VNS	SVRPDSP
Jin et al. (2014)	Tabu Search	VRP
Lahrichi et al. (2015)	Integrative Cooperative Search	MDPVRP
Wang et al. (2015)	Simulated Annealing	VRPSPDTW
Tu et al. (2017)	ILS	VRP
Polat (2017)	VNS	VRPDP

Table **5.2** Parallel VNS metaheuristic applied in several problems

Related work	Parallel Metaheuristic	Problem
García-López et al. (2002)	VNS	P-median
Crainic et al. (2004)	VNS	P-median
Aydin and Sevkli (2008)	VNS	Job shop scheduling
Polacek et al. (2008)	VNS	MDVRPTW
Coelho et al. (2012)	VNS	SVRPDSP
Polat (2017)	VNS	VRPDP
Antoniadis and Sifaleras (2017)	VNS	Inventory optimization problems

5.2.1 Parallel VNS for the VRPTW

Polacek et al. (2008) introduced the first cooperative and adaptive implementation of a VNS for the multi-depot Vehicle Routing Problem with Time Windows (VRPTW). The authors provided two approaches for a parallel VNS. The parallelization approach of the two models is an extension of the Cooperative Parallel Variable Neighborhood Search (CNVNS) by Crainic et al. (2004).

The worker processes communicate exclusively with the master process which operates as the central memory that stores and manages the best found solutions. This allows an asynchronous cooperation of individual processes. Each process runs exclusively on one processor and does not share any resources with other search threads. The differentiation between the two schemes was achieved by applying different parallelization strategies and by changing the configuration between cooperation.

The initial solution was constructed by attaching each customer "C_i" to the nearest depot. Then all customers associated with a depot are ordered with respect to increasing centers of their time windows. The initial solution for both models is not necessarily feasible. The creation of a feasible solution from the initial solution is charged to the VNS iterations that follow. The starting solution of each thread is randomly chosen from the 10 best solutions the solution warehouse has stored, amplifying the diversification of the search.

In the first approach, a coarse-grained cooperation scheme was introduced. The VNS threads communicated their best solutions to the central memory (serving as a "master"). In this approach, every thread has to search through a certain number of neighborhoods. The number of iterations performed by each worker is vastly higher than the number of neighborhoods. The solution warehouse (master process) can accept a non—improving solution if σ iterations have passed without improvement. For this ascending move to occur, the value of the non-improving solution should not exceed θ percent of the value of the best-found solution. If improving solutions are found, the iteration counter for allowing an ascending move gets reset each time. This independent search favors exploration and, as the authors report, reduces the communication with the master process, since at least $2 * \sigma$ iterations are made between communications.

In the second approach, a fine-granularity was realized. The key concept behind the fine-grained cooperation scheme was to produce a parallel VNS, which maintains the successful properties of the sequential VNS, that is to reproduce the way the sequential method works, only faster. Every worker does not necessarily conclude the whole set of neighborhoods in one worker task. More precisely, every k_{max} iterations a working process sends its new best solution value to the master process.

For both models, communication occurred at preset regular intervals determined by the number of iterations (based on σ and k_{max}).

In order to report the contribution of cooperation between the two proposed VNS models, the authors implemented the RPVNS introduced by García-López et al. (2002). The RPVNS does not use any form of cooperation between the individual search threads. Every thread performs a complete VNS run.

The performance was satisfying, but the second approach performed best, notably due to its higher adaptability to the problem instance.

When the problem instances are known in advance and suitable parameter configuration can be made, then the fine-grained cooperation scheme is more suitable. For real-world problems which cannot be analyzed in detail before applying the search, the coarse-grained cooperation scheme is better because no a priori parameter tuning is required.

Both cooperation algorithms displayed satisfying run-time scalability. Specifically, utilizing 32 search threads, the runtime was reduced from 48.7 to 1.7 hours, also the best-known solution was obtained for all 20 MDVRFTW instances, and in 11 cases new best solutions were found.

5.2.2 Parallel VNS for the SVRPDSP

Coelho et al. (2012) proposed an approach for the Single Vehicle Routing Problem with Deliveries and Selective Pickups based on a Hybrid General VNS (HGVNS). The parallel solution is heterogeneous since it is implemented in CPU and GPU and is called HP-HGVNS. The incentive behind this approach is based on the decomposition & distribution of the workload. Since the most expensive part of GVNS is to perform the best improvement local search in the VND method, the GPU is utilized in order to take advantage of the fine-grain parallelism a GPU can offer.

The proposed algorithm based on VNS also includes a high-quality initial solution generator, an integrated CPU–GPU diversification mechanism and four different GPU neighborhoods. The authors names the Four-neighbourhood variable neighbourhood search (FN-VNS). The four different neighbourhood structures used are: *Swap*, $2 - Opt$, $1 - OrOpt$ and $2 - OrOpt$.

Algorithm 16: Pseudo code for Four-Neighborhood VNS algorithm (**FN-VNS**)

```
input  : $iter_{max}$ -iterations without improvement, $CD$: delivery customers, $CP$: pickup
         customers, $f(.)$: evaluation function, $Nk(.)$: neighbourhoods, $M$: cost matrix, $Q$: vehicle
         capacity, $d$: deliveries, $p$: pickups, $r$ : revenues
output: the best solution
1  $S \leftarrow InitialSolution(CD, CP, M, Q, d, p, r)$;
2  while $iter \leq iter_{max}$ do
3      $S' \leftarrow S$;
4      // In order to escape from deeper local optima,
5      // the number of consecutive moves can vary randomly
6      $l \leftarrow random\ number \left[\frac{|C_D|+|C_p|}{2}\right]$ ;
7      // Shake
8      for $i = 1$ to $l$ do
9          $k \leftarrow random\ number\ [1, 4]$;
10         $S'' \leftarrow random\ Neighbor\ from\ N_k(S')$;
11         $S' \leftarrow S''$;
12     end
13     for $k = 1$ to $4$ do
14         $R \leftarrow LocalSearch(S'', f(,), N_k)$ ;
15         // Best Improvement
16         $S^* \leftarrow BI(R, S'', N_k)$;
17         if $f(S^*) < f(S))$ then
18             $S \leftarrow S^*$;
19             $k \leftarrow 1$;
20             $iter \leftarrow 0$;
21         end
22     end
23     $iter \leftarrow iter + 1$;
24 end
25 $return \leftarrow S$;
```

As shown in Algorithm 16, the computational cost of the search process is strongly dominated by the local search (line 12 of Algorithm 16). This is justified by the complexity of the neighborhood structures, which is $O(N^3)$, considering that it is required to perform an $O(N)$ feasibility test for each of the $O(N2)$ moves of each neighborhood.

The authors parallelized the most computationally demanding part of the code. The principal idea is to exploit the massively parallel environment, the high memory bandwidth, and the efficient thread scheduling mechanism of the GPU, to reduce the computational times and to allow tackling larger problem instances. The CPU, on the other hand, is used for creating an initial solution, choosing the operation to be performed, and checking for the best solution generated by the GPU.

The results are compared against other SVRPDSP solutions in literature provided by Bruck et al. (2012). Concerning performance, the $1Or-Opt$ and $2Or-Opt$ local search scored a 27.26 and 43.75 speedup respectively, while the Swap local search had the lowest speedup values, from 1.91 to 14.36. HP-HGVNS achieved an average speedup from 2.73 to 16.23. Dissecting the quality of the results, HGVNS produced better solutions than the best results obtained in literature. FNVNS obtained 51 new better solutions and 7 equal solutions, out of the 68 instances used with an average gap of 5.32% and variability from the average solutions equal to 1.55%. Other recent works with hybrid CPU-GPU parallel VNS methods include that of Antoniadis and Sifaleras (2017).

5.2.3 Parallel VNS for the VRPDDP

Polat (2017), proposed a parallel metaheuristic algorithm for the Vehicle Routing Problem with Divisible Deliveries and Pickups (VRPDDP).

In order to construct different initial solutions for each processor, an extension of the Savings algorithm by Altınel and Öncan (2005) is used. By changing the parameter $\lambda \in [0,5]$, $\mu \in [0,3]$, $\nu \in [0,2]$ of the Savings algorithm extension, various initial solutions for each processor are constructed.

The improvement of the initial solution is achieved by using a number of intra and inter-route neighborhood structures commonly used in the literature. The 8 neighborhood structures used in the VNS are the following:

a. Intra route neighborhood search operators:

- **i.** **swap**, a random position change movement of two customers on the same route.
- **ii.** **2-opt**, swaps pairs of edges in the same route
- **iii.** **3-opt**, deletes three edges in a route and reconnects them in the same route
- **iv.** **Insertion**, randomly selects a customer and moves it to a random position on the same route.

b. Inter route search operators:

- **i.** **exchange(m,n)**, transfers m sequential customers from one route to another and in turn transfers n sequential customers from the second route to the first
- **ii.** **Cross**, exchange is a basic crossover structure between routes
- **iii.** **Shift(0,1)**, is a random movement of a customer from one route to another
- **iv.** **Replace(1,1)**, Is a random permutation movement between two customers from different routes

The proposed VNS scheme accepts a new solution only if there is an improvement of the incumbent solution. Once the region of the current solution has been extensively explored, in order to avoid local optima and to achieve diversification, the author proposed a shaking scheme with two neighborhoods, which are comprised of two sequential $Replace(1,1)$ operations (double Replace perturbation operators).

In this approach, the author used asynchronous cooperation with a centralized information exchange strategy for the parallelization of the VNS (CVNS). More particularly, the star-architecture-based multiprocessor cooperation network was applied, which uses several computation processors and a user interface processor.

Based on Crainic and Hail taxonomy (Crainic and Hail, 2005), this approach is classified as pC/C/MPSS. This strategy is characterized as medium-grained parallelization where data exchange is only performed after the completion of all local search procedures. Communication takes place only after all local search procedures are completed.

The benchmark instances used for the VRPSPD and the VRPDDP were the VRP benchmark problems of Christofides et al. (1979) by Salhi and Nagy (1999) and Nagy et al. (2015). Examining the quality of the results, solutions were improved on or reached the best-known solutions for 179 of 220 benchmark instances while using less computational time. When compared to the Parallel VNS (PVNS), the CVNS approach provides better solutions for 10 of 28 test instances and the same results for the remaining instances within almost half the solution times of the PVNS. Overall improvement of the CVNS over the PVNS is just around 0.14% (0.11% for the instances without a time limit and 0.16% for the time-limited instances).

CHAPTER 6

The proposed GVNS parallelization models

In this section, we present three parallel GVNS models. The main differentiation of the three models is the communication scheme among the search threads.

There are three main steps in parallel algorithm design: the decomposition of the workload, the distribution of the tasks to the available processors and perhaps the most crucial step, the coordination strategy of the components of the parallel program. In our approach, we materialize three scenarios. Our intent is to study how the level of coordination between the components of the parallel VNS algorithm affects performance.

To describe parallel metaheuristic strategies, we adopt the classification of Crainic and Hail (2005) that generalizes that of Crainic et al. (1997). The classification consists of three dimensions that indicate how the global process is controlled (search control cardinality), how information is exchanged among processes (search control and communication), and the variety of solution methods involved in the search for solutions (search strategies) (Crainic, 2008).

6.1 Classification of parallel metaheuristics

In order to specify the differences between the models, we classify them using the taxonomy by Crainic and Hail (2005), as shown in the Table **6.1**. The taxonomy is based on how the process is controlled and how information is exchanged among the processes.

Table **6.1** Parallel process taxonomy

Search Control Cardinality	Search Control and Communications	Search Differentiation
how the global search is controlled	*how information is exchanged*	*do threads start from the same or different solutions*
1-control (1C)	Rigid (RS)	Same initial Point Same Search Strategy (SPSS)
polyControl (pC)	Knoweledge Synchronization (KS)	Same initial Point Different Search Strategy (SPDS)
-	Collegial (C)	Multiple initial Point Same Search Strategy (MPSS)
-	Knowledge Collegial (KS)	Multiple initial Point Different Search Strategy (MPDS)

6.2 The Non-Cooperative Model (NCM)

The proposed implementation employs an island-based scheme where every thread runs the GVNS completely isolated. When the initial solution is generated using the Clarke and Wright algorithm, it is given to each thread. All threads are using the same search strategy (GVNS). Soon after, the GVNS algorithm begins executing without further communication between threads. As threads work independently and possible solutions are improved in each one of them, their paths diverge as the shaking procedure takes place. As the time limit is reached, all threads terminate, and the best solution among all is reported. Also, in the occurrence of the optimal solution acquisition by one thread, the stopping condition is activated, and that causes all threads to break operation.

Based on the classification of Crainic and Hail (2005), the non-cooperative model fits into the pC/RS/SPSS classification. The notation pC stands for “poly-Control”, corresponds to the Search Control cardinality, and indicates that the search control is distributed across many processes. In this model, every single thread has its own search control. The notation RS stands for “Rigid Synchronization” and it means that little or no information exchange takes place in order to improve solutions at the same level of the communication hierarchy. The notation SPSS stands for “Same initial Point, Same search Strategy” and it indicates that all threads had the same initial solution as a starting point and followed the same search strategy. Every single thread executes the code shown in the Algorithm 17. Figure **6.2.1**, shows the execution time line of this non-cooperative model.

Algorithm 17: Pseudo code for the GVNS, Non-Cooperative model (**NCM**)

```
// $fr$: Current best solution route
// $kmax$: Shake k parameter
// $timelimit$: The time limit stopping criteria
input : $fr$, $kmax$, $timelimit$
while $true$ do
    $t \leftarrow CpuTime()$
    $fr' \leftarrow Shake(fr, kmax)$
    $fr'' \leftarrow VND(fr', t, timelimit)$
    if $t > timelimit$ then
        $break$
    end
    // if optimum value exists
    if $best\_value = optimum$ then
        break
    end
end
```

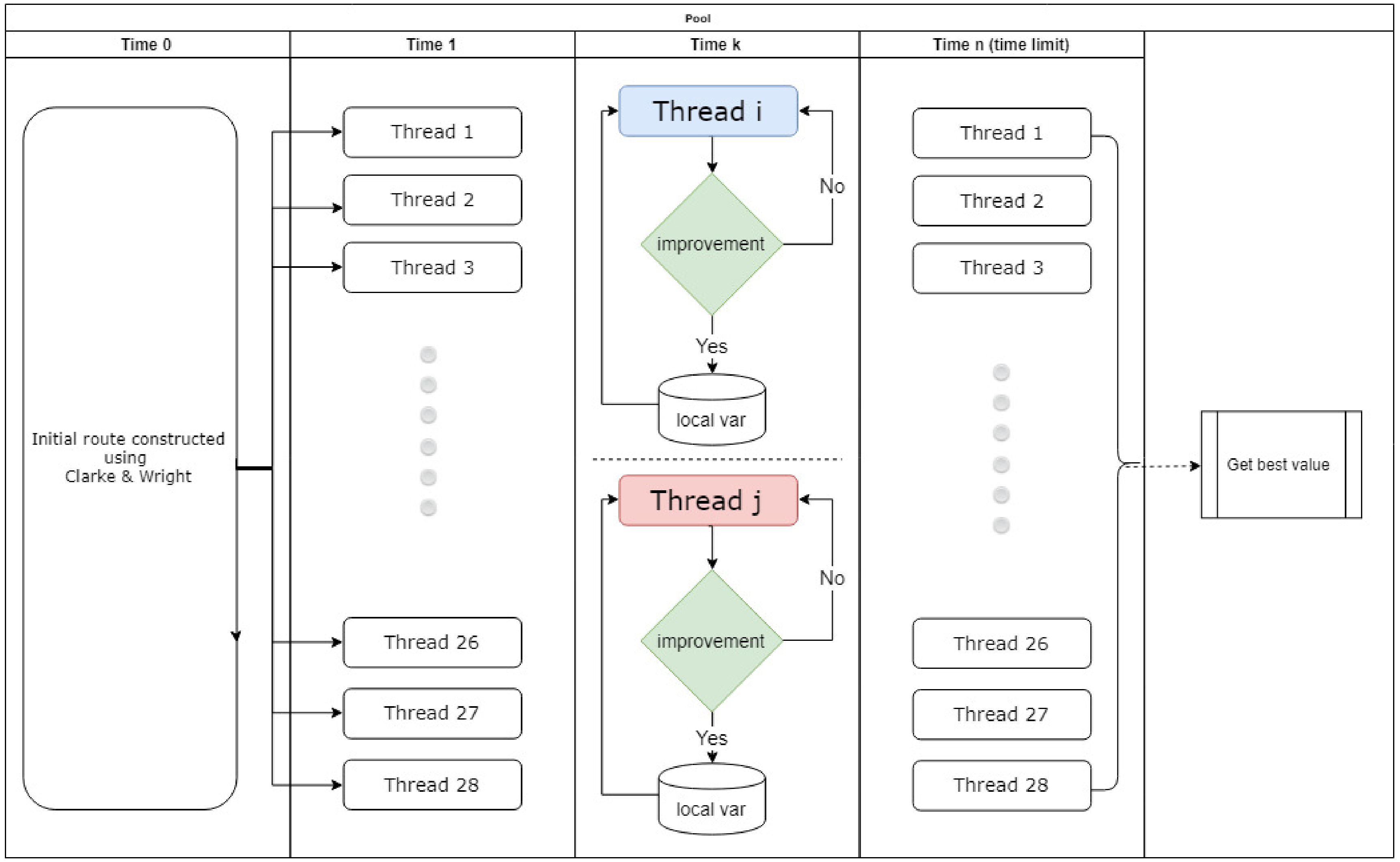

Figure **6.2.1** The non-cooperative model execution timeline.

6.3 The Managed Cooperative Model (MCM)

The first solution produced by the Clarke and Wright algorithm is passed to all threads, except for one, which will take up the role of the server manager, essentially being a solution warehouse. When a single thread improves a solution previously acquired, it communicates with the server manager in order to ask if a better solution exists. In the case that the server manager holds a better solution, the thread rejects the inferior one and continues its work with the one provided. Otherwise (i.e., if the thread holds a better solution than the server manager), the new and improved solution is stored in the server manager and the thread continues with the one it found.

The server manager stores and manages the best-found solution without broadcasting any information to the other threads. Communication between the threads and the server manager is very dense at the start of the GVNS algorithm, promoting intensification, and sparse while the process continues, favoring diversification.

The cooperative model fits into the $pC/C/SPSS$ classification. Every dimension in this taxonomy is identical to the non-cooperative model except for the C class, which stands for “Collegial” and indicates that we extract and use only good solutions from the Manager. Figure **6.3.2** depicts the execution time line of the server manager cooperative model.

The cooperative model exchanges information asynchronously, at irregular intervals dynamically determined by each process. The search process may change its current solution using a solution that is received by the Manager, and for this reason, the intensification level dynamically changes. Since no broadcasting occurs, information will not spread any further.

Algorithm 18: Pseudo code for the GVNS, Managed Cooperative Model (**MCM**)

```
input : $fr$, $kmax$, $timelimit$, $manager$

while $true$ do
    $t \leftarrow CpuTime()$
    $fr' \leftarrow Shake(fr, kmax)$
    $fr'' \leftarrow VND(fr', t, timelimit)$
    if $thread\_solution < thread\_current\_best$ then
        $thread\_current\_best \leftarrow thread\_solution$
        // if optimum value exists
        if $thread\_current\_best < manager\_global$ then
            $manager\_global \leftarrow thread\_current\_best$
        end
        else
            $thread\_current\_best, thread\_solution \leftarrow manager\_global$
        end
    end
    if $t > timelimit$ then
        $break$
    end
    // if optimum value exists
    if $manager\_global = optimum$ then
        break
    end
end
```

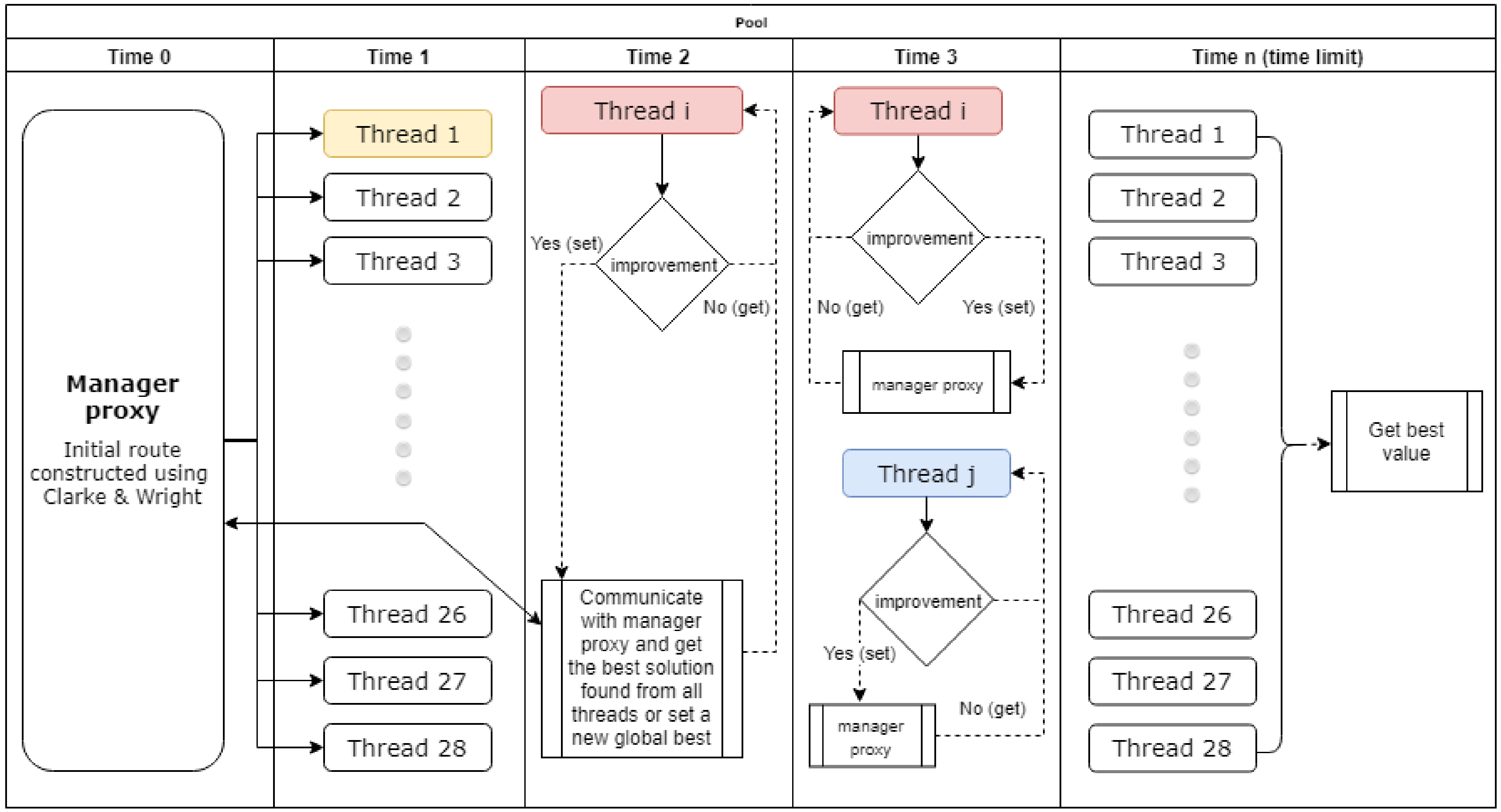

Figure **6.3.2** Managed Cooperation model execution time line.

6.4 The Parameterized Cooperative Model (PCM)

The previous models were each at the other end of the spectrum of the communication strategy. The cooperative model favors intensification without any insight of the search space, while the non-cooperative model promotes exploration. In order to bridge the gap, we consider a parameterized cooperative GVNS model. This time, the cooperative model will accept a solution from the server manager only if the normalized distance between the two solutions is greater than a parameter θ ($distance > \theta$). The distance measure adopted in this dissertation is the Damerau – Levenshtein distance (Damerau, 1964), also termed as edit or Levenshtein distance.

This metric can be described as a search space distance rather than a solution space distance due to the fact that the edit operations applied in the solution representation can be considered to be neighborhood operations. The Figure **6.4.3** illustrates the normalized Damerau – Levenshtein distance calculation between two solutions for a CVRP instance.

In the classical capacitated vehicle routing problem, the solution representation consists of multiple permutations, representing the set of clients and the set of routes. In this thesis, in order to encode a CVRP solution and calculate the distance between two solutions, we assume reversal independence, id est the distance between two routes when reversing one route should be zero.

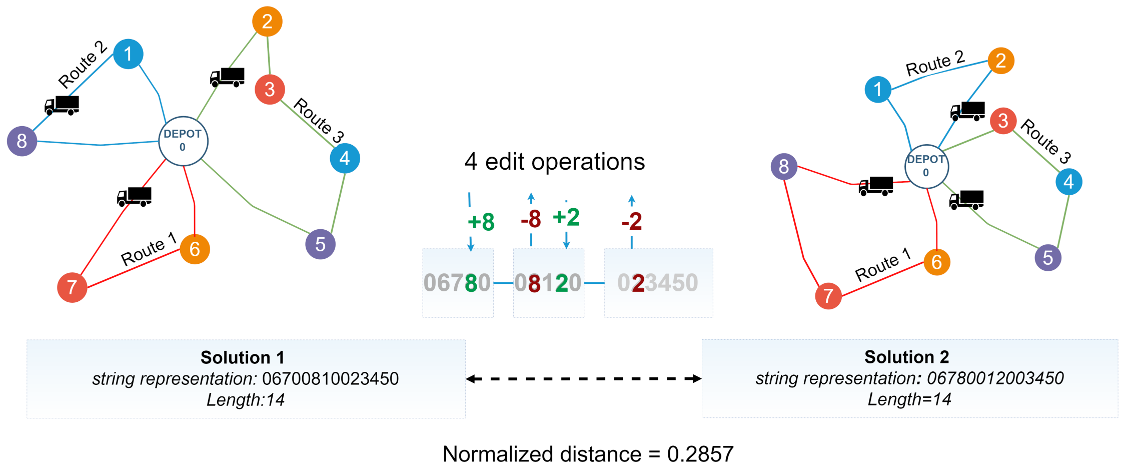

Figure **6.4.3** Levenshtein distance for solutions that are two inter-route relocations apart.

The algorithm returns a normalized float (between 0.0 and 1.0) by evaluating Equation 1.

$$Normalized\ distance = \frac{distance}{maxLength} \tag{1}$$

As seen in Figure **6.4.3** the solution of the CVRP is represented as an array list $\sum_{i=1}^{k}\sum_{j=1}^{n}(C_{ij}+2)$, where k is the number of vehicles, C symbolizes the n the clients that are assigned to route i and 2 depicts the start and the endpoint (the depot) of the route i, thus forming a single string of characters.

In order to calculate the normalized distance using Equation 1 we first calculate the string lengths. In our example from Figure **6.4.3**, Len_1("06700810023450")=14, Len_2("06700810023450")=14 the maximum string length $max(Len_1, Len_2) = 14$. The Levenshtein distance is equal to 4 since only four edit operations are needed to get Solution 2 from Solution 1. Using the values above we get the normalized distance equal to $\frac{4}{14} = 0.2857$.

Algorithm 19: Pseudo code for the GVNS, Parameterized Cooperative Model (**PCM**)

```
input : $fr$, $kmax$, $timelimit$, $manager$
while $true$ do
    $t \leftarrow CpuTime()$
    $fr' \leftarrow Shake(fr, kmax)$
    $fr'' \leftarrow VND(fr', t, timelimit)$
    $\theta \leftarrow dynamic\_decrease(time, iter\_count)$
    if $thread\_solution < thread\_current\_best$ then
        $thread\_current\_best \leftarrow thread\_solution$
        // Communicate with manager
        if $thread\_current\_best \leq manager\_global$ then
            $manager\_global \leftarrow thread\_current\_best$
        end
        else
            // Check if solution exchange can take place
            // Thread should adopt new solution only if distance >$\theta$
            if $\theta > distance(manager\_global, thread\_current\_best)$ then
                $manager\_global \leftarrow thread\_current\_best$
            end
            else
                $thread\_current\_best, thread\_solution \leftarrow manager\_global$
            end
        end
    end
    if $t > timelimit$ then
        $break$
    end
    // if optimum value exists
    if $manager\_global = optimum$ then
        $break$
    end
end
```

The parameter θ ranges from value 1.0 (non-cooperative) to 0.0 (cooperative). In the beginning, it promotes exploration by ignoring solutions stored in the solution warehouse ($distance \leq \theta = 1$), and as the time passes, θ value gradually decreases and the parameterized cooperative method starts to accept values even with small error distances. In our tests, a cut off value (δ) was selected either at the $\delta \leftarrow 20\%$ of the max GVNS iterations ($niter_{\max}$) or at the maximum time ($time_{\max}$) allowed.

In order to find the optimal cut-off value of δ, we performed several tests, spanning from 10%

to 40% of the max iterations or the max time. The best results for all instances were obtained around 20%. Thus, the self-adaptation of the parameter θ is based on the remaining time and GVNS iterations, as depicted in Algorithm 20. The main concept when selecting δ is to block the communication when the algorithm starts the search and the new solutions are more easily found. As a result, this increases the exploration.

In that sense, the Managed Cooperation Model has a $\theta \leftarrow 0$, thus every single solution gets accepted and the non-Cooperative Model on the opposite end, has a $\theta \leftarrow 1$ and consequently no solution gets accepted.

Algorithm 20: Pseudo code for the θ parameter self-adaptation

```
input  : Current time (time), current number of iterations (niter), $\delta$ preset to 0.2
output: $\theta$
// When $\theta \leftarrow 1$ no communication takes place
$\theta = 1$ ;
// Checking the two stopping criteria (number of iterations and exeution time)
if $(niter_i > \delta \times niter_{\max}) \vee (currenttime > \delta \times time_{\max})$ then
    // Gradually decrease $\theta$ value
    $\theta = \min\left((1 - \frac{niter}{niter_{max}}), (1 - \frac{time}{time_{max}})\right)$;
end
```

The parameterized cooperative model fits into the pC/KC/SPSS classification. KC class stands for “Knowledge Collegial” and indicates an asynchronous communication scheme in which the contents of communications are analyzed to infer good solutions.

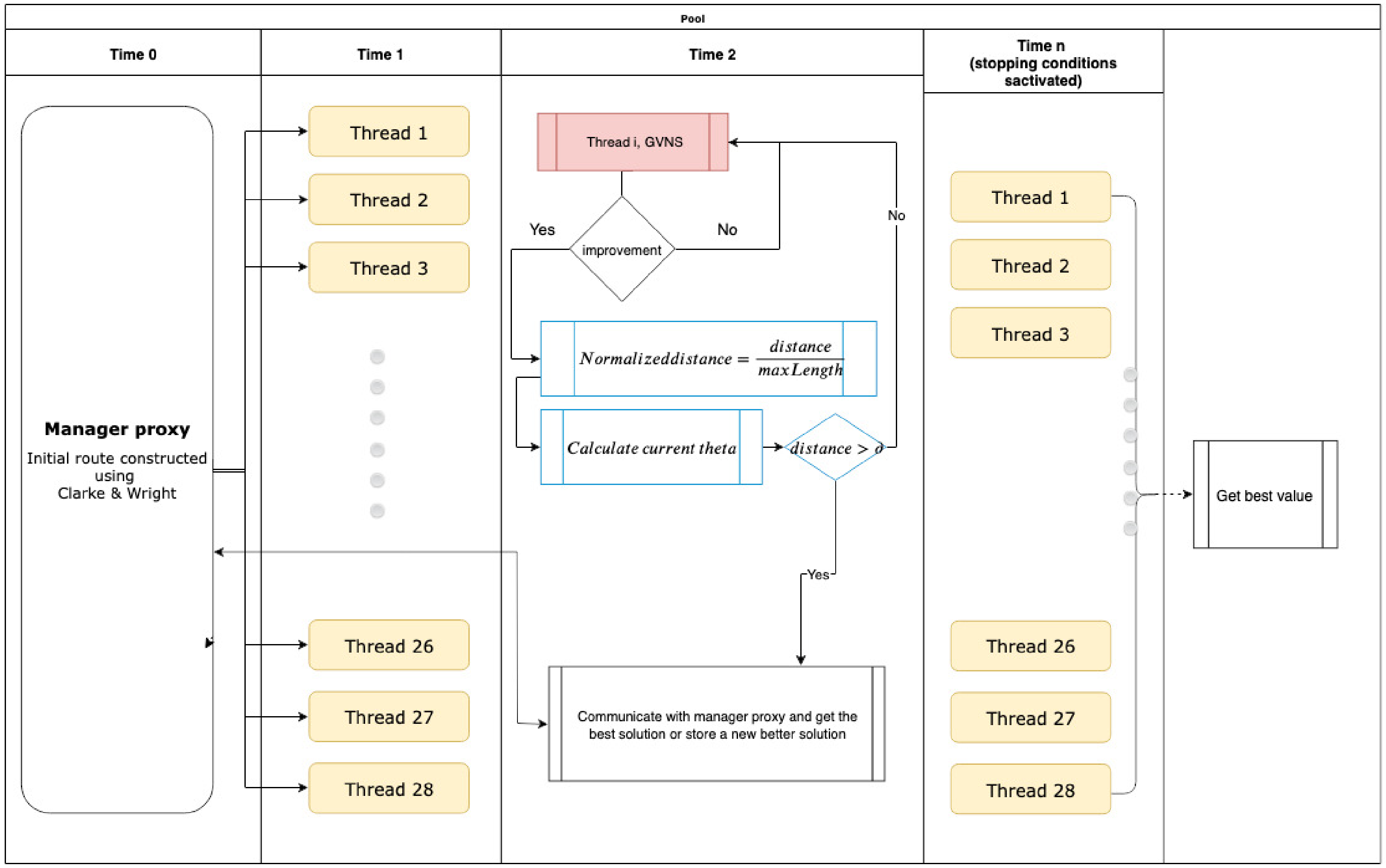
Pool
Time 0
Time 1
Time 2
Time n
(stopping conditions sactivated)
Manager proxy
Initial route constructed using Clarke & Wright
Thread 1
Thread 2
Thread 3
Thread 26
Thread 27
Thread 28
Thread i, GVNS
improvement
Yes
No
No
Normalizeddistance = distance / maxLength
Calculate current theta
distance > ∂
Yes
Communicate with manager proxy and get the best solution or store a new better solution
Get best value

Figure **6.4.4** Parameterized Cooperation model execution timeline.

6.5 Analysis of the cooperation strategies

For the needs of the proposed cooperation models, asynchronous cooperation was selected, since synchronous methods have a more significant computational cost and are unaware of the findings during the search process. Since no broadcasting takes place, each thread can dynamically guide the diffusion of information, depending on its state. The connection pattern in both the proposed communication models is undetermined and adopts a less obtrusive method, where no thread gets interrupted so to display the clear benefits of communication.

In order to better manipulate the interplay between exploration and exploitation, a parameterized cooperation is proposed. This is a fine-grained cooperation method, where a process may discard the best solution collected by the Manager, based on the iterations it has performed and its current status.

The asynchronous cooperation strategy followed in the Managed Cooperation Model and in the Parameterized Cooperation Model is novel compared to the previous works reported by Coelho et al. (Coelho et al., 2012), Polacek (Polacek et al., 2008), and Polat (Polat, 2017). Furthermore, the strategy that follows the principle of controlled diffusion of information at the process level in the parameterized cooperative model is a completely new cooperation strategy, with promising results.

CHAPTER 7

Computational study

7.1 CVRP instances

Currently, the most updated CVRP benchmark instance library can be found at CVRPlib website (Xavier et al., 2019). It contains the following instances:

- Set A (Augerat et al., 1995),
- Set B (Augerat et al., 1995),
- Set E (Christofides and Eilon, 1969),
- Set F (Fisher, 1994),
- Set M (Christofides et al., 1979),
- Set P (Augerat et al., 1995),
- Set Tai (Rochat and Taillard, 1995),
- Set Golden (Golden et al., 1998a),
- Set Li (Li et al., 2005a),
- Set X (Pecin et al., 2014),
- Arnold, Gendreau and Sörensen (2017),

Nature of CVRP instances

As shown in Table **3.1** the majority of instances of the A, B, E, F, M, P sets provided a useful benchmark until the mid-2000's decade. Since then, the Golden set became the second most frequently used benchmark (Uchoa et al., 2017b). The fact that instances with many nodes (above 100) were very thinly populated forced researchers to design new instances to address the lack of a well-established set of benchmark instances able to push the limits of the state-of-the-art algorithms. The set from Arnold et al. (2019) provides instances with very large-scale routing problems. The X set addresses three problems of the previous sets. Older sets were:

i. too easy for current algorithms,

ii. too artificial,

iii. too homogeneous, not covering the characteristics found in real applications

In order to address those issues, the X set introduced several new attributes like depot positioning, customer positioning and clustering, and customer demand values with several options. X set follows the TSPLIB convention of rounding distances, thus reducing the search space by forming a relatively small number of plateaus, but the fact that it has a $1000 * 1000$ grid instead of a $100 * 100$ (A, B, E, F, M, P) significantly reduces this effect. The new instances do not follow the practice of fixing the number of vehicles. Hence, the number K_{min} registered in each instance should be considered only as a lower bound on the number of vehicles in a solution.

Uchoa et. al performed several tests using the BCP exact algorithm (section 3.1) and the ILS metaheuristic (section 3.5.1). According to the results, instance attributes affect the tests in the following way:

a. **Depot positioning** heavily affects tests. Instances with a depot placed in the center are easier than those with a depot installed in a corner; the latter leads to more tours faced towards the same area, and thus more possibilities and complexity for intra-route improvements. The depot placement impact on the CPU time is indisputable. Instances with the depot in the corner had median times 37 times larger when compared with those with the depot placed in the center. Instances with random depot location exhibit moderate difficulty.

b. **Customer positioning** does not seem to affect the performance considerably, but clustered instances appear to be just a bit easier.

c. **Demand distribution** has a notable impact on CPU time. Based on tests using a hybrid ILS, a metaheuristic that uses multiple neighborhoods like VNS, the results showed that:

- **i.** **Unitary demands** make instances much easier. Unitary demand instances with $n = 200$ clients are solved easier than instances with $n = 100$ clients, and demands in $[5 - 10]$. Instances with larger demand values ($[1 - 100]$ and $[50 - 100]$) are harder than instances with small values ($[1 - 10]$ and $[5 - 10]$).
- **ii.** Instances with **few very large demands and many small demands** were difficult to solve, with a median time of almost 12 days
- **iii.** When customer demands have **high coefficient of variation** renders the problem instances moderately more difficult.
- **iv.** The instances with demand **dependent on the quadrant** formed the hardest of all categories. Only eight instances with these attributes could be solved in less than nine days; the remaining 12 instances were stopped with at least 13 days of CPU time.

7.2 Self adaptive parameter tuning

The design process of the GVNS models is characterized by a mixture of experience and intuition and requires an in-depth knowledge of the process characteristics. Furthermore, an essential strength of the proposed approach is that a few parameters need to be set, and tuned. These parameters are easier to estimate empirically. After several experiments, an average value of $k_{max} \simeq 6$, which was obtained by using Equation 1, provided the best results. Parameter l_{max} represents the cardinality of the neighborhoods ($l_{max} \leftarrow count(Neighborhoods)$). Parameter θ is self-adapting, and its value is periodically modified depending on factors that will be discussed in detail in section 6.4. More about parameter tuning strategies are discussed and illustrated by numerical results obtained for different instances in Chapter 7.

$$k_{max} \leftarrow count(Neighborhoods) + 2 * rand(0, count(Neighborhoods)) \tag{1}$$

7.2.1 Stopping conditions

One of the most common stopping criteria of the GVNS approach is the total execution time. Several tests were performed with multiple stopping criteria. In most of the tests, three stopping criteria have been adopted. The first was the total execution time of the algorithm (3600 seconds), the second was the completion of a number of GVNS iterations (300, 500 and 800), and the third was the achievement of the optimal value, when such value was available. Variations from the aforementioned criteria were executed and reported for statistical comparisons between the proposed parallel models.

7.3 Computational experiments

This section reports and analyzes the computational experiments related to the efficiency of the three proposed parallel GVNS models for solving the CVRP. The parallelization of the GVNS method for the proposed models utilized the standard multiprocessing library of Python 3.7. In the cooperative models, instead of using shared memory, a Manager class was used. Manager objects control a server process which handles shared objects and enables other processes to manage them using proxies. Server process managers are more flexible than shared memory objects, since they support arbitrary objects and direct manipulation from processes on the same or different computers over a network. An additional speedup (∼35%) was achieved using Cython. The experiments were conducted using an Intel Core i9 7940X CPU (3.50 GHz) with 32GB RAM. All computational tests were carried out on 191 instances of the CVRP library (Uchoa et al., 2017a) available at `http://vrp.atd-lab.inf.puc-rio.br`.

In order to evaluate the performance of the proposed models, we have focused our analysis in three parts. Firstly, we investigate model performance based on the overall quality of solutions of all instances. Secondly, we select some representative, computationally difficult instances, and check the quality of solutions. The computational difficulty is loosely defined based on the instance size (customer count). Thirdly, we focus our search on the instance properties. We analyze and rank the instances based on instance properties. This analysis contributes to a better evaluation of the effects of each communication strategy and how communication affects the quality of solutions. In order to achieve this, we used the X-set from Uchoa et al. (2017b) that provides all the needed information.

In the first part of this analysis, the effect of the cooperation strategy of each algorithm is studied on all instances from the set A and set B (Augerat et al., 1995), set E (Christofides and Eilon, 1969), and a subset of set M (Christofides et al., 1979). The selected 65 CVRP instances have a number of clients ranging from 32 to 200, known optimum number of vehicles, fixed vehicle capacity, and known best solution. Since all the instances from the sets A, B, E and M have optimal solutions, the reaching of the optimum value is considered a stopping criterion. In order to have a fair comparison and evaluate the performance and the quality of the solutions, all the experiments were repeated ten times, and an average value was reported.

In the results shown in Table **7.1**, essential differences among the compared methods can be observed. The first row refers to the average error. The parameterized model surfaces as the best strategy, producing better solutions. In the second row, we report the average CPU time needed to complete 300 GVNS iterations. The non-communicative model is the fastest method, completing the task after 76.880 seconds on average. A theoretical explanation of this speedup in the non-cooperative strategy is that communication deprives the search procedure of some CPU cycles. The parameterized model surfaces as the best strategy, producing better solutions (0.729% error). In the third row, we report the average computational difficulty, a metric measured based on the number of the GVNS iterations accomplished per second per thread. The PCM model completes a GVNS iteration in 0.182 seconds on average (a 20% increase compared to the performance of the NCM model). In the last row we report the number of the instances in which the model found the optimal solution before the completion of 300 GVNS iterations.

The full benefits of communication are not particularly evident, as the non-cooperative model produces comparable results.

Table **7.1** Comparison of the three GVNS parallel variants at 300 GVNS iterations

	NCM	MCM	PCM
mean error	0.837%	1.260%	0.729%
mean CPU time (secs)	76.880	92.100	88.640
mean computational difficulty (secs)	0.151	0.196	0.182
Optimal number of solutions	21	16	21

At first glance, even though the parameterized model produces better results, the complete absence of communication between processes appears to yield promising results. The non-cooperative model is probing a much more significant portion of the search space with the hope of finding promising solutions; thus having a lower probability of being trapped in a local optimum.

In Figure **7.3.1**, we plot the average error of each parallel model on instances with smaller size and known optimal value. The instances are grouped by their set (sets A, B, E, and M). It should be noted that, all algorithms achieved the optimum value for 14 out of 65 instances; thus, these instances have been omitted. An emerging pattern from this data visualization indicates that, the gap between the three models closes, especially when dealing with computationally harder instances.

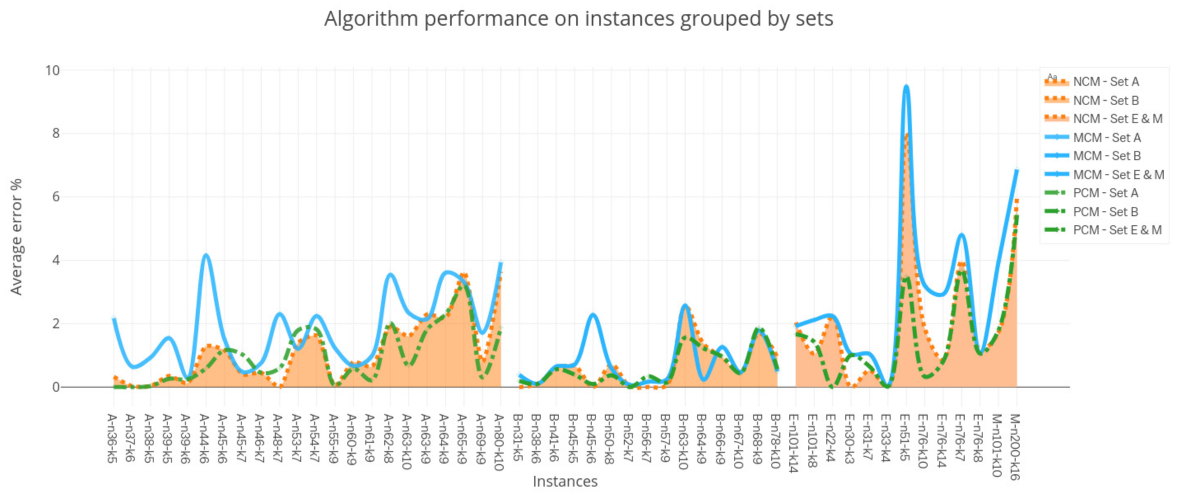

Figure **7.3.1**: Algorithm performance on instances grouped by sets. Methods supporting communication close the performance gap on computationally difficult instances.

The assumption of normality was tested with the Kolmogorov-Smirnov test. The results indicate that the distribution of the data deviates significantly from the normal distribution; therefore, the analyses proceeded with non-parametric tests. For inferential statistics, the Friedman test was applied to the performance results collected by the execution of the three models on the 65 instances. The obtained p-value was almost zero ($3 * 10^{-11}$), thus showing that there is enough statistical evidence to consider the three algorithms different. A typical significance level of $\alpha = 0.05$ was regarded as the threshold for rejecting the null hypothesis.

After the diversity of the parallel model has been statistically established, we studied the observed trend. In order to analyze the detected pattern, we selected the instances where all three algorithms produced non-zero error values within a horizon of 300 GVNS iterations. The size of every instance (number of clients) and the number of routes (number of tracks) can be used to classify the instances into three levels; easy, medium, and hard. Other factors like depot positioning, a high coefficient of variation in demands, if the clients' positions form clusters, also play an essential role in instance difficulty. The remaining 51 instances were classified based on their computational difficulty, forming four categories as follows:

- Easy (A-n36-k5, A-n37-k6, A-n38-k5, A-n39-k5, A-n39-k6, A-n55-k9, B-n31-k5, B-n34-k5, B-n45-k6, B-n52-k7, B-n57-k9, E-n22-k4, E-n33-k4),
- Medium (A-n44-k6, A-n45-k6, A-n45-k7, A-n46-k7, A-n48-k7, A-n53-k7, A-n54-k7, A-n60-k9, A-n61-k9, A-n62-k8, A-n63-k10, A-n63-k9, A-n69-k9, B-n38-k6, B-n41-k6, B-n43-k6, B-n45-k5, B-n50-k8, B-n56-k7, B-n64-k9, B-n66-k9, B-n78-k10, E-n31-k7, E-n51-k5),
- Hard (A-n64-k9, A-n65-k9, A-n80-k10, B-n63-k10, B-n67-k10, B-n68-k9, E-n76-k7, E-n76-k8, E-n76-k10, E-n76-k14, E-n101-k14, E-n101-k8, M-n101-k10, M-n200-k16)

- Very Hard, eleven instances from the Golden set (Golden et al., 1998b).

This classification was loosely based on the instance size (customer count). Table **7.2** summarizes the results and makes it easier for one to observe the effect of communication on instances with greater difficulty.

Table **7.2** Comparison of the three parallel GVNS algorithms on all instances classified by difficulty.

Instance Class	Model	Avg. Error %	GVNS iterations
Easy	NCM	0.042%	300
	MCM	0.351%	300
	PCM	0.103%	300
Medium	NCM	0.990%	300
	MCM	1.731%	300
	PCM	1.059%	300
Hard	NCM	2.043%	300
	MCM	3.386%	300
	PCM	1.978%	300
Very hard	NCM	9.620%	500
	MCM	9.380%	500
	PCM	8.270%	500

Within every category, the well-known Wilcoxon non-parametric test was performed for pairwise comparisons, in order to analyze whether the three parallel models produce different results or not. In order to have a fair comparison and evaluate the performance and the quality of the solutions, each parallel model was executed ten times, with each execution thread reaching 50, 100, 200 and 300 GVNS iterations. The results from pairwise Wilcoxon tests are presented in the following Table **7.3**.

Table **7.3**: Results from pairwise Wilcoxon tests, where the groups of algorithms that differ significantly are underlined.

Parallel GVNS models	Easy p-value	Medium p-value	Hard p-value	Golden p-value
NCM vs MCM	<u>0.014</u>	0.089	<u>0.021</u>	0.678
NCM vs PCM	0.944	0.409	<u>0.002</u>	<u>0.000</u>
MCM vs PCM	<u>0.032</u>	<u>0.014</u>	<u>0.032</u>	<u>0.009</u>
Best method	NCM	PCM, NCM	PCM	PCM

The resulting p-values obtained from the Easy class analysis demonstrate that the managed cooperative model differs significantly from both the non-cooperative and the parameterized model when solving easier instances. The non-cooperative method delivers better results in this category. The unfiltered cooperation method performed poorly while, on the other hand, the parameterized cooperation yielded better results but still didn't exceed the solution quality of the non-cooperative method, and according to the Wilcoxon test, the solutions obtained from the parameterized cooperation and the non-cooperative method do not represent two different populations (p-value = 0.341).

When solving instances characterized as of medium difficulty, the parameterized cooperation method managed to produce almost the same quality of solutions as the non-cooperative method. All models fail to differentiate between one another according to the Wilcoxon test.

In the more difficult instance category classified as Hard, the parameterized cooperation model produced superior solutions regarding quality. This time, the non-cooperative algorithm performed worst than the managed cooperative model. At this point, all algorithms produced solutions that represent different populations according to the Wilcoxon test.

When analyzing the most difficult instances from the Golden set, unfiltered communication seems to have no statistically significant differences when compared to the non-cooperative model, even though it generally leads to better solutions. In particular, after ten repeats the managed cooperation model had an average improvement of 0.26%, but the Wilcoxon test returned a p-value of 0.678.

The PCM had an average improvement of 9.620% − 8.270% = 1.35% at the Golden subset, and 1.910% − 1.070% = 0.84% at the Hard class when compared to the NCM (Table **7.3**). When the time execution was increased to 500 GVNS iterations, PCM still produced better solutions with an average improvement of 0.98%, as depicted in Figure **7.3.2**. According to the Wilcoxon test, there are statistically significant differences between the other models (Table **7.3**).

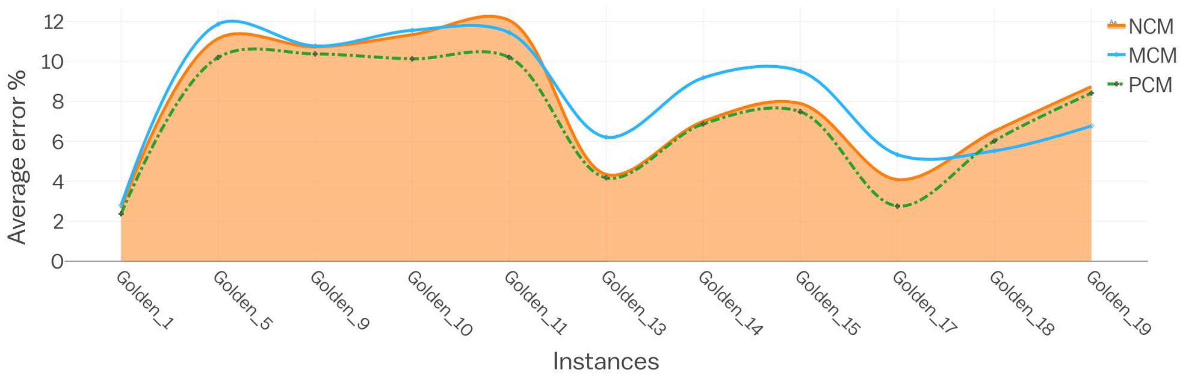

Figure **7.3.2** Average error after 10 repeats of 500 GVNS iterations in 11 Golden instances.

An observation supporting our findings was also made by Groër et al. (Groër et al., 2011). The authors examined the gain provided by cooperation, by studying solution quality within two versions of the same algorithm. Both versions used the same metaheuristic solution methods except that, one variant did not use information sharing among the processors. The authors noticed that the cooperative algorithm did not perform well on small problems, specifically on the instances of Christofides and Eilon (1969); Christofides et al. (1979). Their cooperative algorithm did produce solutions with almost two times better quality on the problem sets of Taillard (Taillard, 1993b), Golden et al. (Golden et al., 1998b) and Li et al. (Li et al., 2005b).

7.3.1 Communication overhead

In order to determine the communication overhead between processors, the total execution time was measured. The time window with the most intensive information sharing is located at the first GVNS iterations. Separately timing the messages passed from the processors to the solution warehouse at irregular intervals, dynamically determined by each process, consists a non-trivial task using Python multithreading library. Therefore, a total time execution measurement should reflect the communication overhead with better accuracy. To get this measurement, ten GVNS iterations for the three models were executed and this experiment was repeated 40 times. The additional, average execution time of MCM compared to NCM (no communication overhead) is depicted in the third column of Table **7.4**.

By comparing the MCM model with the NCM model we find that, on average, communication requires an extra 10% in overall execution time (Figure **7.3.3**). For example, as seen in Table **7.4**, upon completing the first ten GVNS iterations in the Golden set, the MCM model had a completion time of 227.969 seconds (9.623% more than NCM). Thus, it is safe to assume that, the communication overhead for the PCM model lies somewhere in between that of the NCM model and the MCM model, since the communication is far more sparse while delivering better results with less communication.

Table **7.4** MCM model performance and timings

Set	Avg. relative error %	Avg. additional time due to communication overhead%
Set A	1.289	11.460
Set B	3.135	15.138
Set E	5.101	5.077
Set M	15.269	10.713
Golden	11.959	9.623

The best performance of the three models at 800 GVNS iterations after ten repeats are depicted in Tables **7.5**, **7.6**, **7.7** and **7.8**. For the Golden subset, 500 GVNS iterations were executed. Instances with no known optimum are colored. Optimum and best known values are taken from the solutions provided in Uchoa et al. (2017a).

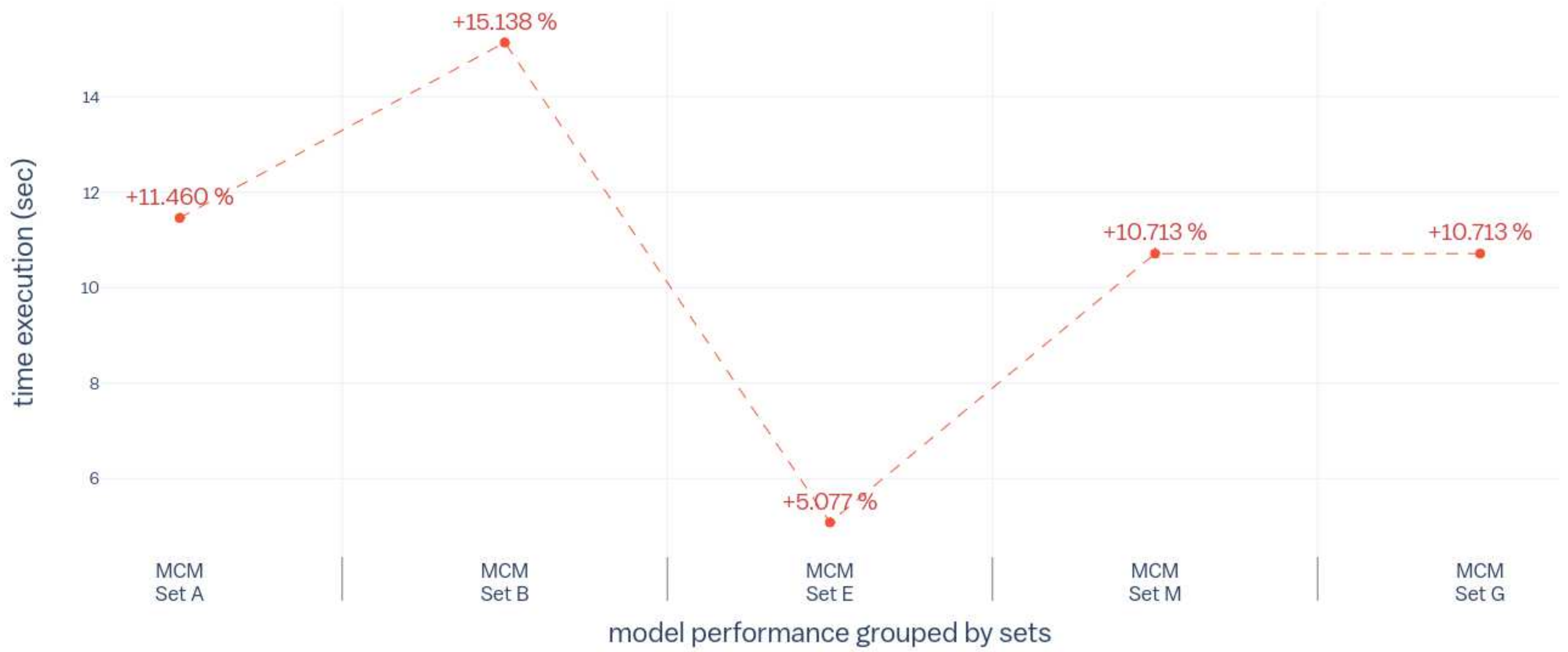

Figure **7.3.3** Communication overhead expressed in total time execution

Table **7.5** Best performance of the three models in A set. Stopping condition: 800 iterations

Instances	GVNS Iters	Best Known	Clarke Wright	GVNS	Rel. error %	Method
A-n32-k5	800	787.0819	1,595.1053	787.0819	0	All models
A-n33-k5	800	662.1101	1,160.8442	662.1101	0	All models
A-n33-k6	800	742.6932	1,214.5044	742.6932	0	All models
A-n34-k5	800	780.9361	1,750.1267	780.9361	0	All models
A-n36-k5	800	802.1318	1,347.7531	802.1318	0	NCM,PCM
A-n37-k5	800	672.4652	1,615.9513	672.4652	0	All models
A-n37-k6	800	950.8522	1,467.1614	950.8522	0	All models
A-n38-k5	800	734.1846	1,688.6356	734.1846	0	All models
A-n39-k5	800	828.9891	1,328.892	829.4375	0	All models
A-n39-k6	800	833.2046	1,665.7231	833.2046	0	All models
A-n44-k6	800	939.3346	1,418.0345	939.3346	0	NCM, PCM
A-n45-k6	800	944.8763	1,807.3193	944.8763	0	NCM, PCM
A-n45-k7	800	1,146.9089	1,683.5652	1,146.9089	0	NCM
A-n46-k7	800	917.9073	1,720.1586	917.9073	0	NCM, PCM
A-n48-k7	800	1,074.3378	1,805.8584	1,074.3378	0	All models
A-n53-k7	800	1,012.3255	2,348.7395	1,012.2490	0	NCM, PCM
A-n54-k7	800	1,171.7843	2,396.2949	1,183.5745	0.3996	NCM
A-n55-k9	800	1,074.4636	2,218.6536	1,074.4636	0	NCM, PCM
A-n60-k9	800	1,355.7992	2,839.9319	1,355.7992	0	MCM,PCM
A-n61-k9	800	1,039.0784	1,989.6459	1,040.3088	0.1182	MCM,PCM
A-n62-k8	800	1,294.2821	2,548.6326	1,313.0479	0.7362	PCM
A-n63-k10	800	1,313.7294	2,656.6606	1,316.8519	0.2371	PCM
A-n63-k9	800	1,622.1446	2,715.2231	1,633.9357	0.7216	PCM
A-n64-k9	800	1,400.8320	2,402.5803	1,403.2783	0.1743	MCM
A-n65-k9	800	1,181.6874	2,743.0330	1,181.6874	0	PCM
A-n69-k9	800	1,165.9945	2,630.5024	1,170.5383	0.3881	MCM,PCM
A-n80-k10	800	1,766.4999	3,167.9758	1,790.4506	1.3376	PCM

Table **7.6** Best performance of the three models in B set. Stopping condition: 800 iterations

Instances	GVNS Iters	Best Known	Clarke Wright	GVNS	Rel. error %	Method
B-n31-k5	800	676.0884	1,870.3339	676.0884	0	All models
B-n34-k5	800	790.1838	1,272.2225	790.1838	0	All models
B-n35-k5	800	956.2941	1,918.5958	956.2941	0	All models
B-n38-k6	800	807.8788	1,780.4048	807.8788	0	All models
B-n39-k5	800	553.1564	1,165.3777	553.1564	0	All models
B-n41-k6	800	833.8063	1,436.6050	833.8063	0	All models
B-n43-k6	800	746.9838	1,137.7297	746.9838	0	All models
B-n44-k7	800	914.9648	1,484.5364	914.9648	0	All models
B-n45-k5	800	754.4388	1,690.8717	754.4388	0	All models
B-n45-k6	800	680.4379	1,449.5807	680.4379	0	NCM ,PCM
B-n50-k7	800	744.2280	1,687.9685	744.2280	0	All models
B-n50-k8	800	1,321.5236	1,743.5549	1,319.5033	0	PCM
B-n51-k7	800	1,019.3251	2,995.2715	1,022.6471	0	All models
B-n52-k7	800	749.9697	1,859.6626	749.9697	0	All models
B-n56-k7	800	712.9161	1,714.6548	712.9161	0	NCM,PCM
B-n57-k7	800	1,143.9458	2,814.8279	1,143.3284	0	All models
B-n57-k9	800	1,603.3706	2,676.2986	1,603.3706	0	All models
B-n63-k10	800	1,501.2707	2,125.5090	1,506.8827	0.3724	MCM
B-n64-k9	800	869.3157	2,688.2004	869.3157	0	All models
B-n66-k9	800	1,325.355	3,410.4368	1,327.4408	0.1571	PCM
B-n67-k10	800	1,039.3589	1,727.8862	1,039.3589	0	PCM
B-n68-k9	800	1,278.2107	2,303.6853	1,283.8157	0.4365	MCM
B-n78-k10	800	1,229.2734	3,301.4128	1,232.6011	0.2699	PCM

Table **7.7** Best performance of the three models in E & M set. Stopping condition: 800 iterations

Instances	GVNS Iters	Best Known	Clarke Wright	GVNS	Rel. error %	Method
E-n13-k4	800	247	371	247.0000	0	All models
E-n22-k4	800	375.2798	660.4315	375.2798	0	All models
E-n23-k3	800	568.5625	1,305.4011	568.5625	0	All models
E-n30-k3	800	538.7947	1,353.3146	538.7947	0	All models
E-n31-k7	800	379	1,036	379.0000	0	All models
E-n33-k4	800	837.6716	1,234.9409	837.6716	0	All models
E-n51-k5	800	524.9442	1,440.7513	524.9442	0	NCM
E-n76-k7	800	666.8325	2,069.3508	697.7697	2.26	PCM
E-n76-k8	800	740.6554	2,067.9832	740.6554	0	PCM
E-n76-k14	800	1,026.7063	2,355.0591	1,026.706	0	PCM
E-n76-k10	800	837.3556	2,431.7793	838.9785	0.1934	PCM
E-n101-k14	800	1,082.6501	3,000.6836	1,094.6382	1.0951	PCM
E-n101-k8	800	826.9080	2,453.4512	832.7041	0.6960	PCM
M-n101-k10	800	820	2,482.2693	819.5575	0	PCM
M-n200-k16	800	1,364.2988	5,038.9443	1,331.2376	2.7471	PCM

Table **7.8** Best performance of the three models in Golden set. Stopping condition: 500 iterations

Instances	GVNS Iters	Best Known	Clarke Wright	GVNS	Rel. error %	Method
Golden 1	500	5,627.5400	23,137.2109	5,742.6367	2.0751	PCM
Golden 5	500	6,460.9800	29,397.4941	7,194.6733	10.1977	PCM
Golden 9	500	585.4300	1,614.6719	646.8176	10.3750	PCM
Golden 10	500	741.5600	2,034.4878	827.7824	11.1288	PCM
Golden 11	500	918.4500	2,970.5544	1,029.8816	11.4432	PCM
Golden 13*	500	859.1100	3,563.9204	894.5257	4.1738	PCM
Golden 14*	500	1,081.3100	5,387.9766	1,160.2756	6.8712	PCM
Golden 15	500	1,345.2300	6,892.6626	1,446.0418	7.4805	PCM
Golden 17*	500	707.7900	2,578.7881	727.8627	2.7618	PCM
Golden 18*	500	997.5200	3,751.8257	1,053.2850	5.5213	MCM
Golden 19*	500	1,366.8600	5,227.4507	1,464.7230	6.7674	MCM

7.3.2 Taking the instance attributes into account

In order to further study the observed patterns, we tested the three proposed parallel models using the X-Set instances generated by Uchoa et al. (2017b). Utilizing all the information gathered in section 7.1, the computational effort associated with the instance characteristics can be summarized in Table **7.9**. In order to rank the instances by difficulty, we assigned the following values to the instance characteristics:

- $Customer\ positioning = 1$
  - **i.** $Clustered = 1$
  - **ii.** $Random\ Clustered\ (RC) = 5$
  - **iii.** $Random = 7$
- $Depot\ positioning = 5$
  - **i.** $Center = 1$
  - **ii.** $Random\ Clustered\ (RC) = 5$
  - **iii.** $Corner = 10$
- $Customer\ demands = 10$
  - **i.** $Unitary,\ Small\ values = 1$
  - **ii.** $High\ variability,\ Large\ values = 5$
  - **iii.** $Quadrant = 10$

Table **7.9** The computational effort associated with instance characteristics

Computational effort	**Depot positioning**	**Customer positioning**	**Customer demands**
	(Significant effect)	(Light effect)	(Great effect)
Small	Center	Clustered	Unitary; Small values

Intermediate	Random	High variability	High variability;
			Large values;
			Few large and many small

Large	Corner	-	Quadrant

For example, using the above values and taking into account the customer count, the instance X-n101-k25 has a total computational difficulty (Tdf) of 183:

$$\left.\begin{array}{l}[\text{Customer demands}] * [\text{Intermediate}] = 10 * 5 = 50 \\ [\text{Root positioning}] * [\text{Intermediate}] = 5 * 5 = 25 \\ [\text{Customer positioning}] * [\text{Intermediate}] = 1 * 7 = 7\end{array}\right\} \text{Tdf} = \text{df} + \text{Customers} = 82 + 101 = 183$$

In Table **7.10** we display the Tdf for the first 45 instances of the X-Set.

Table **7.10** Computational difficulty of some X-set instances based on instance attributes

Instance name	Customers (n)	Demand	Root position	Customer position	Difficulty	Total Diff.
X-n101-k25	101	[0, 100]	Random	Random	82	183
X-n106-k14	106	[0, 100]	Corner	Clustered	101	207
X-n110-k13	110	[0, 10]	Center	Random	22	132
X-n115-k10	115	[0, 99]	Center	Random	62	177
X-n120-k6	120	[0, 1]	Corner	RC	65	185
X-n125-k30	125	[0, 100]	Random	Clustered	76	201
X-n129-k18	129	[0, 10]	Corner	RC	65	194
X-n134-k13	134	[0, 100]	Random	Clustered	76	210
X-n139-k10	139	[0, 10]	Center	Random	22	161
X-n143-k7	143	[0, 99]	Corner	Random	107	250
X-n148-k46	148	[0, 10]	Random	RC	50	198
X-n153-k22	153	[0, 98]	Center	Clustered	56	209
X-n157-k13	157	[0, 1]	Random	Clustered	36	193
X-n162-k11	162	[0, 100]	Center	RC	60	222
X-n167-k10	167	[0, 10]	Corner	Random	67	234
X-n172-k51	172	[0, 100]	Center	RC	60	232
X-n176-k26	176	[0, 100]	Corner	Random	107	283
X-n181-k23	181	[0, 1]	Random	Clustered	36	217
X-n186-k15	186	[0, 100]	Random	Random	82	268
X-n190-k8	190	[0, 10]	Corner	Clustered	61	251
X-n195-k51	195	[0, 100]	Center	RC	60	255
X-n200-k36	200	[0, 100]	Random	Clustered	76	276
X-n204-k19	204	[0, 100]	Center	RC	60	264
X-n209-k16	209	[0, 10]	Corner	Random	67	276
X-n214-k11	214	[0, 100]	Center	Clustered	56	270
X-n219-k73	219	[0, 1]	Corner	Random	67	286
X-n223-k34	223	[0, 10]	Random	RC	40	263
X-n228-k23	228	[0, 100]	Random	Clustered	76	304
X-n233-k16	233	[0, 100]	Center	RC	60	293
X-n237-k14	237	[0, 1]	Corner	Random	67	304
X-n242-k48	242	[0, 10]	Corner	Random	67	309
X-n247-k50	247	[0, 100]	Center	Clustered	56	303
X-n251-k28	251	[0, 10]	Random	RC	40	291
X-n256-k16	256	[0, 100]	Center	Clustered	56	312
X-n261-k13	261	[0, 100]	Corner	Random	107	368
X-n266-k58	266	[0, 10]	Random	RC	40	306
X-n270-k35	270	[0, 100]	Center	RC	60	330
X-n275-k28	275	[0, 1]	Random	Clustered	36	311
X-n280-k17	280	[0, 100]	Corner	Random	107	387
X-n284-k15	284	[0, 10]	Random	Clustered	36	320
X-n289-k60	289	[0, 100]	Corner	RC	105	394
X-n294-k50	294	[0, 99]	Center	Random	62	356
X-n298-k31	298	[0, 10]	Random	Random	42	340
X-n303-k21	303	[0, 100]	Center	Clustered	56	359
X-n308-k13	308	[0, 95]	Corner	RC	105	413
X-n336-k84	335	[0, 100]	Corner	Random	107	443

Several runs of each model were executed in order to observe the effects of communication and discover if any of the models is superior compared to the other. The test set is composed of a subset of the X-set containing the following instances:

X-n110-k13 (Tdf: 132), X-n143-k7 (Tdf: 250), X-n153-k22 (Tdf: 209), X-n256-k16 (Tdf: 312), X-n261-k13 (Tdf: 368), X-n280-k17 (Tdf: 387).

The computational difficulty of the instances mentioned above spans from 132 to 387. The test is repeated until the single termination criterion, a certain number of GVNS iterations is met. The three parallel models were tested with the following GVNS iterations: 10, 20, 40, 100, and 300. All tests were repeated ten times.

Before proceeding into the analysis of the data obtained from the tests, a Friedman test was applied to the results. The received p-value was 0.000182, proving that there is sufficient proof to consider the three algorithms separate. A typical significance level $\alpha = 0.05$ was acknowledged as the threshold for rejecting the null hypothesis.

This time, instead of using just the instance size, the classification into the three groups "Easy, Medium, Hard" is based on the difficulty the algorithm encounters in completing one GVNS iteration as seen in Table **7.11**. Based on this factor, the three groups formed are: Easy(X-n110-k13), Medium (X-n143-k7, X-n153-k22) and Hard (X-n261-k13, X-n256-k16, X-n280-k17). The instances in Table **7.11** are sorted in descending order of difficulty, and the order approximately coincided with the number of customers.

Table **7.11** Instances sorted by computational difficulty defined by GVNS iterations

Instance	**df**	**Tdf**	**mean error %**	**GVNS iters per sec ↓**	**Iterations**
MCM performance on X subset ($\theta \leftarrow 0$)					
X-n110-k13	22	132	1.2755	0.40	300
X-n143-k7	107	209	12.3691	0.22	300
X-n153-k22	56	312	3.6825	0.14	300
X-n261-k13	56	368	12.9790	0.07	300
X-n256-k16	107	250	4.0196	0.04	300
X-n280-k17	107	387	16.1482	0.03	300
NCM performance on X subset ($\theta \leftarrow 1$)					
X-n110-k13	22	132	1.5824	0.73	300
X-n143-k7	107	209	11.0923	0.22	300
X-n153-k22	56	312	4.1002	0.15	300
X-n261-k13	56	368	12.4041	0.07	300
X-n256-k16	107	250	3.6238	0.05	300
X-n280-k17	107	387	15.9495	0.04	300
PCM performance on X subset ($\theta \leftarrow dynamic\_range(0\ to\ 1)$)					
X-n110-k13	22	132	1.4474	0.48	300
X-n143-k7	107	209	7.6543	0.16	300
X-n153-k22	56	312	3.7153	0.12	300
X-n261-k13	56	368	11.8334	0.06	300
X-n256-k16	107	250	3.4001	0.04	300
X-n280-k17	107	387	16.0622	0.03	300

Again, the same pattern emerges. Communication seems to work better for smaller instances and small-time horizon. This similar trend, observed in the previous results, occurs once more in the X-Set instances. Given a horizon of 300 GVNS iterations, the average error of the three models is shown in Figure **7.3.4**. In easier instances, the Managed Cooperation Model performs better (1.275%) but the performance drops as the difficulty increases (8.025% at the Medium category and 11.048% at the Hard).

A timeline of the overall performance of the three models, with different time horizons (10, 20, 40, 100 and 300 GVNS iterations) is shown in Figure **7.3.5**. The communication scheme in Managed Cooperative Model works well, compared to the Non-Cooperative Model for the first GVNS iterations. More specifically, it is the best choice at 10 GVNS iterations. As solutions become rarer to find, diversification is the best strategy to follow. At 300 GVNS iterations, the overall performance of the MCM is the worst strategy to follow.

One more interesting conclusion about the communication overhead can be drawn from the GVNS iterations achieved per second. MCM communication overhead has an impact of 16.1453% on performance while PCM has a 5.9740% additional cost over MCM.

Finally, the impact of communication appears to be a function of two factors: the time of communication and the size of solution space (the size of the neighborhood to explore). It appears that filtered or no communication near the end of the search yields better results and communication favors vast neighborhoods.

7.3.3 Solution exchange

In conclusion, information sharing seems to outperform the independent search method and is a valuable strategy for tackling hard instances. The parameterized model is achieving better convergence rate and solution quality by reducing the level of interaction between the threads. Sparse information sharing in the parameterized cooperation model (Figure **7.3.6**) leads to a better performance than dense solution exchange (Figure **7.3.7**), thus reducing the communication overhead.

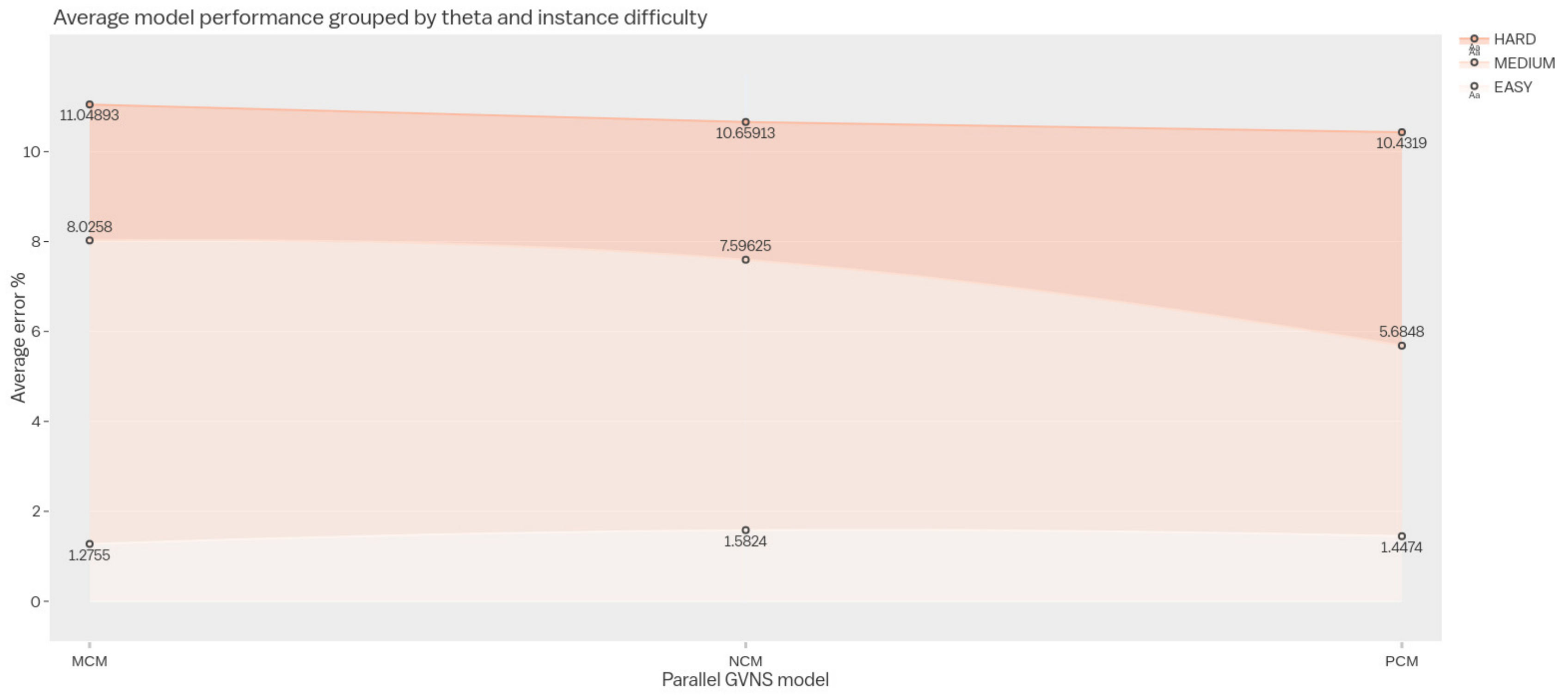

Figure **7.3.4** The 3 stacked area plots show the performance of each model in Uchoa instances (Xset) grouped by theta and instance difficulty

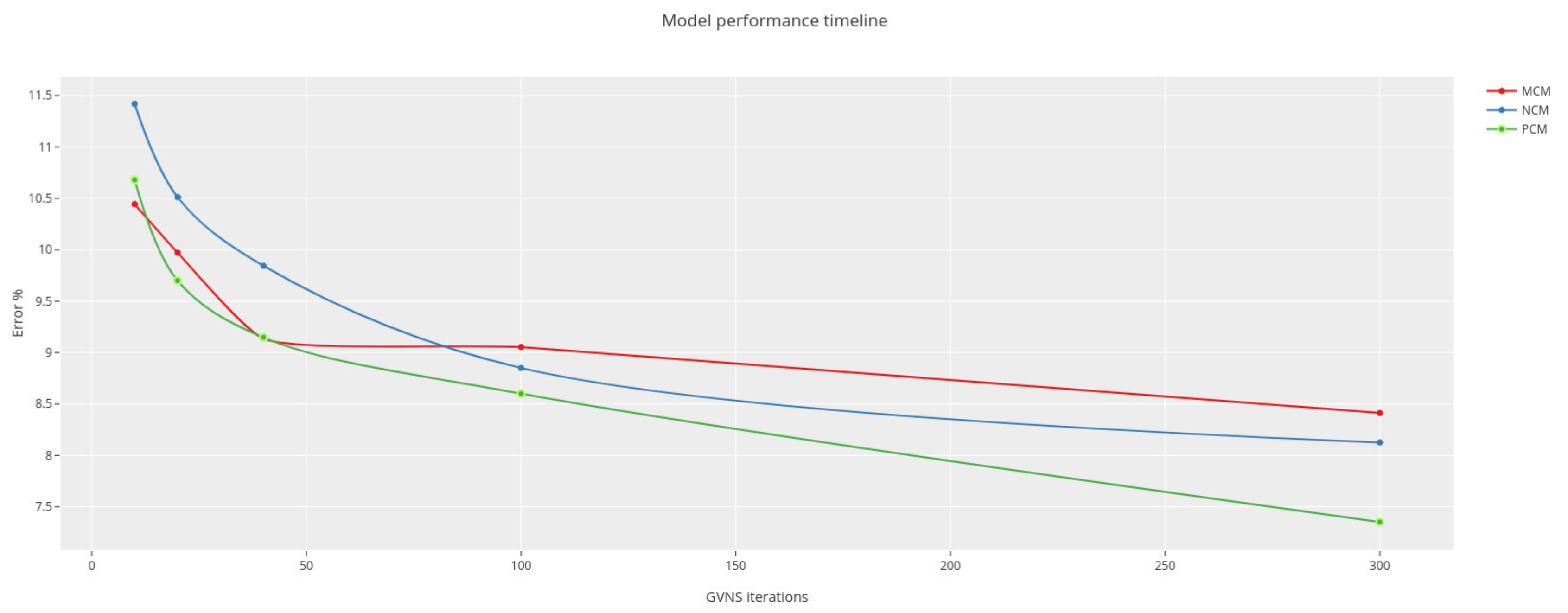

Figure **7.3.5** Model performance time-line (GVNS iterations)

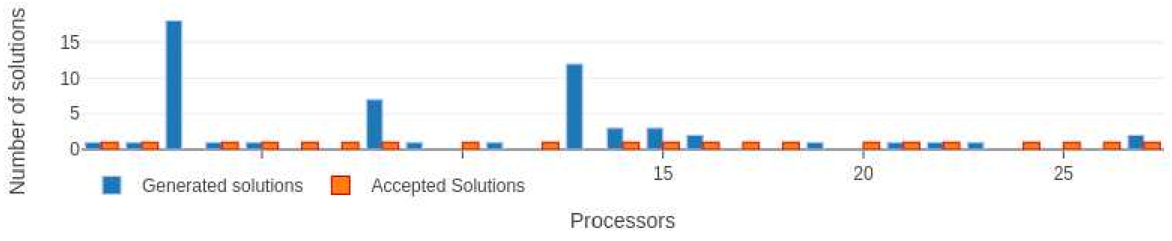

Figure **7.3.6**: Solution exchange in parameterized model is sparse. This leads to better exploration through partial thread isolation.

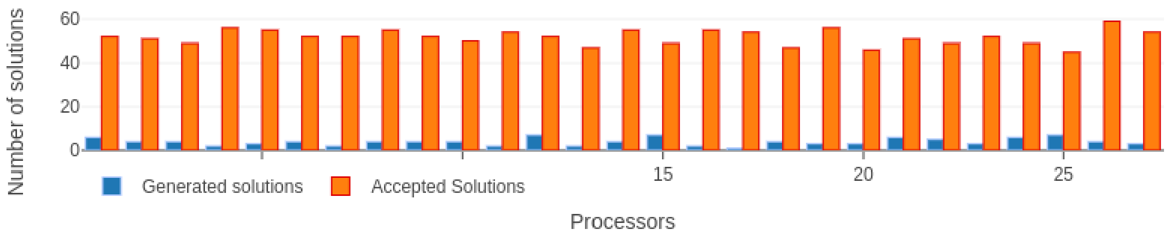

Figure **7.3.7**: Solution exchange in managed cooperation is dense. Generation of new global solutions stabilizes in a lower plateau.

CHAPTER 8

Conclusions

8.1 Conclusions and future work

In this paper, a literature review with recent successful parallel implementations of VNS regarding different variants of VRPs was presented. Also, three models for the parallelization of VNS for the efficient solution of CVRP were proposed. Several well known instances for the CVRP were used in order to compare and analyze the cooperation strategies between the three parallel metaheuristic models.

A strong indication that the cooperation strategy can have a decisive influence on the quality of the solutions, seems to exist. Specifically, partial cooperation through filtered communication seems to provide better quality solutions on harder problems. To the best of the author's knowledge, the proposed cooperative search mechanism presented in this paper is a novel approach. Finally, the impact of communication appears to be a function of two factors: the time of communication and the size of solution space (the size of the neighborhood to explore). It appears that filtered or no communication near the end of the search yields better results and communication favors vast neighborhoods.

The findings presented in Section 7.3, are in line with previous findings by other researchers working with different solution methods and on different optimization problems. Thus, an observation supporting our findings was also made by Groër et al. (2011). The authors examined the gain provided by cooperation by studying solution quality within two versions of the same algorithm. Both versions used the same metaheuristic solution methods except that one variant did not use information sharing among the processors. The authors noticed that the cooperative algorithm did not perform well on small problems, specifically on the instances of Christofides and Eilon (1969); Christofides et al. (1979). Their cooperative algorithm produced solutions with almost two times better quality on the problem sets of Taillard (1993b), Golden et al. (1998b) and Li et al. (2005b).

In another study by Davidović and Crainic (2015) , the researchers examine several parallel local search strategies. The authors conclude that, a parallel approach which dynamically partitions the neighborhoods among the processors yields the best results. This method uses a supervisor that, broadcasts a new solution among the workers. A negative aspect of this approach is the significant increase in communication. Even though the best results were obtained using a modest number of processors due to less communication overhead, the authors noticed that, the use of a large number of processors is beneficial when the instance dimensions grow. Thus, the communication cost proves to be not an important factor in the solution of large-scale hard instances.

Munera et al. in their recent paper in 2016 (Munera et al., 2016) presented a cooperative parallel extremal optimization for the solution of the quadratic assignment problem. The authors mention that, they have performed the tuning process in a manual way without parameter self-adaptation. Therefore, the self-adaptive mechanism presented in this work constitutes a step towards this direction. To the best of the authors' knowledge, the proposed cooperative search mechanism presented in this paper is a novel approach.

Such cooperative parallelization strategies may help compute solutions for even more complicated integrated problems, e.g., location-inventory-routing problems (Karakostas et al., 2019). Future studies may include the use of machine learning and artificial intelligence in order to achieve even better algorithm configuration and parameter tuning. Also, the use of smarter memory-based strategies could provide better solutions.

Bibliography

Achuthan, N., L. Caccetta, and S. Hill (1997). On the vehicle routing problem. *Nonlinear Analysis: Theory, Methods & Applications 30*(7), 4277–4288.

Alexander, A., H. Walker, and M. Naim (2014, September). Decision theory in sustainable supply chain management: a literature review. *Supply Chain Management: An International Journal 19*(5/6), 504–522.

Altınel, İ. K. and T. Öncan (2005, aug). A new enhancement of the clarke and wright savings heuristic for the capacitated vehicle routing problem. *Journal of the Operational Research Society 56*(8), 954–961.

Antoniadis, N. and A. Sifaleras (2017, apr). A hybrid CPU-GPU parallelization scheme of variable neighborhood search for inventory optimization problems. *Electronic Notes in Discrete Mathematics 58*, 47–54.

Arnold, F., M. Gendreau, and K. Sörensen (2019, July). Efficiently solving very large-scale routing problems. *Computers & Operations Research 107*, 32–42.

Arnold, F., M. Gendreau, and K. Sörensen (2019, jul). Efficiently solving very large-scale routing problems. *Computers & Operations Research 107*, 32–42.

Augerat, P., J. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 495, Institute for Systems Analysis and Computer Science (IASI), Rome.

Augerat, P., J. M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi (1995, 01). Computational results with a branch and cut code for the capacitated vehicle routing problem. *Istituto di Analisi dei Sistemi ed Informatica, CNR 1*(1), 3–22.

Aydin, M. E. and M. Sevkli (2008). Sequential and parallel variable neighborhood search algorithms for job shop scheduling. In *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, pp. 125–144. Springer Berlin Heidelberg.

Baldacci, R., P. Toth, and D. Vigo (2010). Exact algorithms for routing problems under vehicle capacity constraints. *Annals of Operations Research 175*(1), 213–245.

Balinski, M. L. and R. E. Quandt (1964, April). On an integer program for a delivery problem. *Operations Research 12*(2), 300–304.

Bard, J. F., L. Huang, M. Dror, and P. Jaillet (1998, Sep). A branch and cut algorithm for the vrp with satellite facilities. *IIE Transactions 30*(9), 821–834.

Beasley, J. (1983, January). Route first—cluster second methods for vehicle routing. *Omega 11*(4), 403–408.

Benoist, T., B. Estellon, F. Gardi, R. Megel, and K. Nouioua (2011). Localsolver 1.x: a black-box local-search solver for 0-1 programming. *4OR 9*(3), 299–316.

Birrattari, M., L. Paquete, T. Stützle, and K. Varrentrapp (2001). Classification of metaheuristics and design of experiments for the analysis of components. In *Classification of metaheuristics and design of experiments for the analysis of components*.

Blum, C. and A. Roli (2003, sep). Metaheuristics in combinatorial optimization. *ACM Computing Surveys 35*(3), 268–308.

Bodin, L. D. and L. Berman (1979, May). Routing and scheduling of school buses by computer. *Transportation Science 13*(2), 113–129.

Bouthillier, A. L., T. Crainic, and P. Kropf (2005, jul). A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intelligent Systems 20*(4), 36–42.

Bouthillier, A. L. and T. G. Crainic (2005, jul). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research 32*(7), 1685–1708.

Bruck, B. P., A. G. dos Santos, and J. E. C. Arroyo (2012, jun). Hybrid metaheuristic for the single vehicle routing problem with deliveries and selective pickups. In *2012 IEEE Congress on Evolutionary Computation*. IEEE.

Burrows, C. P., S. Eilon, C. D. T. Watson-Gandy, and N. Christofides (1972). Distribution management: Mathematical modelling and practical analysis. *Applied Statistics 21*(3), 337.

Cattaruzza, D., N. Absi, and D. Feillet (2016, January). Vehicle routing problems with multiple trips. *4OR 14*(3), 223–259.

Christofides, N. and S. Eilon (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society 20*(3), 309–318.

Christofides, N., A. Mingozzi, and P. Toth (1979). The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi (Eds.), *Combinatorial Optimization*, Volume 1, Chapter 11, pp. 315–338. Chichester, UK: Wiley Interscience.

Clarke, G. and J. W. Wright (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*(4), 568–581.

Coelho, I. M., L. S. Ochi, P. L. A. Munhoz, M. J. F. Souza, R. Farias, and C. Bentes (2012). The single vehicle routing problem with deliveries and selective pickups in a cpu-gpu heterogeneous environment. In *14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems (HPCC-ICESS)*, pp. 1606–1611. IEEE.

Cordeau, J.-F., G. Laporte, M. W. Savelsbergh, and D. Vigo (2007). Chapter 6 vehicle routing. In *Transportation*, pp. 367–428. Elsevier.

Cordeau, J.-F. and M. Maischberger (2012, sep). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research 39*(9), 2033–2050.

Crainic, T. G. (2008). Parallel solution methods for vehicle routing problems. In *The vehicle routing problem: Latest advances and new challenges*, pp. 171–198. Springer.

Crainic, T. G., M. Gendreau, P. Hansen, and N. Mladenović (2004, may). Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics 10*(3), 293–314.

Crainic, T. G. and N. Hail (2005). Parallel metaheuristics applications. *Parallel metaheuristics: A new class of algorithms 47*, 447–494.

Crainic, T. G., M. Toulouse, and M. Gendreau (1997). Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing 9*(1), 61–72.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM 7*(3), 171–176.

Dantzig, G. B. and J. H. Ramser (1959). The truck dispatching problem. *Management Science 6*(1), 80–91.

Davidović, T. and T. G. Crainic (2015). Parallel local search to schedule communicating tasks on identical processors. *Parallel Computing 48*, 1–14.

de Córdoba, P. F., L. M. García-Raffi, A. Mayado, and J. M. Sanchis (2000, June). A real delivery problem dealt with monte carlo techniques. *Top 8*(1), 57–71.

Fisher, M. L. (1994, aug). Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research 42*(4), 626–642.

Fisher, M. L. and R. Jaikumar (1981). A generalized assignment heuristic for vehicle routing. *Networks 11*(2), 109–124.

Fukasawa, R., H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R. F. Werneck (2005, October). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming 106*(3), 491–511.

García-López, F., B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega (2002). The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics 8*(3), 375–388.

Garey, M. R. (1979). Computers and intractability : A guide to the theory of np-completeness. *Revista Da Escola De Enfermagem Da U S P 44*(2), 340.

Gaskell, T. J. (1967, sep). Bases for vehicle fleet scheduling. *Journal of the Operational Research Society 18*(3), 281–295.

Gayialis, S. P., G. D. Konstantakopoulos, and I. P. Tatsiopoulos (2018, September). Vehicle routing problem for urban freight transportation: A review of the recent literature. In *Operational Research in the Digital Era – ICT Challenges*, pp. 89–104. Springer International Publishing.

Gayialis, S. P., G. D. Konstantakopoulos, and I. P. Tatsiopoulos (2019). Vehicle routing problem for urban freight transportation: A review of the recent literature. In A. Sifaleras and K. Petridis (Eds.), *Operational Research in the Digital Era – ICT Challenges*, Cham, pp. 89–104. Springer International Publishing.

Gendreau, M., G. Ghiani, and E. Guerriero (2015, December). Time-dependent routing problems: A review. *Computers & Operations Research 64*, 189–197.

Gendreau, M. and J.-Y. Potvin (2010). *Handbook of metaheuristics*, Volume 2. Springer.

Gendreau, M., J.-Y. Potvin, O. Bräumlaysy, G. Hasle, and A. Løkketangen (2007). Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In *Operations Research/Computer Science Interfaces*, pp. 143–169. Springer US.

Gillett, B. E. and L. R. Miller (1974, April). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research 22*(2), 340–349.

Glover, F. and G. A. Kochenberger (2003). *Handbook of Metaheuristics.* Springer US.

Golden, B. L., T. L. Magnanti, and H. Q. Nguyen (1977). Implementing vehicle routing algorithms. *Networks 7*(2), 113–148.

Golden, B. L., E. A. Wasil, J. P. Kelly, and I.-M. Chao (1998a). The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results. In *Fleet Management and Logistics*, pp. 33–56. Springer US.

Golden, B. L., E. A. Wasil, J. P. Kelly, and I.-M. Chao (1998b). The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In *Fleet management and logistics*, pp. 33–56. Springer.

Gribkovskaia, I., G. Laporte, and A. Shyshou (2008, September). The single vehicle routing problem with deliveries and selective pickups. *Computers & Operations Research 35*(9), 2908–2924.

Groër, C., B. Golden, and E. Wasil (2011). A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing 23*(2), 315–330.

Gurpreet, E. and D. Vijay (2014, 01). Open vehicle routing problem by ant colony optimization. *International Journal of Advanced Computer Science and Applications 5*(3), 63–67.

Hansen, P., N. Mladenović, J. Brimberg, and J. A. M. Pérez (2010). Variable neighborhood search. In *Handbook of metaheuristics*, pp. 61–86. Springer US.

Hansen, P., N. Mladenović, and J. A. M. Pérez (2008). Variable neighbourhood search: methods and applications. *4OR 6*(4), 319–360.

Hansen, P., N. Mladenović, and J. A. M. Pérez (2009, oct). Variable neighbourhood search: methods and applications. *Annals of Operations Research 175*(1), 367–407.

Hashimoto, H., M. Yagiura, S. Imahori, and T. Ibaraki (2010, September). Recent progress of local search in handling the time window constraints of the vehicle routing problem. *4OR 8*(3), 221–238.

Hill, A. V. and W. C. Benton (1992, April). Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *The Journal of the Operational Research Society 43*(4), 343.

Ibrahim, A., R. O Abdulaziz, J. Ishaya, and S. Sowole (2019, 06). Vehicle routing problem with exact methods. *The International Organization of Scientific Research (IOSR) 15*, 5–15.

Jaradat, G., M. Ayob, and I. Almarashdeh (2016). The effect of elite pool in hybrid population-based meta-heuristics for solving combinatorial optimization problems. *Applied Soft Computing 44*, 45–56.

Jin, J., T. G. Crainic, and A. Løkketangen (2014, apr). A cooperative parallel metaheuristic for the capacitated vehicle routing problem. *Computers & Operations Research 44*, 33–41.

Juan, A. A., J. Faulin, E. Pérez-Bernabeu, and O. Domínguez (2013). Simulation-optimization methods in vehicle routing problems: A literature review and an example. In *Modeling and Simulation in Engineering, Economics, and Management*, pp. 115–124. Springer Berlin Heidelberg.

Karakostas, P., A. Sifaleras, and M. C. Georgiadis (2019). Basic VNS algorithms for solving the pollution location inventory routing problem. In *Variable Neighborhood Search. ICVNS 2018. Lecture Notes in Computer Science*, Volume 11328, pp. 64–76. Springer, Cham.

Koskosidis, Y. A., W. B. Powell, and M. M. Solomon (1992, May). An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Science 26*(2), 69–85.

Lahrichi, N., T. G. Crainic, M. Gendreau, W. Rei, G. C. Crişan, and T. Vidal (2015, oct). An integrative cooperative search framework for multi-decision-attribute combinatorial optimization: Application to the MDPVRP. *European Journal of Operational Research 246*(2), 400–412.

Lampkin, B. and A. Wren (1972, September). Computers in transport planning and operation. *Operational Research Quarterly (1970-1977) 23*(3), 404.

Laporte, G. (1992, jun). The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research 59*(3), 345–358.

Laporte, G. and Y. Nobert (1987). Exact algorithms for the vehicle routing problem. In *North-Holland Mathematics Studies*, Volume 132, pp. 147–184. Elsevier.

Laporte, G. and F. Semet (2002, jan). 5. classical heuristics for the capacitated VRP. In *The Vehicle Routing Problem*, pp. 109–128. Society for Industrial and Applied Mathematics.

Le Bouthillier, A. and T. G. Crainic (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research 32*(7), 1685–1708.

Lenstra, J. K. and A. H. G. R. Kan (1981). Complexity of vehicle routing and scheduling problems. *Networks 11*(2), 221–227.

Li, F., B. Golden, and E. Wasil (2005a, may). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research 32*(5), 1165–1179.

Li, F., B. Golden, and E. Wasil (2005b). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research 32*(5), 1165–1179.

Malandraki, C. and M. S. Daskin (1992, August). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science 26*(3), 185–200.

Mladenović, N. (2011). *1st International Symposium & 10th Balkan Conference on Operational Research*, pp. 1–23. UOM.

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

Moreno-Pérez, J., P. Hansen, H. Montreal, and N. Mladenović (2004, 10). Parallel variable neighborhood searches. pp. 1–21.

Munera, D., D. Diaz, and S. Abreu (2016). Solving the quadratic assignment problem with cooperative parallel extremal optimization. In *Evolutionary Computation in Combinatorial Optimization*, pp. 251–266. Springer.

Naddef, D. and G. Rinaldi (2002, January). 3. branch-and-cut algorithms for the capacitated VRP. In *The Vehicle Routing Problem*, pp. 53–84. Society for Industrial and Applied Mathematics.

Nagy, G., N. A. Wassan, M. G. Speranza, and C. Archetti (2015, may). The vehicle routing problem with divisible deliveries and pickups. *Transportation Science 49*(2), 271–294.

Nesmachnow, S. (2014). An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics 3*(4), 320.

Newton, R. M. and W. H. Thomas (1974, August). Bus routing in a multi-school system. *Computers & Operations Research 1*(2), 213–222.

O., E. M. T., A. H. E. Z., and M. G. E. (2015, dec). Literature review on the vehicle routing problem in the green transportation context. *Luna Azul 1*(42), 362–387.

Osman, I. H. (1993, December). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research 41*(4), 421–451.

Pecin, D., A. Pessoa, M. Poggi, and E. Uchoa (2014). Improved branch-cut-and-price for capacitated vehicle routing. In *Integer Programming and Combinatorial Optimization*, pp. 393–403. Springer International Publishing.

Pecin, D., A. Pessoa, M. Poggi, and E. Uchoa (2016, June). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation 9*(1), 61–100.

Pérez, J. A. M., P. Hansen, and N. Mladenović (2005, September). Parallel variable neighborhood search. In *Parallel Metaheuristics*, pp. 247–266. John Wiley & Sons, Inc.

Pessoa, A., R. Sadykov, E. Uchoa, and F. Vanderbeck (2019). A generic exact solver for vehicle routing and related problems. In *Integer Programming and Combinatorial Optimization*, pp. 354–369. Springer International Publishing.

Pillac, V., M. Gendreau, C. Guéret, and A. L. Medaglia (2013, February). A review of dynamic vehicle routing problems. *European Journal of Operational Research 225*(1), 1–11.

Pisinger, D. and S. Ropke (2007, August). A general heuristic for vehicle routing problems. *Computers & Operations Research 34*(8), 2403–2435.

Poggi, M. and E. Uchoa (2014, November). Chapter 3: New exact algorithms for the capacitated vehicle routing problem. In *Vehicle Routing*, pp. 59–86. Society for Industrial and Applied Mathematics.

Polacek, M., S. Benkner, K. F. Doerner, and R. F. Hartl (2008, dec). A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows. *Business Research 1*(2), 207–218.

Polat, O. (2017). A parallel variable neighborhood search for the vehicle routing problem with divisible deliveries and pickups. *Computers & Operations Research 85*, 71–86.

Rochat, Y. and É. D. Taillard (1995, sep). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics 1*(1), 147–167.

Rothlauf, F. (2011). *Design of Modern Heuristics.* Springer Berlin Heidelberg.

Ryan, D. M., C. Hjorring, and F. Glover (1993, March). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society 44*(3), 289–296.

Salhi, S., A. Imran, and N. A. Wassan (2014, December). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers & Operations Research 52*, 315–325.

Salhi, S. and G. Nagy (1999, oct). A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society 50*(10), 1034–1042.

Salhi, S., N. Wassan, and M. Hajarat (2013, September). The fleet size and mix vehicle routing problem with backhauls: Formulation and set partitioning-based heuristics. *Transportation Research Part E: Logistics and Transportation Review 56*, 22–35.

Schulz, C., G. Hasle, A. R. Brodtkorb, and T. R. Hagen (2013). GPU computing in discrete optimization. part II: Survey focused on routing problems. *EURO Journal on Transportation and Logistics 2*(1-2), 159–186.

Shi, J. and Q. Zhang (2018, apr). A new cooperative framework for parallel trajectory-based metaheuristics. *Applied Soft Computing 65*, 374–386.

Skouri, K., A. Sifaleras, and I. Konstantaras (2018). Open problems in green supply chain modeling and optimization with carbon emission targets. In P. M. Pardalos and A. Migdalas (Eds.), *Open Problems in Optimization and Data Analysis*, pp. 83–90. Springer Optimization and Its Applications.

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

Sörensen, K. and F. W. Glover (2013). Metaheuristics. In *Encyclopedia of Operations Research and Management Science*, pp. 960–970. Springer US.

Sörensen, K., M. Sevaux, and F. Glover (2016). A history of metaheuristics. In *OR2016: Annual International Conference of the German Operations Research Society*, Hamburg, Germany.

Taillard, É. (1993a, December). Parallel iterative search methods for vehicle routing problems. *Networks 23*(8), 661–673.

Taillard, É. (1993b). Parallel iterative search methods for vehicle routing problems. *Networks 23*(8), 661–673.

Talbi, E.-G. (2009, jun). *Metaheuristics.* John Wiley & Sons, Inc.

Toffolo, T. A., T. Vidal, and T. Wauters (2019, may). Heuristics for vehicle routing problems: Sequence or set optimization? *Computers & Operations Research 105*, 118–131.

Toth, P. and D. Vigo (2002, jan). 1. an overview of vehicle routing problems. In *The Vehicle Routing Problem*, pp. 1–26. Society for Industrial and Applied Mathematics.

Tu, W., Q. Li, Q. Li, J. Zhu, B. Zhou, and B. Chen (2017, mar). A spatial parallel heuristic approach for solving very large-scale vehicle routing problems. *Transactions in GIS 21*(6), 1130–1147.

Uchoa, E., D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian (2017a). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research 257*(3), 845–858.

Uchoa, E., D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian (2017b, mar). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research 257*(3), 845–858.

Vazirani, V. V. (2003). *Approximation Algorithms.* Springer Berlin Heidelberg.

Vigo, D. (1996, feb). A heuristic algorithm for the asymmetric capacitated vehicle routing problem. *European Journal of Operational Research 89*(1-2), 108–126.

Wang, C., D. Mu, F. Zhao, and J. W. Sutherland (2015, may). A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering 83*, 111–122.

Wren, A. and A. Holliday (1972, September). Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Research Quarterly (1970-1977) 23*(3), 333.

Xavier, I., E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, A. Subramanian, T. Vidal, and D. Oliveira (2019). Cvrplib - all instances.

Yellow, P. C. (1970, jun). A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly (1970-1977) 21*(2), 281.

Zhong, Y. and M. H. Cole (2005, March). A vehicle routing problem with backhauls and time windows: a guided local search solution. *Transportation Research Part E: Logistics and Transportation Review 41*(2), 131–144.