Université des Sciences et Technologies de Lille – Lille 1

Département de formation doctorale en informatique **École doctorale SPI Lille**
UFR IEEA

Solving Dynamic Vehicle Routing Problems: From Single-Solution Based Metaheuristics to Parallel Population Based Metaheuristics

Résolution des Problèmes Dynamiques de Tournées de Véhicules: De Métaheuristiques à Base de Solution Unique aux Métaheuristiques Parallèles à Base de Population

THÈSE

présentée et soutenue publiquement le **02 décembre 2011**

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Mostepha Redouane Khouadjia

Composition du jury

Président :	– Gilles Goncalves, Professeur des Universités, Université d'Artois, France.
Rapporteurs :	– Pascal Bouvry, Professeur des Universités, Université du Luxembourg. – Van-Dat Cung, Professeur des Universités, INPG, France. – Geir Hasle, Research Director, SINTEF Applied Mathematics, Oslo, Norvège.
Examinateurs :	– Gabriel Luque, Assistant Professor, Université de Málaga, Espagne. – Dominique Feillet, Professeur des Universités, EM de Saint-Etienne, France.
Directeur de thèse :	– El-Ghazali Talbi, Professeur des Universités, Université Lille-1, France.
Co-Encadreur de thèse :	– Laetitia Jourdan, Professeur des Universités, Université Lille-1, France.

Laboratoire d'Informatique Fondamentale de Lille — UMR USTL/CNRS 8022
INRIA Lille - Nord Europe

Numéro d'ordre: 40638

Abstract

Many problems in the real world have dynamic nature and can be modeled as dynamic combinatorial optimization problems. However, research on dynamic optimization focuses on continuous optimization problems, and rarely targets combinatorial problems. One of the applications in dynamic combinatorial problems that has received a growing interest during the last decades is the on-line or dynamic transportation systems. A typical problem of this domain is the Dynamic Vehicle Routing Problems (DVRPs). In this latter, the dynamism can be attributed to several factors (weather condition, new customer order, cancellation of old demand, vehicle broken down, etc.). In such application, information on the problem is not completely known a priori, but instead is revealed to the decision maker progressively with time. Consequently, solutions for different instances have to be found as time proceeds, concurrently with managing the incoming information. Such problems call for a methodology to track their optimal solutions through time.

In this thesis, the dynamic vehicle routing problem is addressed and developing general methodologies called metaheuristics to tackle this problem is investigated. Their ability to adapt to the changing environment and their robustness are discussed.

First, a definition and a description of the problem as well as measures of algorithm performance are reviewed. Then, mechanisms to improve the ability of the algorithm to efficiently track optima shifting due to environmental changes are investigated. These methods include adaptive changing of the neighborhood in single solution based metaheuristics (S-metaheuristics), an adaptive memory mechanism for population based metaheuristics (P-metaheuristics), the use of parallel multi-populations approach as an additional means to control diversity and the incorporation of flexibility into metaheuristics in order to obtain robust algorithms able to produce good solutions by anticipating future changes.

Experimental results demonstrate that the methods are effective on this problem and hence have a great potential for other dynamic combinatorial problems.

Key-words: Dynamic vehicle routing problem, single-solution based metaheuristics, population-based metaheuristics, multi-population based metaheuristics, flexibility and robustness.

Résumé

Beaucoup de problèmes dans le monde réel ont une nature dynamique et peuvent être modélisés comme des problèmes dynamiques d'optimisation combinatoire. Cependant, les travaux de recherches sur l'optimisation dynamique se concentrent essetiellement sur les problèmes d'optimisation continue, et ils ciblent rarement les problèmes combinatoires.
Une des applications dans le domaine des problèmes dynamiques combinatoires ayant reçu un intérêt croissant au cours de ces dernières décennies est le système de transport en ligne où dynamique. Un problème typique de ce domaine est le Problème Dynamique de Tournées de Véhicules (PDTV). Dans ce dernier, le dynamisme peut être attribué selon plusieurs facteurs (conditions météorologiques, nouvelle commande client, annulation d'une commande précédente, véhicule tombant en panne, etc.). Dans un tel problème, les informations ne sont pas complètement connues a priori, mais plutôt révélées au décideur progressivement avec le temps. Par conséquent, les solutions des différentes instances doivent être trouvées au fur et à mesure du temps simultanément avec les informations entrantes. Ces problèmes font appel à une méthodologie capable de suivre les solutions optimales au cours du temps. Dans cette thèse, le problème dynamique de tournées de véhicules est étudié et le développement de méthodologies générales appelées métaheuristiques pour sa résolution est traité. Leur capacité à s'adapter à l'évolution de l'environnement et leur robustesse sont discutées.
En premier, une définition et une description du problème ainsi que les mesures de performance des algorithmes sont passées en revue.
Ensuite, les mécanismes permetant l'amélioration de la faculté de l'algorithme pour suivre efficacement le déplacement de l'optimum sont étudiés. Ces méthodes incluent le changement adaptatif du voisinage dans les métaheuristiques à base de solution unique (S-métaheuristiques), des mécanismes de mémoire adaptative pour les métaheuristiques à base de population (P-métaheuristiques), l'utilisation des approches multi-populations parallèles comme moyen supplmentaire de contrôle de diversité et enfin l'intégration de la flexibilité dans les métaheuristiques afin d'obtenir des algorithmes robustes capables de produire de bonnes solutions en anticipant les changements futurs.
Les résultats des expérimentations montrent que les méthodes sont efficaces sur ce problème et ont donc un grand potentiel pour d'autres problèmes combinatoires dynamiques.

Mots-clés: problème dynamique de tournées de véhicules, métaheuristiques à base de solution unique, métaheuristiques à base de population, métaheuristiques à base de multi-population, flexibilité et robustesse.

Acknowledgments

I would like to extend my sincerest gratitude to all people who have assisted me in completing this thesis. In particular, I would like to thank the following individuals.

Thanks to Professor Laetitia Jourdan and Professor El-Ghazali Talbi for supervising this work, for their background knowledge and for their support and guidance throughout the research. To them my warmest thanks for spending and sharing their time with me during these years.

Thanks to the members of my thesis committee who have taken the time to read the thesis and provide valuable comments on my work.

I am thankful for the financial support from the Associated Team Program MOMDI, which allowed me to do several internships in Spain into NEO research group and where I met excellent researchers who agreed to host me in their research lab in order to learn from them.

Thanks to Professor Enrique Alba who roused up my interest in dynamic optimization, and with whom I have shared many fruitful discussions.

I was very lucky to meet in this research group a wonderful colleagues and friends and I want to thank them all for their interesting views on different issues: Briseida, Juan José, Guillermo, José Manuel, Paco, Jamal, Francisco, Javier and Pablo.

Thanks to my colleagues and the staff at INRIA Lille and Dolphin research group for the friendly atmosphere, with special thanks to Marie-Eléonnore, Moustapha, Mathieu, Yacine, Ahcéne, Karima, Imen, Thé Van, Khadidja, Nadia and Inés.

Last, but not least, I am very grateful to my whole family for supporting me in this long journey, and especially to my parents who think of any accomplishment of mine as their own.

Contents

List of Figures

List of Tables

Introduction

This Ph.D. thesis focuses on solving dynamic combinatorial problems and particularly logistic and transportation problems. It is the results of three years held in Dolphin[1] research group of the French National Institute for Research in Computer Science and Control INRIA Lille Nord Europe and Lille's Computer Science Laboratory (LIFL, CNRS, Lille-1 University).

Many complex real-world problems are dynamic, and change over time, whether their objective function, decision variables, or the constraints. This involves that the optimal solution might change at any time due to changes in the environment. These problems are grouped into a class known as Dynamic Combinatorial Optimization Problems (DCOPs). This dynamism can be attributed to several factors; natural (weather conditions), human and material (absence and sickness of workers, machines broken down), or business factors (new job opportunity, cancellation of old ones, production and quality changing, etc.). In such applications, information on the problem is not completely known a priori, but instead is revealed to the decision maker progressively with time. Consequently, solutions for different instances of a typical dynamic problem have to be found as time proceeds, concurrently with the incoming information. Such problems call for a methodology to track their optimal solutions through time.

Nowadays, real-time information and communication systems become increasingly available and the processing of real-time data becomes increasingly affordable, more and more new versions of highly dynamic real-world applications are created.

In distribution systems (repair services, courier mail services, dial-a-ride services, etc.) most operations become under strict temporal restrictions. In addition, recent advances in information and communication technologies, vehicle fleets can now be managed in real-time. When jointly used, devices like geographic information systems (GIS), global positioning systems (GPS), traffic flow sensors and cellular telephones are able to provide real-time data, such as current vehicle locations, new customer requests, and periodic estimates of road travel times. This large amount of data can be used to reduce the cost and improve the service level of companies, and might provide revised routes that can be timely generated as soon as new events occur. This fact has caused an increasing interest in dynamic or on-line transportation models and systems in which data are considered to be time-dependent.

[1] Discrete multi-objective Optimization for Large scale Problems with Hybrid dIstributed techNiques.

One of the problems that has received a growing interest during the last years in on-line transportation systems is the Dynamic Vehicle Routing Problems (DVRPs). The traditional Vehicle Routing Problem (VRP) consists in constructing minimum cost routes for the fleet of vehicles to serve a set of customers so that they are visited exactly once.

There are many applications in which the problem can be considered on-line. To name a few examples; the routing of police cars, taxi services or ambulances. Furthermore, there are even applications that would traditionally be considered offline, but could be subject to sudden changes, like new customers appearing that urgently need service, traffic accidents making the planned routes impossible, changes in the demands of customers, changed traveling times due to heavy traffic, vehicles breaking down, etc. We consider here the Dynamic Capacitated Vehicle Routing Problem (DCVRP) where the customer orders are unknown when the optimization process begins, i.e. their orders and positions will be known only after the vehicles have left the starting node. Thus, the initial problem's specification can change while the vehicles are serving their previously assigned customers. This involves that the optimal solution might change at any time due to these new customer demands.

In the literature, many approaches for solving DVRP can be found. Some of them correspond to simple, yet specialized, constructive/improvement heuristics, while the rest represent sophisticated metaheuristics approaches. During the last decade, metaheuristics are raising a large interest in dynamic optimization problems and particularly on-line transportation domain. They represent more general approximate algorithms applicable to a large variety of optimization problems. They provide acceptable solutions in a reasonable time for solving hard and complex problems in science and engineering. Metaheuristics solve instances of problems by exploring large solution search space of these instances. These algorithms achieve this by reducing the effective size of the space and by exploring that space efficiently. Mainly, they can be grouped into two classes of algorithms; single-solution based metaheuristics and population based metaheuristics.

This thesis is concerned with the use of metaheuristics to tackle the dynamic vehicle routing problem with dynamic immediate requests. It provides a state of the art and latest research on dynamic vehicle routing problems, and how metaheuristics may be applied to face this kind of problems.

The research presented in this thesis has progressed in three phases. In the first phase, state-of-the-art that covers a description and specificities of the problem is given. Dynamic performance measures are presented in order to quantify the adaptation of the algorithms throughout the optimization process. Besides, the dynamic benchmarks of the problem and their development and generation are reported to guide the future experiments.

The second, the main phase, covers the development of adaptive algorithms on DVRP. These algorithms cover the different classes of metaheuristics.

A class of approaches seem to be interesting for this problem is the Single-solution based metaheuristics (S-metaheuristics). They could be viewed as search trajectories over the search space of the problem. These trajectories are performed by iterative procedures that move from the current solution to another one in the search space.
One of the recent approaches in the filed of S-metaheuristic is Variable Neighborhood Search (VNS). It consists in adaptively changing the neighborhood in order to get different local optima and to escape from local optima. In addition, the characteristic of changing the neighborhood structure might offer a powerful mechanism of adaptivity to the environmental changing. Since different neighborhoods generate different landscapes. Moreover, a solution that is locally optimal on the search landscape with respect to a neighborhood is probably not locally optimal with respect to another neighborhood and the global optima is one of the local optima of a given neighborhood.

For that purpose, we address in this thesis the class of S-metaheuristics represented by Variable Neighborhood Search for solving Dynamic Vehicle Routing Problem.

Another class of metaheuristics might adapt to the changing in different ways. Indeed, a standard approach to deal with dynamism in optimization problems is to regard each change as the arrival of new problem instance that has to be solved from scratch. However, this simple idea is often impractical. On the one hand, it could be too time-consuming. On the other hand, environmental changes in real life typically do not alter the problem completely, but affect only some part of the problem at a time. For example, not all vehicles break down at once, not all pre-made assignments are canceled, weather changes affect only parts of roads, any other events like sickness of employees and machine breakdown do not happen all at once. Thus, after an environmental change, there remains some information from the past that can be used in the future. The required algorithm should not only be able to track combinatorial problems, but should also be adaptive to changes in the environment.

Population-based metaheuristics (P-Metaheuristics) exhibit a number of potential advantages for such purposes. They start from an initial population of solutions and iteratively apply the generation of a new population and the replacement of the current one.
Particle Swarm Optimization (PSO) is one of the most commonly used P-Metaheuristic for solving dynamic continuous problems as it proved to be effective solvers for a broad range of these problems. Second, there are several characteristics inherent and attributed to PSO that encourage their use for

dynamic problems.

For this purpose, we present in this thesis an Adaptive Particle Swarm Optimization for solving Dynamic Vehicle Routing Problem. The underlying principle this approach is based on swarm intelligence, and hence they are expected to be capable of adaptation to environmental changes. In addition, PSO has proved to be suitable for dynamic environments due to their ability to store and exploit previous solutions. One of the most appealing features for dynamic environments is that, at any given instant, it deals with a population of solutions and even if the environment changes, it is likely that some solutions in the population remain feasible and retain some of their good quality. These solutions could be useful for tracking the new optimum since the dynamic change may cause the optima to be in the neighborhood of an old solution more often.

However, the main problem with P-metaheuristics used for dynamic optimization problems appears to be that they eventually converge to a non-global optimum and thereby lose their diversity necessary for efficiently exploring the search space. Consequently, also their ability to adapt to a change in the environment when such a change occurs. Therefore, approaches should counterbalance the effect of diversity loss by maintaining diversity throughout the run. This may be achieved by a Multi-Population based Metaheuristics (MP-Metaheuristics). In multi-population approach, a part of the population clusters around any local optimum it may discover, and remains close to this optimum for further exploration. The remainder of the population continues to search for new local optima, and the process is repeated if any more local optima are found. Furthermore, the subpopulation can cooperate and exchange information during the search and be more reactive to the next changing. Besides, parallelizing such a metaheuristics in real-time context is an important aspect due to the hard requirement on search time especially when we deal with strong dynamic problems in which changes occur in repeated manner and within short intervals.

Thus, we investigate in this thesis whether a multi-population metaheuristic might also be beneficial in dynamic vehicle routing problems. For this purpose, we elaborate a Multi-Swarm Optimization approach and evaluating it on parallel architecture.

The third phase aims to demonstrate that developing a potentially effective algorithms is not enough. One should aim at creating robust solutions that maintain high solution quality even when the environment changes. This robustness can be implemented by a flexibility measure that allows the anticipation of changes (future customer orders) and to explicitly search for solutions that not only have high quality, but that allow the adaptation of a high quality solutions after the environment has changed. S-Metaheuristics

seem to be good candidates for implementing this robustness since they tend to be more effective in terms of intensification than diversification in the search which can lead to loss their ability to track the shifting optimum.

Therefore, the last objective of this thesis is to integrate the flexibility measure into a S-metaheuristic such as Variable neighborhood search in order to build a solution that might anticipate the forthcoming arrival of new orders. In our context, if new requests are expected, rather than just react to the new demand, one should anticipate a change by trying to maintain flexibility. As we will show, for the dynamic vehicle routing problem such flexibility can be maintained by an early assignment of vehicles in the service area. This allows to increase the availability of vehicles for the new customer orders leading to decrease the number of vehicle detours or new assignment of vehicle that can increase the solution cost.

All these aspects are addressed in this thesis, providing a holistic view on the challenges and opportunities of applying metaheuristics on Dynamic Vehicle Routing Problems (DVRP), and suitable novel approaches are developed for each aspect.

The structure of this thesis is described in what follows:

- **Chapter 1** gives a general overview on Dynamic Vehicle Routing Problems (DVRPs). Different related approaches from single solution based metaheuristics to population based metaheuristics are outlined. Afterwards, performance measures defined in the literature for dynamic optimization problems are presented. Finally, available benchmark data sets of the problem are presented as well as our new class of benchmarks and their generation.

- **Chapter 2** deals with solving the dynamic vehicle routing with single solution based metaheuristics. It begins with the common concepts of this class of metaheuristics. Then, Variable Neighborhood Search (VNS) is proposed for solving DVRP. A comparison with respect to the quality of the solutions and execution time of our approach and state-of-the-art metaheuristics is done. Lastly, dynamic performances of our approach are assessed.

- **Chapter 3** concerns the design and implementation of population-based metaheuristics for solving our problem. The common and specific search concepts of this class of metaheuristics are outlined. An Adaptive Particle Swarm Optimization (APSO) is proposed. Comparison is carried out to investigate the merits of adapting solutions to changes by using adaptive memory mechanism. The adaptivity of our algorithm is assessed throughout dynamic performance indicators.

- **Chapter 4** explores the possible ways that researchers have to produce parallel multi-population approaches to solve dynamic vehicle routing problems by using Multi-Adaptive Particle Swarm Optimization (MAPSO). A unified cooperative parallel model for multi-population metaheuristics is analyzed in terms of design. Experimental results are provided to evaluate the dynamic as well as parallel performances of our approach.

- **Chapter 5** addresses the issue of finding solutions that are not only optimal with respect to the current situation, but also with respect to the expected changes in the environment. The aim is to find robust solutions and flexible solutions that allow easy and successful adaptation by anticipating future customer needs. As described previously, a flexibility strategy is suggested and integrated in a Flexible Variable Neighborhood Search (FVNS) that, when taken into account during optimization, may yield significantly better results in a dynamic environment with new orders arriving over the time. This strategy is validated with experiments on conventional set of benchmarks.

This thesis concludes with a summary of our contributions and an outlook on future work.

Chapter 1

Dynamic Vehicle Routing Problems: Overview, Approaches, and Performance Measures

Contents

1.1 Introduction

Thanks to recent advances in information and communication technologies, vehicle fleets can now be managed in real-time. When jointly used, devices like geographic information systems (GIS), global positioning systems (GPS), traffic flow sensors and cellular phones are able to provide real-time data, such as current vehicle locations, new customer requests, and periodic estimates of road travel times. If suitably processed, this large amount of data can be used to reduce the cost and improve the service level of a modern company. To this end, revised routes have to be timely generated as soon as new events occur [Ghiani 2003].

In this context, Dynamic Vehicle Routing Problems (DVRPs) are getting increasingly important [Psaraftis 1995, Montemanni 2005b, Housroum 2006, Hanshar 2007]. These problems are also known as on-line or real-time Vehicle

Routing Problems. The VRP [Dantzig 1959] is a well-known combinatorial problem which consists in designing routes for a fleet of capacitated vehicles that are to service a set of geographically dispersed points (customers, stores, schools, cities, warehouses, etc.) at the least cost (distance, time, or any other desired factor). It is possible to define several dynamic features which introduce dynamism in the classical VRP: roads between two customers could be blocked off, customers could modify their orders, the travel time for some routes could be increased due to bad weather conditions, etc. This implies that Dynamic VRPs constitute in fact a set of different problems, which are of crucial importance in today's industry, accounting for a significant portion of many distribution and transportation systems.

The main goal of this chapter is to present the problem of DVRP and methods from literature for its resolution. The remainder of this chapter is organized as follows. Section 1.2 describes the dynamic VRP and its specific characteristics. A succinct overview on the most proposed solution representations is given in Section1.3. A summary on solving methods from strategies through heuristics and metaheurtistics is given in Section 1.4. In order to measure the dynamic performances of approaches, Section 1.5 presents some measures that can be used to this end. The most popular and available benchmarks are reported in Section 1.6, and finally, Section 1.7 presents conclusion and opens some lines for next chapters.

1.2 Dynamic Vehicle Routing Problem

In this section, we present the problem definition and its formal description in Section 1.2.1 and Section 1.2.2. Then, Section 1.2.3 addresses how to quantify the dynamism into such problem. Besides, Section 1.2.4 compare both static and dynamic versions of the problem in terms of characteristics, and Section 1.2.5 presents some interests of the problem. Finally, a state-of-the art on the common variants of the problem is presented in the Section 1.2.6.

1.2.1 Problem Definition

Psaraftis [Psaraftis 1995, Psaraftis 1988] was among the first to study the Dynamic Vehicle Routing Problem (DVRP). He defines the static problem as: "*if the output of a certain formulation is a set of preplanned routes that are not re-optimized and are computed from inputs that do not evolve in real-time*". While he refers to a problem as dynamic if "*the output is not a set of routes, but rather a policy that prescribes how the routes should evolve as a function of those inputs that evolve in real-time*".

This definition is elaborated in Larsen *et al.* [Larsen 2008], in which the problem is said dynamic when not all information relevant to the planning of the routes is known by the planner when the routing process begins. Besides, the information can change after the initial routes have been constructed. On the counterpart, in the static vehicle routing problem, information is assumed to be relevant, includes all attributes of the customers such as the geographical location of the customers, the service time at each customer and all the details about customer demand. Furthermore, the travel times of the vehicle between the customers must be known by the planner.
From the above definitions, we can understand that when some inputs to the problem are revealed during the execution of the algorithm. Thus, it is not possible to determine in advance a set of optimized routes in a dynamic problem. Problem solution evolves as inputs are revealed to the algorithm and to the decision maker.
Obviously, the DVRP is a richer problem compared to the conventional static VRP. If the problem class of VRP is denoted $P(VRP)$ and the problem class of DVRP is denoted $P(DVRP)$, then $P(VRP) \subset P(DVRP)$. Given that SVRP belongs to the class of NP-hard Problems, the DVRP belongs also to this class, and it is more complicated than SVRP since a static problem should be solved each time a new customer demand is received.

1.2.2 Formal Description

The conventional VRP can be mathematically modeled by using an undirected graph $G = (C, E)$, where C is a vertex set, and E is an edge set. They are expressed as $C = \{c_0, c_1, ..., c_n\}$, and $E = \{(c_i, c_j)|c_i, c_j \in C, i < j\}$. Furthermore, a set of m homogeneous vehicles each with capacity Q originate from a single depot, represented by the vertex c_0 and must service all the customers represented by the set C. The quantity of goods q_i requested by each customer i ($i > 1$) is associated with the corresponding vertex. The goal is to find a feasible set of tours with the minimum total traveled distance. The VRP thus consists in determining a set of m vehicle routes of minimal total cost, starting and ending at a depot, such that every vertex in C is visited exactly once by one vehicle. The sum of the items associated with the vertexes contained in it never exceeds the corresponding vehicle capacity Q. The capacity means the quantity of items (goods) that the vehicle could carry during its travel. Let be $R_1, \ldots, R_m$ a partition of C representing the routes of the vehicles to service all the customers. The cost of a given route $R_j = \{c_0, c_1, \ldots, c_{k+1}\}$,

where $c_i \in C$ and $c_0 = c_{k+1}$ (denote the depot), is given by:

$$Cost(R_j) = \sum_{i=0}^{k} d_{i,i+1} \tag{1.1}$$

and the cost of the problem solution (S) is:

$$F_{VRP}(S) = \sum_{j=1}^{m} Cost(R_j) \tag{1.2}$$

With a constraint on the vehicle capacity:

$$\sum_{i=1}^{k} q_i \times y_i^j \leq Q^j \tag{1.3}$$

Where:
q_i: the associated quantity of the customer c_i (items to be delivered/picked up),
Q^j: capacity of the vehicle j, and

$$y_i^j = \begin{cases} 1, \text{ if } c_i \text{ is served by the vehicle } j \\ 0, \text{ otherwise} \end{cases} \tag{1.4}$$

We will consider a service time δ_i (time needed to unload/load all goods), required by a vehicle to load the quantity q_i at c_i. It is required that the total duration of any vehicle route (travel plus service times) may not surpass a given bound T, so, a route $R_j = \{c_0, c_1, \ldots, c_{k+1}\}$ is feasible if the vehicle stops exactly once in each customer and the travel time of the route does not exceed a prespecified bound T corresponding to the end of the working day.

$$\sum_{i=0}^{k} d_{i,i+1} + \sum_{i=1}^{k} \delta_i \leq T \tag{1.5}$$

There may exist some restrictions such as the capacity of each vehicle, total traveling distance allowed for each vehicle, time windows to visit the specific customers, and so forth. The basic VRP deals with customers who are known in advance; all other information such as the driving time between the customers and the service times at the customers are also usually known prior to the planning.

The Dynamic Vehicle Routing Problem (DVRP) [Psaraftis 1995] is strongly related to the static VRP, as it can be described as a routing problem in which information about the problem can change during the optimization process. Any dynamic combinatorial problem can be expressed as

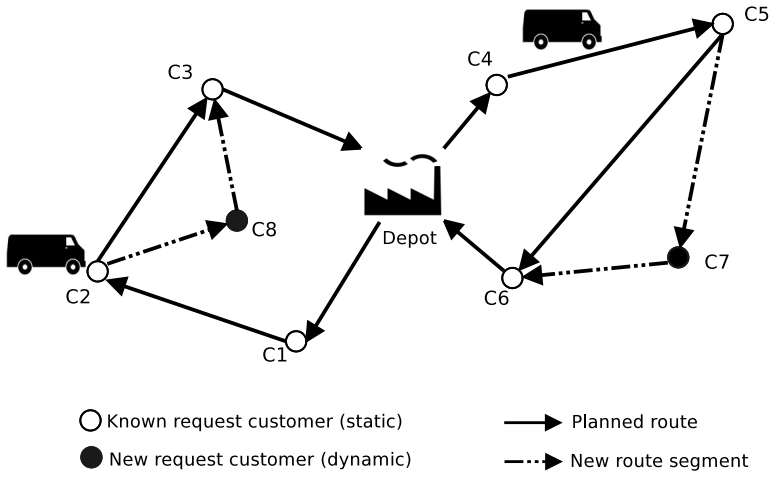


Figure 1.1: A dynamic vehicle routing with dynamic requests case.

$P(t) = (X(t), f(t))$, which is a time dependent formulation of the previous definition of the static VRP. Thus, in that definition, any components of the problem can change with time. Whether the objective function, decision variables, or the constraints. This involves that the optimal solution might change at any time due to changes in the environment. The problem cannot be solved in advance because the decision maker does not have a priori knowledge of the entire problem [Bianchi 2009]. Therefore, the goal of the optimization process is no longer finding a single optimal solution, but rather, tracking the shifting optima over the time since the optimal solution for one instance could be a poor solution or possibly even infeasible for the next environment.

Thus, a discrete-time dynamic problem and can be viewed as a series of P instances; each instance is a static problem, which starts at time t and must be solved within a specific deadline $\triangle_t$. We summarize that as follows (Figure 1.2):

$$P = \{(P_i, t_i, \triangle_t) \ / \ i = 0, 1, \ldots, i_{max}\} \tag{1.6}$$

with this information the duration of the instance i is $\triangle_i = t_{i+1} - t_i$. The maximum number of instances i_{max} can be infinite if the problem is open-ended. A new instance P_{i+1} is generated by the action of the environment change δ_{i+1} on the instance P_i. This is expressed by $P_{i+1} = P_i \oplus \delta_{i+1}$. So, the solutions obtained at time t_i could not be the feasible solutions for time t_{i+1} .

This change in the environment can be due to several factors; for example, travel times can be time- [Haghani 2005] or traffic-dependent [Wang 2007], orders may be withdrawn or changed [Sun 2007], some clients may be unknown when the execution begins [Magalhães 2006], etc. One standard approach to deal with this changing is to consider the entire problem as a series of instances. Each change corresponds to the arrival of new optimization problem that has

to be solved. The time consecrated for solving each instance depends on the frequency of changes [Branke 2002].

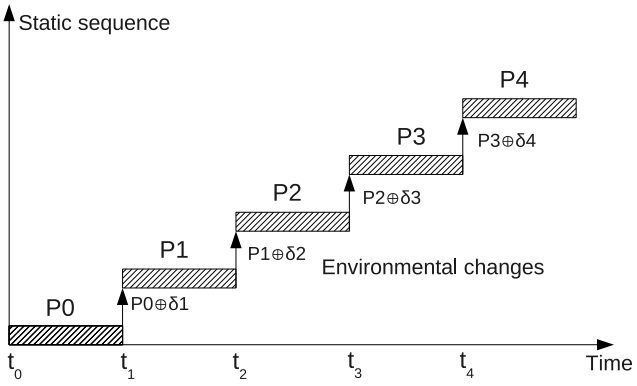


Figure 1.2: Decomposition of a dynamic problem $P = (P_1, P_2, P_3, P_4)$ in a sequence of static instances.

The goal is to design an optimization algorithm that is capable of continuously adapting the solution to a changing environment. This approach is now commonly followed by the community that works on the DVRP domain [Kilby 1998, Montemanni 2005b, Housroum 2006, Hanshar 2007].
A simple example of a dynamic vehicle routing situation is shown in Figure 1.2.2. In the example, two un-capacitated vehicles must service both known and new request customers.

1.2.3 Degree of Dynamism

Designing a real-time routing algorithm depends to a large extent on how much the problem is dynamic. The dynamic aspect of the problem concerns the number of dynamic events, their location and the time in when these events take place. To quantify this concept, Lund *et al.* [Lund 1996] and Larsen [Larsen 2000] have defined the degree of dynamism of a problem (*dod*). Without loss of generality, we assume that the planning horizon is a given interval $[0, T]$, possibly divided into a finite number of time slices. Let n_s and n_d be the number of static and dynamic requests, respectively. Moreover, let $t_i \in [0, T]$ be the time when the request i appears. Static requests are such that $t_i = 0$ while dynamic ones have $t_i \in]0, T]$. Lunda *et al.* [Lund 1996] define the degree of dynamism *dod* as :

$$dod = \frac{n_d}{n_s + n_d} \tag{1.7}$$

which may vary between 0 and 1. Larsen [Larsen 2000] generalizes the definition proposed by Lund *et al.* in order to take into account both dynamic

request occurrence times and possible time windows. He observed that system in which dynamic requests are received late of the planning horizon $[0, T]$ is more dynamic than others in which the requests occur at the beginning of the working day. Thus, he introduces a new measure of dynamism :

$$edod = \frac{\sum_{i=1}^{n_s+n_d}(t_i/T)}{n_s + n_d} \tag{1.8}$$

The effective degree of dynamism then represents an average of how late the requests are received compared to the latest possible time the requests could be received. It can easily be seen that *edod* ranges between 0 and 1. It is equal to 0 if all user requests are known in advance while it is equal to 1 if all user requests occur at time T .
Finally, Larsen extends the definition of *edod* to take into account possible time windows on user service time. Let $[a_i, b_i]$ be the interval time of the client i referred as time window (tw), with a_i and b_i corresponding to the earliest time the service begin and the latest possible time that service should begin, respectively.

$$edod_{tw} = \frac{\sum_{i=1}^{n_s+n_d}[T - (b_i - t_i)]/T}{n_s + n_d} \tag{1.9}$$

It is also obvious that $edod_{tw}$ varies between 0 and 1. Moreover, if no time windows are imposed (i.e., $a_i = t_i$ and $b_i = T$), then $edod_{tw} = edod$.

Thus, the degree of dynamism measure gives rise to different dynamic levels. It is possible to categorize the vast majority of routing systems found in practice by using three echelons. We can discern between weakly, moderately, and strongly dynamic systems. For instance, supply and distribution companies (such as those of distributing heating oil) are known as weakly dynamic. Couriers and appliance repair service companies exhibit moderate dynamic behavior. Finally, emergency services and taxicab services are strongly dynamic [Larsen 2000].

1.2.4 Dynamic Versus Static

Dynamic problems are typically derived from static ones, by revealing or updating on-line one or more parameters of those which define static instance. In the following we point out some important differences between static and dynamic problems [Bianchi 2000].

Time The main feature of dynamic problems, which is not present in static ones is the dimension of time as integral part of the instance description. The dispatcher must as a minimum know the position of all vehicles at

any given moment and particularly when the request for service or other information is received by the dispatcher.

Future information In a static problem all information is assumed to be known and of the same quality. In a real-life dynamic routing problem information about the input instance may be in part given a priori, and in part dynamically revealed or updated. The future is almost never known with certainty. It may be either completely unknown or at least partially known under certain conditions with some probabilistic assumptions. Bianchi [Bianchi 2000] preconizes that information update mechanisms have to be integrated into the solution methods.

Objective function One may consider the dynamic problem as a series of static problems (sub-problems), with the goal of tracking the objective function optimum as closely as possible throughout the time. However, if some stochastic information is related to the problem, they should be considered by the objective function. Depending on the nature of the system, the objective to be optimized is often a combination of different measures. DVRP inherits the classical objectives defined into the conventional VRP. Moreover, the dynamic nature of problem leads to emerging of new objectives. For instance, in the weakly dynamic systems the focus is on minimizing routing cost [Montemanni 2005b, Hanshar 2007]. However, in strongly dynamic system such as an emergency services, the interest is to minimize the expected response time (i.e. the expected time lag between the moment when the user request occurs and its service time) [Larsen 2002, Mitrović-Minić 2004a, Gendreau 2006]. Furthermore, there are other objectives such as maximizing the expected number of requests serviced during a given period of time [Bent 2003, Bent 2004].

Strategy Strategy implies which actions should be taken at each state of the problem progress. It can concern the way in which the vehicle have to be positioned in the service area (see Section 1.4.1), as well as, the manner in which the requests have to be handled. For instance, one can consider a static new sub-problem each time a new customer request arrives, or waiting a certain time period for a subset of new customers, or consider a spatial clustering of customers and so on.

1.2.5 DVRP Interests

There are several important problems that must be solved in real-time. In [Gendreau 1998, Larsen 2000, Ghiani 2003], the authors list a number of real-

life applications that motivate the research in the domain of dynamic vehicle routing problems.

- *Supply and distribution companies*: In seller-managed systems, distribution companies estimate customer inventory level in such a way to replenish them before stock depletion. Hence, demands are known beforehand in principle and all customers are static. However, because demand is uncertain, some customers might run out their stock and have to be serviced urgently.

- *Courier Services*: It refers to the international express mail services that must respond to customer requests in real-time. The load is collected at different customer locations and have to be delivered at another location. The package to be delivered is brought back to a remote terminal for further processing and shipping. The deliveries form a static routing problem since recipients are known by the driver. However, most pickup requests are dynamic because neither the driver nor the planner knows where the pickups are going to take place.

- *Rescue and repair service companies*: Repair services usually involve a utility firm (broken car rescue, electricity, gas, water and sewer, etc.) that responds to customer requests for maintenance or repair of its facilities.

- *Dial-a-ride systems*: Dial-a-ride systems are mostly found in demand-responsive transportation systems aimed at servicing small communities or passengers with specific requirements (elderly, disabled). These problems are of the many-to-many when any node can serve as a source or destination for any commodity or service.

  Customers can book a trip one day in advance (static customers) or make a request at short notice (dynamic customers).

- *Emergency services*: They cover the police, firefighting and ambulance services. By definition, the problem is pure dynamic since all customers are unknown beforehand and arrive in real-time. In most situations, routes are not formed because the requests are usually served before a new request appears. The problem then is to assign the best vehicle (for instance the nearest) to the new request. Solving methods are based on location analysis for deciding where to dispatch the emergency vehicles or to escape the downtown traffic jam.

- *Taxi cab services*: Managing taxi cabs is still another example of a real-life dynamic routing problem. In most taxi cab systems the percentage of

dynamic customers is very high, i.e., only very few customers are known by the planner before the taxi cab leaves the central at the beginning of its working day.

1.2.6 Related Works

In this section, we present a classification and an overview on the state-of-the-art of dynamic vehicle routing problems. Different surveys have been proposed on DVRPs [Bianchi 2000, Ghiani 2003, Branchini 2009]. Psaraftis [Psaraftis 1988] defines that a vehicle routing problem is dynamic when some inputs to the problem are revealed during the execution of the algorithm. Demand information is not known when vehicles are assigned, and demand information is revealed on-line. Problem solution evolves as inputs are revealed to the algorithm and to the decision maker. Possible information attributes might include evolution of information (static/dynamic), quality of information (known-deterministic/forecast/probabilistic/unknown), availability of information (local/global), and processing of information (centralized/decentralized).

From this definition, we propose to classify the DVRPs according to the degree of knowledge that we have on the input data of the problem and quality of the available information. A dynamic problem can be either deterministic or stochastic (see Figure 1.3). DVRP is deterministic if all data related to the customers are known when the customer demands arrive, otherwise it is stochastic. Both of these classes can be subject to different factors such as service time window, traffic jam, road maintenance, weather changes, breakdown of vehicles and so on. These factors often change the speed of vehicles and the travel time of arriving at the depot. Consequently, they lead to another sub-variants of the problem (see Table 1.1):

1. **Deterministic**: In deterministic case, all the data related to the inputs are known. For instance, when new customer demand appears, customer location and the quantity of his demand are known. Different types of deterministic DVRP can be found in the literature as:

   (a) **Dynamic Capacitated Vehicle Routing Problem (DCVRP):**

   An important number of works exist on this variant [Kilby 1998, Gendreau 1999, Montemanni 2005b] which represents the conventional definition of the problem, and where the existence of all customers and their localizations are deterministic, but their order can arrive at any time. The objective is

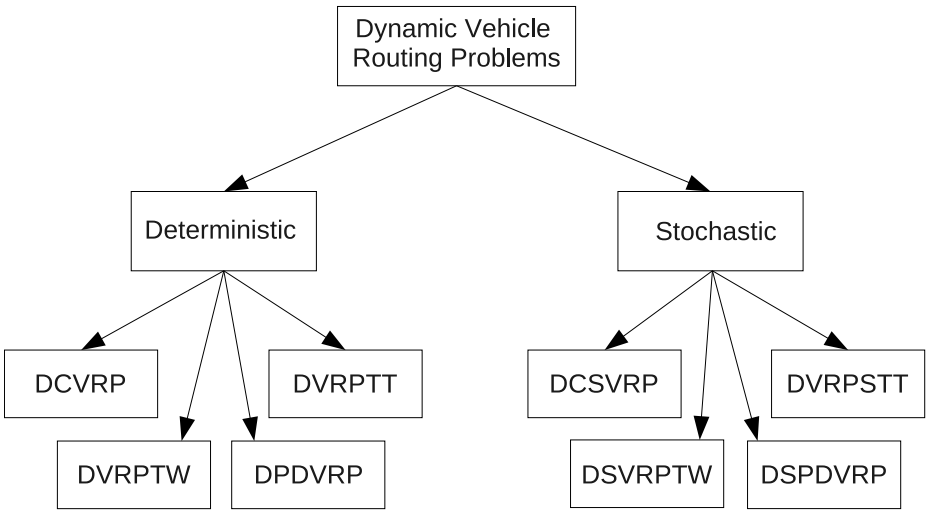


Figure 1.3: Classification of DVRPs according to deterministic and stochastic information related to customer requests.

to find a set of routes with the lowest traveled distance, and with respect to vehicle capacity limit.

The Dynamic Traveling Repairman Problem (DTRP) belongs to this class of problems. Bertsimas and Van Ryzin introduce this problem in [Bertsimas 1991, Bertsimas 1993b]. It is described as a problem in which demands arrive according to Poisson process in Euclidean service region, and their locations are distributed throughout the service region. The goal is to minimize the expected time that the demand spends in the system (i.e. the average time a customer must wait before its request is completed), as opposed to the expected distance that the vehicle travels. The service times of requests are not known to the dispatcher, until the service at the respective customers is completed.

Where all demands are dynamic in DTRP, i.e. all customers are immediate request customers. Larsen*et al.* [Larsen 2002] define the Partially Dynamic Traveling Repairman Problem (PDTRP) that is a variant of this problem involving both advance and immediate request customers.

Furthermore, the problems seek to optimize different objective functions. The dispatcher is more interested in minimizing the distance traveled by the repairman than to minimize the overall system time.

(b) **Dynamic Vehicle Routing Problem with Time Windows (DVRPTW):** It is one of the most well-studied variant of DVRP [Larsen 2004, Mitrović-Minić 2004a, Alvarenga 2005, Fabri 2006, Housroum 2006, Wang 2007, Oliveira 2008]. Besides, the possibility of requiring services in real time, the time window associated to each customer i follows a specific interval time $[a_i, b_i]$, that must be satisfied. Larsen*et al.* [Larsen 2002] proposed online policies for the Partially Dynamic Traveling Salesman Problem with Time Windows (PDTSPTW) that could be considered as an instance of DVRPTW with a single vehicle.

The objective is to minimize the lateness at customer location. A simple policy consists to require the vehicle to wait at the current customer location until it can service another customer without being early. Other policies, may suggest to reposition the vehicle at a location different from that of the current customer based on prior information on future requests.

(c) **Dynamic Vehicle Routing Problem with time-dependent Travel Times (DVRPTT):** Described in [Haghani 2005], it assumes that the travel times from the customer i to the customer j is variable through the time. This variation could occur due to the type of the road, weather and traffic conditions that may strongly influence the speed of vehicles and hence travel times.

(d) **Dynamic Pickup and Delivery Vehicle Routing Problem (DPDVRP):** Based on the conventional Pickup and Delivery Vehicle Routing Problem (PDVRP) [Savelsbergh 1995]. The problem consists of determining a set of optimal routes for a fleet of vehicles in order to serve transportation requests [Mitrović-Minić 2004a]. The objective is to minimize total route length, i.e., the sum of the distances traveled by all the vehicles, under the following constraints: all requests must be served, each request must be served entirely by one vehicle (pairing constraint), and each pickup location has to be served before its corresponding delivery location (precedence constraint). The dynamic version arises when not all requests are known in advance.

Swihart and Papastavrou [Swihart 1999] have introduced a new variant of the DTRP where each service request has a pickup and a delivery location. The objective is to minimize the expected system time. The authors consider the unit-capacity case where the vehicle can carry no more than one item, as well as the case where the vehicle can carry an arbitrarily large number of items.

Attanasio *et al.* present in [Attanasio 2004] parallel implementations of a tabu search method developed previously by Cordeau and Laporte [Cordeau 2003] for the Dynamic Dial-a-Ride Problem (DDARP). In this latter the requests are received throughout the day and the primary objective is to accommodate as many requests as possible according to the available fleet of vehicles. Furthermore, the routes are designed under the constraint that customers specify pick-up and drop-off requests between origins and destinations. Yang *et al.* [Yang 2004] introduce a real-time multi-vehicle truckload pickup and delivery problem. They propose a mixed-integer programming formulation for the off-line version of the problem and propose a new rolling horizon re-optimization strategy for a dynamic version.

2. **Stochastic**: In stochastic dynamic problems (also known as probabilistic dynamic problems) uncertain data are related to customer locations, demands or travel times and are represented by stochastic processes.

   (a) **Dynamic and Stochastic Capacitated Vehicle Routing Problem (DSCVRP)**: It considers customer requests are unknown and revealed over time. In addition, customer locations and service times are random variables and are realized dynamically during plan execution. Bent and Van Hentenryck [Bent 2003, Bent 2004] considered DVRP with stochastic customers. They proposed a multiple scenario approach that continuously generates routing plans for scenarios including known and immediate requests to maximize the number of serviced customers. The approach was adapted from Solomon benchmarks with varying the degree of dynamism. Hvattum *et al.* [Hvattum 2006] addressed this variant of the problem. The authors consider both customer locations and demands may be unknown in advance. They formulate the problem as a multi-stage stochastic programming problem, and a heuristic method was developed to generate routes by exploiting the information gathered on future customer demand.

   (b) **Dynamic and Stochastic Vehicle Routing Problem with Time Windows (DSVRPTW)**: It has been introduced in [Pavone 2009]. In this problem, each service request is generated according stochastic process; once a service request appears, it remains active for a certain deterministic amount of time, and then expires. The objective is to minimize the number of possible vehicles and ensure that each demand is visited before its expira-

tion. Furthermore, this problem has been considered by Bent *et al.* in [Bent 2007].

(c) **Dynamic Vehicle Routing Problem with Stochastic Travel Times (DVRPSTT)**:

It assumes that the problem is subject to a stochastic travel time which represents a random variable in an interval. The travel times change from one period to the next. Some works present this version of the problem as in [Potvin 2006], where the the travel time to the next destination is perturbed by adding a value generated with a normal probability law. This perturbation represents any unforeseen events that may occur along the current travel journey. It is known to the dispatching system only when the vehicle arrives at its planned destination.

(d) **Dynamic and Stochastic Pickup and Delivery Vehicle Routing Problem (DSPDVRP)**:In this version of the problem stochastic process concerns the quantity of demand that the vehicle must pick or delivery to each customer. Thus, we have vagueness in quantities to pick up or deliver at the customers' location [Xu 2008]. The demand of each customer is revealed only when the vehicle reaches the customer. The distribution can be modeled by using a probabilistic law, such as a normal law for example, or by using fuzzy logic.

Table 1.1: Major publications on different variants of Dynamic Vehicle Routing Problems.

DVRPs	Authors	Classes	Characteristics	Objectives
Deterministic	Psaraftis *et al.* [Psaraftis 1988] Kilby *et al.* [Kilby 1998] Montemanni *et al.* [Montemanni 2005b] Hanshar *et al.* [Hanshar 2007] Branchini *et al.* [Branchini 2009] Bertsimas and Van Ryzin [Bertsimas 1991, Bertsimas 1993b]	DCVRP	Dynamic requests Vehicle has a limited capacity	Minimization of the total traveled distance lateness at the customer
	Oliveira *et al.* [Oliveira 2008] Gendreau *et al.* [Gendreau 1999] Mitrović-Minić *et al.* [Mitrović-Minić 2004a] Larsen *et al.* [Larsen 2004] Housroum *et al.* [Housroum 2006] Alvarez *et al.* [Alvarenga 2005]	DVRPTW	Dynamic requests Time window	Minimize the total traveled distance and minimize the sum of lateness at the customer
	Haghani *et al.* [Haghani 2005]	DVRPTT	Dynamic requests Variable Travel Times	Minimize the sum of lateness at the customer
	Gendreau *et al.* [Gendreau 2006] Mitrović-Minić *et al.* [Mitrović-Minić 2004a] Angelelli *et al.* [Angelelli 2004] Yang *et al.* [Yang 2004] Swihart and Papastavrou [Swihart 1999]	DPDVRP	Dynamic requests Pickup and Delivery	Minimization of total travel time, lateness over all pick-up and delivery locations and sum of overtime over all vehicles
Stochastic	Bent *et al.* [Bent 2003, Bent 2004] Hvattum *et al.* [Hvattum 2006]	DSCVRP	Random customer locations Random service times	Maximize the number of serviced customers Minimize the total traveled distance
	Pavone *et al.* [Pavone 2009] Bent *et al.* [Bent 2007]	DSVRPTW	Random customer locations Random service times Time window	Minimize the number of vehicles, and the traveled distance Minimize the wait for completion of service
	Potvin *et al.* [Potvin 2006] Djadane *et al.* [Djadane 2006]	DVRPSTT	Dynamic requests and travel times are subjected to stochastic variations	Minimize the travel times, lateness at customer locations and lateness at the depot
	Xu *et al.* [Xu 2008]	DSPDVRP	Stochastic pickup and delivery service	Minimize the total distance covered by vehicles, and lateness at customer

1.3 Solution Representation

Different works have been achieved for solving the DVRP in the literature. Nevertheless, few works emphasize the representation used for problem's solution [Montemanni 2005b, Housroum 2006, Hanshar 2007].

The underlying idea is to use a dynamic structure with a variable length. The dynamic encoding is justified by the fact that demands arrive over the time and have to be inserted in the existing routes or by creating new ones. The representation must take into account the routes and information related to the customers as well as the vehicles. It must distinguish the pending customers that have newly been added to the day's schedule, but not yet assigned to any vehicle and committed customers that have already been visited by a given vehicle. In [Montemanni 2005b], Montemanni *et al.* propose a representation for their Ant Colony System (ACS) algorithm. The authors consider v dummy depots (one for each vehicle of the fleet) and they refer to them as $d_1, ..., d_v$. Solutions retrieved by ants will be represented as long, single tours. In this context, nodes contained within two consecutive dummy depots d_a and d_b (with d_a , $d_b \in \{1, ..., v\}$) form the (partial) tour associated with vehicle a. The partial tour associated with vehicle b will start from the dummy depot d_b, which corresponds to the location of the last customer committed to vehicle b. The starting time from d_b corresponds to the end of the serving time for the last customer committed to vehicle b, while the capacity of b will be equal to the residual capacity of b, i.e. Q_b minus the quantity ordered by customers already committed to vehicle b.
Another representation is proposed by Hanshar *et al.*[Hanshar 2007] for a Genetic Algorithm for DVRP. Their chromosomal representation consists of two types of nodes: a node with a positive integer number representing a single customer (who has not yet been assigned to a vehicle) and node depicted with a negative integer number representing a group of clustered customers that have already been committed to a given vehicle. Thus, the chromosome consists of integers, where new customers are directly represented on a chromosome with their corresponding positive index number and each committed customer is indirectly represented within one of the groups representing a given deployed vehicle. When the chromosome is decoded, new customers could be added to these pre-existing vehicles (i.e., groups) if they still have the capacity to accommodate new customer orders.
Always in this connection, Garrido and al. [Garrido 2010] have tackled the DVRP by an Evolutionary Hyper-Heuristics (EH-DVRP). The authors propose a chromosome representation for the low-level heuristics composed by two main data structures; a list of new unassigned customers represented by their identifier, and a set of routes which represents a set of partial solutions

or states of the problem, formed by committed and uncommitted requests.

1.4 Solving Methods

In solving Dynamic Vehicle Routing Problems, we have to take into account two important aspects; the events managing and the resolution process. The event managing task is in charge of collecting new demands and keeps trace of already served customers and of the position of the vehicles. It uses this information in order to generate a sequence of static VRP subproblems which are solved by one of the methods described below. Also, the event manager plays the role of scheduler which commits the vehicles to customers according to the solution obtained by the solving method. Different event managers have been developed in literature and have similar job [Montemanni 2005b, Housroum 2006, Hanshar 2007]. For the optimization task, it consists in providing feasible solutions according to the static problem input. In this section, we present a classification and a brief overview of the main methods proposed in literature for dynamic vehicle routing problems.
As seen in Figure 1.4, solving methods can be divided into three main categories: positioning strategies, heuristics, and metaheuristics.

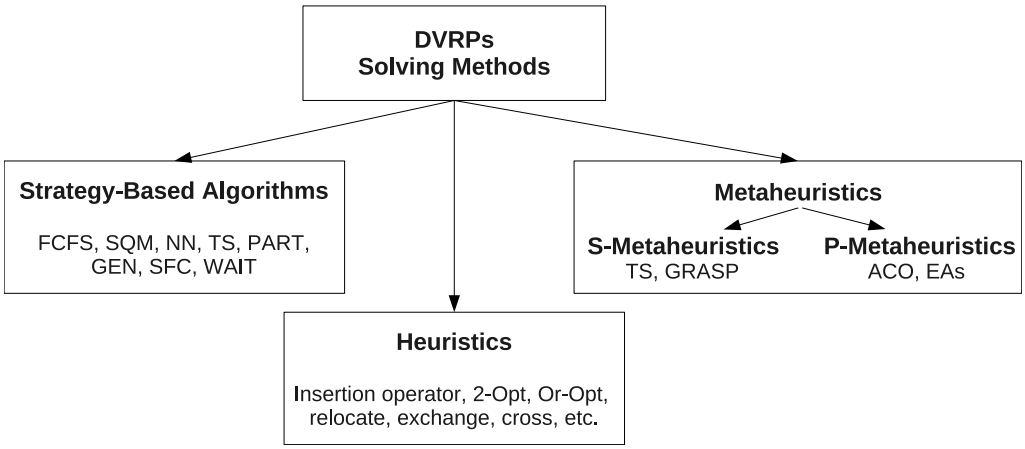


Figure 1.4: Classification of solving methods proposed in literature for DVRPs.

1.4.1 Strategy-Based Algorithms

Generally, the strategies are represented as simple policies, which specifie the actions that the system should bring to handle the current state and the

properties of the dynamic problem (for example, location of appearance, frequency of events, number of orders known beforehand, length of the working day, duration of the planning horizon, and so on) [Bianchi 2000, Ghiani 2003]. In [Bertsimas 1991, Bertsimas 1993a, Bertsimas 1993b, Papastavrou 1996, Swihart 1999, Song 2005, Song 2005, Branke 2005b] strategies have been defined for single or multi vehicles conditions. They consider the spacial and time distributions of the dynamic requests and enable the system to plan the service each time the problem changes.

Strategies are applied repeatedly in order to dispatch requests to vehicles and build routes. We outline some of them :

1. **First Come First Served (FCFS)**: Requests are served in the order in which they are received by the dispatcher.

2. **Stochastic queue median (SQM)**: Locate the vehicle at the median position in the service region and serve the customer according to FIFO strategy. When the service is finished, the vehicle returns to the median.

3. **Nearest Neighbor (NN)**: A greedy strategy in which the vehicle serves the nearest unserviced request after a service.

4. **Traveling Salesman (TS)**: Requests are collected into a sets of a given size. Once a set of demands has been batched the TSP is solved. The demands are serviced by following the optimal tour which start and end at the depot [Bertsimas 1993b, Bertsimas 1993a].

5. **Partitioning (PART)**: The service area is partitioned into sub-regions. A vehicle visits the sub-regions in a given order and with respect of passing from one sub-region to an adjacent one. Demands into each sub-region are serviced in a FIFO order [Bertsimas 1991].

6. **Generation (GEN)**: This strategy combines SQM and TSP strategies. It consists to position the vehicle at the median of the region service and serve the request when it arrives (generation). At the completion time, the vehicle returns to the median if there are no requests in a waiting queue. Otherwise, the TSP is solved on the existing requests and the vehicle is committed according to the optimal tour (next generation) [Papastavrou 1996].

7. **Space Filling Curve (SFC)**: It services customers as they are encountered in repeated clockwise sweeps of a circle which guides the routes construction [Bertsimas 1991, Bertsimas 1993b].

8. **Waiting (WAIT)**: It aims to find an optimal waiting schedule for the vehicles to maximize the probability that a new customer can be incorporated into one of the tours while reducing the average length of the detour that is necessary to serve this customer. Larsen *et al.*[Larsen 2004] proposed strategies to deal with the Partially Dynamic Traveling Salesman Problem with Time Windows (PDTSPTW) in which the aim is to minimize lateness. One strategy requires the vehicle, when idle, to wait at the current customer location until it can service another customer without being early. Another strategy, propose to reposition the vehicle at a location different from that of the current customer based on a priori information on future requests. Vehicle relocation is also addressed by Bent and Van Hentenryck [Bent 2007]. Computational results indicate that these two strategies are very effective in maximizing the number of served customers, particularly for strong dynamic systems where the *dod* is high. Branke *et al.* [Branke 2005b] as well as Mitrovic-Minic and Laporte [Mitrović-Minić 2004b], have found that in the case of several vehicles, waiting allows the vehicles to remain at suitable locations during their tours, and improves the probability of being able to serve new customers.

1.4.2 Heuristics

Usually, simple strategies are related to specific conditions, which leads to lower empirical performance in other operational conditions. In these conditions, heuristics could be employed to improve the performance of the algorithms [Psaraftis 1988, Kilby 1998, Mitrović-Minić 2004a, Hvattum 2006, Branchini 2009]. These heuristics are essentially constructive and improvement procedures. Constructive heuristics create initial solutions, i.e., set of routes, from scratch. On the contrary, as support mechanisms, improvement procedures repair previously constructed solutions by performing simple reassignment moves. According to the planning horizon, these procedures reoptimize the vehicle route when new demand customer appears. The customers are inserted on the routes following the cheapest insertion (i.e. best position of the current routes) [Psaraftis 1988].

Mitrović-Minić *et al.* [Mitrović-Minić 2004a] apply the cheapest insertion procedure in order to determine the overall best insertions for the locations of a request before its insertion. The improvement procedure is based on Tabu Search (TS). It is applied after the reinsertion procedure and it runs while new requests are being received.

Kilby *et al.* in [Kilby 1998] use an insertion heuristic which inserts customers and is improved by using different heuristics as 2-Opt [Lin 1965], Or-

Opt [Or 1996], relocate, exchange, and cross [Breedam 1994].

Branchini *et al.* [Branchini 2009] propose an adaptive granular local search heuristic for the DVRP. This heuristic has a reduced neighborhood based on candidate-list strategy. Such neighborhood discards long tour segments which represent high costs arcs due to the small probability of these being part of good quality solutions and focuses on promising short tour segments. The heuristic adaptation assumes adjusting the size of the search space according to the short time that is available for the optimization method in different periods of the working day.

1.4.3 Metaheuristics

Metaheuristics design a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. They make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. They provide acceptable solutions in a reasonable time for solving hard and complex problems in science and engineering. Their main exploring characteristic is based on the balance between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. This balance between diversification and intensification is important, on one side to quickly identify regions in the search space with high quality solutions and on the other side not to waste too much time in regions of the search space which either are already explored or do not provide high quality solutions.

Metaheuristics can be grouped into two classes of algorithms; single-solution based metaheuristics and population based metaheuristics [Talbi 2009].

Single-solution based metaheuristics (S-metaheuristics) could be viewed as search trajectories over the search space of the problem. These trajectories are performed by iterative procedures that move from the current solution to another one in the search space. Furthermore, they incorporate techniques that enable the algorithm to escape from local minima. This class of metaheuristic algorithms includes -but is not restricted to- tabu search, simulated annealing, threshold accepting, variable neighborhood search, iterated local search, guided local search, GRASP, and so on.

On the other side, the population based metaheuristics (P-metaheuristics) start from an initial population of solutions and iteratively apply the generation of a new population and the replacement of the current one. They regroup algorithms such as evolutionary algorithms (genetic algorithms, evolu-

tion strategies, genetic programming, evolutionary programming, estimation of distribution algorithms, differential evolution, and co-evolutionary algorithms), swarm intelligence-based methods (e.g., ant colonies, particle swarm optimization), scatter search, bee colony, artificial immune systems, and so on.

We give an overview on the main existing works in literature on both classes of metaheuristics for dynamic vehicle routing problems in what follows:

1.4.3.1 Single-Solution Based Metaheuristics

Many works are related to trajectory-based metaheuristics for solving DVRP (see Table 1.2). Gendreau *et al.* in [Gendreau 1999] propose a parallel tabu search heuristic with an adaptive memory mechanism taken from Rochat and Taillard work [Rochat 1995]. The adaptive memory stores previously found elite solutions and used them to generate new starting points for the tabu search. This is achieved by combining routes taken from different solutions in memory. Any new solution produced by the tabu search is included in the memory if it is not filled yet. Otherwise, the new solution replaces the worst solution in memory, if it is better. The parallelization of the procedure was achieved at two different levels: (1) different tabu search threads run in parallel, each of them starting from a different initial solution; and (2) within each search thread, many tabu searches run independently on subproblems obtained through a decomposition procedure of the whole problem. For the parallel implementation a masterslave scheme was chosen to implement the procedure. The master process manages the adaptive memory and creates initial solutions for the slave processes that run the tabu search.

Gendreau *et al.* [Gendreau 2006] develop a tabu search heuristic for the Dynamic Vehicle Routing Problem with Pickup and Delivery (DPDVRP). The neighborhood structure is based on the ejection chain heuristic in which a request is taken from one route and moved to another route, thus forcing a request from that route to move to yet another route, and so on. The chain may be of any length and may be cyclic or not. The authors reuse the mechanism of adaptive memory.

Ichoua *et al.* in [Ichoua 2000] reuse the same algorithm with some enhancing related to the strategy for assigning customer requests to vehicles.

Mitrović -Minić *et al.* [Mitrović-Minić 2004a] deal with the Dynamic Pickup and Delivery Problem with Time Windows (DPDVRPTW) and applied the cheapest insertion procedure in order to determine the overall best insertions for the locations of a request before its insertion. The improvement procedure is based on Tabu Search (TS). It is applied after the reinsertion

procedure and it runs while new requests are being received.

Hanshar *et al.* implement a basic tabu search in [Hanshar 2007]. Two operators are employed as neighborhood structure procedures; inversion operator and λ-exchange operator [Osman 1993], each one was applied according to some probability. Furthermore, Montemanni [Montemanni 2005b] implement a GRASP (Greedy Randomized Adaptive Search Procedure) for dealing with DVRP. Basically, initial tours are generated by iteratively selecting the next customers to visit. The procedure is repeated until a complete solution is built.

Attanasio *et al.* present in [Attanasio 2004] use different neighbor for their parallel tabu search implemented for the Dynamic Dial-a-Ride Problem (DDARP). A neighbor solution is obtained by applying a simple operator that removes a customer from a route and reinserted it into another one.

1.4.3.2 Population-Based Metaheuristics

Several population-based metaheuristics have been proposed in the literature (see Table 1.2). An outline on the major works on these approaches is given in what follows:

1.4.3.3 Ant Colony Optimization

Ant System (AS) has been applied to tackle a large variety of Dynamic Vehicle Routing Problems [Gambardella 2003, Tian 2003, Montemanni 2005b, Rizzoli 2007, Jun 2008]. In this metaheuristic a colony of artificial ants is used to construct solutions guided by the pheromone trails and heuristic information.

Tian *et al.* [Tian 2003] present a hybrid Ant System to handle the dynamism by means of modifying the pheromone matrix in order to take advantage of the old information gathered during the previous search. They propose a new pheromone initialization for new demands, which works better than a re-start optimization. Furthermore, they use a simple strategy that consists in grouping new requests at every fixed interval-time before their introduction into the system. In addition, they make further improvements on vehicle routes with the local search 2-Opt heuristic.

Jun *et al.* [Jun 2008] address a hybrid multi-objective ant colony algorithm for solving DVRPTW. They consider two sub-objectives such as vehicle number and the time cost as an independent objective. In their Ant Colony Algorithm, an Evolutionary Algorithm (EA) is embedded to increase the pheromone update. They explain that EA participates to speed up the convergence of their algorithm.

Montemanni *et al.* [Montemanni 2005b] exploit some characteristics of the Ant Colony System optimization paradigm to smoothly save information about promising solutions when the optimization problem evolves because of the arrival of new orders. One of these characteristics is the pheromone conservation procedure. It contains information about the characteristics of good solutions for these problems. In particular, pairs of customers who have been visited in sequence in good solutions, will have high values in the corresponding entries of the pheromone matrix. In dynamic context, it is used to pass information about the properties of good solutions from a previously obtained results for the new changing environment since the two problems are potentially very similar. This operation avoids the restarting of the optimization at each time from scratch.

Based on the Montemanni's algorithm, Rizzoli *et al.* [Rizzoli 2007] discuss the applications of ACO on a number of real-world problems. They propose some results obtained by their algorithm on an on-line VRP for fuel distribution in the city of Lugano (Switzerland). Oliveira *et al.* [Oliveira 2008] propose an Ant Colony Algorithm for the DVRPTW with two different forms of attractiveness (time windows and distance) for building the vehicle routes. According to their experiments, more the degree of dynamism is higher, fewer customers will be served.

Chitty *et al.* [Chitty 2004] introduce a hybrid dynamic programming-ant colony optimization approach to solve bi-criterion Vehicle Routing Problems. The aim is to find routes that have both shortest overall travel time and the smallest variance in travel time. The hybrid approach uses the principles of dynamic programming to first solve simple problems using ACO (routing from each adjacent node to the end node), and then builds on this to eventually provide solutions (i.e. Pareto fronts) for routing between each node in the network and the destination node. However, the hybrid technique updates the pheromone concentrations only along the first edge visited by each ant. As a result it is shown to provide the overall solution in quicker time than an established bi-criterion ACO technique that is concerned only with routing between the start and destination nodes, allowing re-routing vehicle to dynamic changes within the road network.

1.4.3.4 Evolutionary Algorithms

Another popular P-metaheuristic is the Genetic Algorithm (GA) in which a population of individuals is modified by recombination and mutation operators.

Hanshar and Ombuki-Berman [Hanshar 2007] propose a GA that handles the optimization of the static VRP like-instances that correspond to the whole

dynamic problem. The GA is launched at each fixed duration and must run within an efficient amount of time. The fitness evaluation involves the vehicle routes obtained after the translation of the chromosome representation. It returns the total travel distance/cost of the routes. The Best-Cost Route Crossover (BCRC) is used as crossover operator, and the inversion operator as mutation.

Housroum *et al.* [Housroum 2006] deal with Dynamic Vehicle Routing Problem with Time Windows (DVRPTW). The authors propose an approach based on genetic algorithm. For their algorithm, they suggest PMX crossover, and different mutation operators such as Or-Opt, 1-Opt, or swap. They validate their approach on modified Solomon's benchmarks which have been proposed by Gendreau *et al.* [Gendreau 1999]. Zhao *et al.* in [Zhao 2008] use a similar GA than Housroum's algorithm [Housroum 2006] for solving the Dynamic Vehicle Routing Problem with time-dependent Travel Times (DVRPTT).

Alvarenga *et al.* propose in [Alvarenga 2005] a hybrid GA with Column Generation Heuristic for the DVRPTW. The authors propose a specific crossover, where at the first step, it makes a random choice of routes from each parent involved. After all feasible routes are inserted in the offspring. New routes are created if some customers remain after the insertion step. As mutation operator, a total of eight different operators are used in their work.

Branke *et al.* [Branke 2005b] propose a GA with different waiting strategy for vehicles for DCVRP. A two-point crossover is chosen and the mutation is done by adding to each value a normally distributed random value.
For their Evolutionary Hyper-Heuristics (EH-DVRP) [Garrido 2010], Garrido *et al.* propose a high-level algorithm which evolves and combines different types of low-level heuristics (constructive, perturbation, and noise heuristics) to solve the problem. Each individual of the population refers to a sequence of genes that correspond to a constructive and improvement heuristics which gradually inserts customers and repairs the set of routes created so far. These dedicated heuristics are applied to construct and improve partial states of the problem. The hyper-heuristic uses four operators to find new individuals: One recombination and three mutation-like operators. The recombination operator performs a one point crossover to generate two new offspring. For the mutation operators, the first one randomly selects and copies one of the heuristics to another position in the chromosome which allows to include new heuristics in a different steps of the algorithm. The second operator selects and replaces a gene by one single heuristic. The authors' idea is to give an alternative heuristic which may perform better in cooperation with existing ones. The last operator deletes a gene from the chromosome and discards some heuristics which cannot be useful to improve candidate solutions.

Wang *et al.* [Wang 2007] propose an EA for solving the DVRPTW. For the algorithm's reproduction phase, the authors use two-points crossover operator and a mutation operator that consists in changing the assignment of unserved customers to another vehicle. In order to enhance their algorithm, the authors hybridize their algorithm with a modified Dijkstras algorithm for finding the real-time shortest paths.

Jih *et al.* [Jih 1999] address a hybrid genetic algorithm for solving single-vehicle pickup and delivery problem with time windows and capacity constraints (DPDVRPTW). The approach enables dynamic programming to achieve real-time performance and genetic algorithms to approximate optimal solutions. The initial population is created by the dynamic programming instead of generating it randomly. The dynamic programming passes the unfinished routes to genetic algorithms to produce final solutions. The authors compare the performance of four crossover operators. These operators are: order crossover (OX), uniform order-based crossover (UOX), merge cross #1 (MX1) and merge cross #2 (MX2) [Blanton 1993]. In addition, they consider three mutation operators: (i) Two genes are selected randomly, and their positions are interchanged (swap operator). (ii) Randomly two cut sites are chosen, and the order of the sub-route specified by the genes is inverted (inverse operator). (iii) If the vehicle arrives at the i_{th} stop and violates the constraints, it disturbs the order of the genes within the first i_{th} sub-route (rearrangement operator).

Haghani *et al.* [Haghani 2005], deal with the pick-up and delivery vehicle routing problem with soft time windows in which are considered multiple vehicles with different capacities, real-time service requests, and real-time variations in travel times between demand nodes. This algorithm includes a vehicle merging operator in addition to the generic genetic operators, namely the crossover and the mutation operators.

Bosman *et al.* [Bosman 2006] introduce a probabilistic model to describe the behavior of the load announcements. This allows the routing to make informed anticipated moves to customers where loads are expected to arrive shortly. The approach outperforms the EA which only considers currently available loads. Only mutation is considered. In the mutation of an individual, two vehicles are chosen randomly (could be the same), and two customers from their respective routes are chosen randomly, and are swapped. This operator allows visits to customers to be exchanged between vehicles or to be re-ordered in the route of a single vehicle directly.

Van Hemertand and La Poutré [Van Hemert 2004] present an evolutionary algorithm that is able to provide solutions in real-time for the DVRP. The authors analyze the benefit of anticipatory vehicle moves within regions that have a high potential of generating loads (fruitful regions). Only mutation

is considered. Two vehicles, possibly the same one, are chosen randomly. In both vehicles two nodes are selected and are swapped.

Table 1.2: State-of-the-art metaheuristics for DVRP and its variants.

Metaheuristics		Authors	Problem	Operators
Single-Solution Based	Tabu Search	Hanshar *et al.* [Hanshar 2007]	DCVRP	λ-interchange and inversion
		Gendreau1 *et al.* [Gendreau 1999] Ichoua *et al.* [Ichoua 2000, Ichoua 2003]	DVRPTW	CROSS exchange
		Mitrović-Minić *et al.* [Mitrović-Minić 2004a]	DPDVRP	
	GRASP	Montemanni *et al.* [Montemanni 2005b]	DCVRP	Greedy insertion
Population Based	Ant Colony Optimization	Montemanni *et al.* [Montemanni 2005b] Tian *et al.* [Tian 2003] Chitty [Chitty 2004] Rizzoli *et al.* [Rizzoli 2007]	DCVRP	Greedy heuristic 2-opt heuristic Dynamic programming none
		Jun *et al.* [Jun 2008] Oliveira *et al.* [Oliveira 2008]	DVRPTW	Cooperation with EA none
	Evolutionary Algorithms	Hanshar *et al.* [Hanshar 2007]	DCVRP	BCRC crossover Mutation (inversion)
		Branke *et al.* [Branke 2005b]		Two-point crossover Mutation (replacement)
		Garrido2010 *et al.* [Garrido 2010]		One-point crossover 3 mutation operators (replacement,insertion,deletion)
		Van Hemertand and La Poutré [Van Hemert 2004]		mutation(CROSS exchange)
		Housroum *et al.* [Housroum 2006]	DVRPTW	PMX crossover 3 mutations (Or-Opt, 1-Opt, swap)
		Alvarenga *et al.* [Alvarenga 2005]		Specific crossover 8 mutations (insertion, exchange, ...)
		Wang *et al.* [Wang 2007]		Two-points crossover Mutation(Insertion)
		Jih *et al.* [Jih 1999]	DPDVRPTW	3 crossovers(OX, UOX MX1, MX2) 3 mutations (rearrangement, swap, inverse)
		Bosman *et al.* [Bosman 2006]		Crossover (none) CROSS exchange mutation
		Zhao *et al.* [Zhao 2008]	DVRPTT	PMX crossover 3 mutations (Or-Opt, 1-Opt, swap)

1.5 Dynamic Performance Measures

The aim of optimization in dynamic environments is not only to find an optimum within a given number of generations, but rather a perpetual adaptation to the changing environment. Besides the accuracy of an approximation at time t, the stability of the algorithm is also of interest as well as the recovery time to reach again a certain approximation quality. We report here some measures that could be used for evaluating the performance of an algorithm designed for the DVRP.

Weicker [Weicker 1999, Weicker 2002] proposes measures and considers that they have to be taken into account when analyzing and comparing algorithms for dynamic problems. Three features have been proposed for Evolutionary Algorithm in order to describe the goodness of a dynamic adaptation process: accuracy, stability, and ε-reactivity. However these measures could be applied to any algorithm A used to solve a dynamic problem.

The accuracy should measure the closeness of the current best found solution to the actual best solution. The *accuracy* at time t for a fitness function F and optimization algorithm A is defined as:

$$accuracy_{F,A}^{(t)} = \frac{F(best_A^{(t)}) - Min_F^{(t)}}{Max_F^{(t)} - Min_F^{(t)}} \tag{1.10}$$

where $best_t^A$ is the best solution found by an evolutionary algorithm (EA) in the population at time t. The maximum and minimum fitness values in the search space are represented by Max_F^t and Min_F^t.

Thus, for the examined algorithm, this ratio quantifies the loss of cost-efficiency stemming from the lack of full information. The optimization accuracy ranges between 0 and 1, where accuracy equal to 1 is the ideal value.

As a second goal, stability is an important issue in optimization. In the context of dynamic optimization, an adaptive algorithm is called stable if changes in the environment do not affect the optimization accuracy severely. Even in the case of drastic changes an algorithm should be able to limit the respective fitness drop. The stability at time t is defined as

$$stability_{F,A}^{(t)} = max\{0, accuracy_{F,A}^{(t)} - accuracy_{F,A}^{(t-1)}\} \tag{1.11}$$

and ranges between 0 and 1. A value closes to 0 implies a high stability.

Finally, another aspect to be considered is the ability of the algorithm to react quickly to changes. This is measured by the ε-reactivity, which ranges

in $[1, maxgen]$ (a smaller value implies a higher reactivity):

$$reactivity^{(t)}_{F,A,\varepsilon} = \begin{array}{l} \min\left\{t' - t \mid t < t' \leq maxgen, t \in \mathbb{N}, \frac{accuracy^{(t')}_{F,A}}{accuracy^{(t)}_{F,A}} \geq (1-\varepsilon)\right\} \\ \cup\{(maxgen - t)\} \end{array} \tag{1.12}$$

Since it is difficult to know which is the best achievable value in a dynamic problem, Weicker points out that an average of several generations should be used instead.

Weicker summarizes different proposals for accuracy in problems where the global optimum is unknown, offering the following options to be used instead Max^t_F:

$$CurrentBest^{(t)}_{F,A} = \max\{F(\omega)|\omega \in P^{(t)}_A\} \tag{1.13}$$

$$CurrentBestOffline^{(t)}_{F,A} = \max_{1\leq t'\leq t}\{CurrentBest^{(t')}_{F,A}\} \tag{1.14}$$

$$CurrentAverage^{(t)}_{F,A} = \frac{1}{|P^{(t)}_A|}\sum_{\omega\in P^{(t)}_A}(F(\omega)) \tag{1.15}$$

where $P^{(t)}_A$ is the population of the algorithm at time t. Another approach to measure the accuracy without actually knowing the best possible fitness is based on the assumption that the best fitness value will not change much within a small number of generations. For that, a window is introduced inside the time span of the problem and the accuracy window $Acc_{F,EA,W}$ is measured within this window of length W.

$$windowAcc^{(t)}_{F,EA,W} = \max\left\{\frac{F(\omega) - windowWorst}{windowBest - windowWorst}|\omega \in P^{(t)}_{EA}\right\} \tag{1.16}$$

$$windowBest = \max\{F(\omega)|\omega \in P^{(t')}_A, (t-W) \leq t' \leq t\} \tag{1.17}$$

$$windowWorst = \min\{F(\omega)|\omega \in P^{((t')}_A, (t-W) \leq t' \leq t\} \tag{1.18}$$

Alternatively to the fitness based performance measures, *genotype* or *phenotype* based measures can also be used to give an approximated value of the optimization accuracy. Weicker notes that these measures require full global knowledge of the position of the current optimum and gives two variants. The first proposal uses the minimal distance of the individuals in the population to the current optimum ω^* in the search space and is given as follows:

$$bestDist^{(t)}_{F,A} = \max\left\{\frac{maxdist - d(\omega^*, \omega)}{maxdist}|\omega \in P^{(t)}_{EA}\right\} \tag{1.19}$$

where *maxdist* is the maximum distance between two solutions in the search space Ω.

The second proposal has been introduced by Salomon and Eggenberger [Salomon 1998] and used the distance of the mass center or centroid ω_{center} of the population to the current optimum ω_* and is obtained by:

$$centerDist^{(t)}_{F,EA} = \frac{maxdist - d(\omega^*, \omega_{center})}{maxdist} \tag{1.20}$$

Weicker notes that the first proposal seems to be straightforward to assess the approximation quality, the second performance measure is more difficult to interpret. It requires that the population as a whole describes very closely the region of the optimum.

1.6 Benchmarks

Benchmarks can be defined as standard test problems designed to serve as bases for algorithm assessment and comparison. The main reason for testing an algorithm on such problems is to compare the obtained results with those obtained by other algorithms and hence prove the superiority or not of the tested algorithm. Instances can be constructed from two types of data: randomly generated data, and real-life data. On the one side, random data is easy to obtain and enables to deduce conclusions about the algorithm performance. On the other side, instance from the second source would be a particular instance from the real-world. Thus, this kind of instance could be used to attest that the algorithm has the capability to solve real-world problems.

Different sets of benchmarks have been proposed in the literature for DVRP. Most of them are derived from some very popular static VRP benchmark datasets.

Kilby *et al.* [Kilby 1998] propose a set of instances, namely Taillard [Taillard 1993] (13 instances), Christophides and Beasley [Christofides 1984] (7 instances) and Fisher *et al.* [Fisher 1995] (2 instances). Their size ranges from 50 to 199 customers.These instances were organized and extended by Montemanni *et al.* [Montemanni 2005a]. He organized the instances into two groups, pickup and delivery, and gave to each instance an available time which signifies when the order was placed into the system and a duration, which represents the least amount of time a vehicle waits at a customer.

For the conventional instances of VRP variants, Alvarenga *et al.* [Alvarenga 2005] modify Solomon's instances to get DVRP with Time Windows (DVRPTW) instances. Gendreau *et al.* [Gendreau 2006]

propose their own instances for the DVRP with Pickup and Delivery, where, Attanasio *et al.* [Attanasio 2004] introduce a benchmark data set for the Dynamic Dial-a-Ride Problem (DDARP).

Nevertheless, in dynamic optimization, a test problem is characterized by a particular dynamic scenario, which refers to the sequence of events or environmental changes in the problem. Moreover, one could be faced with the difficulty of finding a real life situation which matches the instance that represents a typical dynamic desired scenario(s). We need a general-purpose benchmark generator. This generator must use different functions with tunable parameters to produce wide varieties of scenarios. Thus, the user can accurately pre-determine particular set of events for the test instances. Furthermore, generating dynamic instances must take into account mainly two aspects; data which are related to the structure of the instance(topology of customer, distribution in the service area,...) and data that is time dependent (arrival of customer demands, travel time, ...). Test instances should also be able to cover wide ranges of environmental changes.

Therefore, we propose a DVRP Instance Generator(DVRPGen) available online[1], and that takes into account the features above. It provides dynamic customized instances named *K-series*. Our generator provides instances according to different dynamic scenarios such as dynamic request, dynamic travel times, vehicle breakdown, etc. We can associate a stochastic process to each variable of the instances (requests or quantity of demands, travel times, etc.). Most of VRP variants are represented including time windows, multi-depot, and pickup and delivery problems. Each user has the possibility to generate dynamic customized instances according to different spatial topologies of customers in service area (cluster, uniform, mix), as well as time distributions (i.e. arrival of customer demands can follow uniform, poisson, or normal distributions). Furthermore, the generator offers a view of the instance in 2D dimensional space.

In order to obtain dynamic instances, some characteristics have been added to the instances:

- Length of the working day $[0, T]$.
- Occurrence time of each request. It contains, for each request, the moment of the working day, when the order becomes known to the planner.
- Duration service of each request. It represents, for each order, the time required to serve the corresponding customer.

[1] `http://dolphin.lille.inria.fr/Research/Benchmark`, `http://neo.lcc.uma.es/dynamic`

- Number of vehicles. It contains the dimension in number of vehicles of the fleet available for serving the customers.

We report in Table 1.3 some set of benchmarks that have been used or proposed in the literature and which are available online[2].

[2] http://www.fernuni-hagen.de/WINF/inhalte/benchmark_data.htm

Table 1.3: Benchmark data sets for dynamic vehicle routing problems.

Author(s)	Problems	Instance Description	Problem size
Montemanni *et al.* [Montemanni 2005b]	DCVRP	– 44 instances derived from problem instances of Taillard, – Christofides, Beasley and Fisher.	50-199
Branke *et al.* [Branke 2005b]		– 7 Instances derived from the OR library of Beasly.	up to 1000 requests
Gendreau *et al.* [Gendreau 2006]	DPDVRPTW	– 15 problem instances. – Fixed vehicle fleet size (10, 20).	up to 200
Fabri and Recht. [Fabri 2006]		– Derived from static Li & Lim PDPTW.	up to 500 requests for 1000 nodes
Pankratz [Pankratz 2005]			50 requests (100 nodes)
Mitrovic-Minic *et al.* [Mitrović-Minić 2004a]		– 90 problem instances (30 instances per size).	100-1000
Attanasio *et al.* [Attanasio 2004]	DDARP	– Using the problem instances of Cordeau *et al.* [Cordeau 2003]. – 26 problem instances (6 of them are real-life cases).	24-144
Chen *et al.* [Chen 2006b]	DVRPTWTT	– 56 instances derived from the VRPTW Solomon instances. – Include the time-dependent travel time.	100 requests
Khouadjia 1	DCVRP, DVRPTW, DPDVRP, DVRPTT, DSVRP,DSVRPTW, DSPDVRP, DVSVRPTT	– Includes different dynamic scenarios (dynamic customers, vehicle broken down, roads blocked-off, etc.). – Several variants are represented (time windows, pickup & delivery, multi-depot, etc.). – Customized instances according to the customer distribution (cluster, uniform, mix between both).	unbounded

1.7 Conclusion

An overview of different aspect of Dynamic Vehicle Routing Problem (DVRP) has been presented in this chapter. This problem is important both in research and industrial domains due to its many real world applications.

The state-of-the-art presented in this chapter covers a description of the problem, its representation as well as the existing solving methods. Dynamic performance measures are presented in order to quantify the adaptation of the algorithms throughout the optimization process. The online available benchmarks are reported and a benchmark generator has been presented. It provides instances that could be generated according to different dynamic scenarios (dynamic requests, vehicle broken down, etc.) as well as different VRP variants (time window, multi-depot, pickup & delivery, etc.). These instances will be used to guide the experiment protocol.

From the proposed survey, we can conclude that a well-designed approach should not be restricted to a given class of methods (strategies, heuristics, metaheuristics), but has to take into account different features and mechanisms that have been employed on these different techniques. Furthermore, measuring the adaptability of an algorithm over the optimization process is a major stake when we face dynamic optimization problems. Robustness of solutions should be taken into account also during the designing step.

The next chapters will handle these open issues, we will expose in this thesis our contribution in solving the Dynamic Capacitated Vehicle Routing Problem (DCVRP). In this variant of the problem some customers are unknown when the optimization process begin, i.e. their orders and positions will be known only after the vehicles are already in route. Thus, the goal is to serve the set of customers and minimize the traveled distance by the vehicle fleet.
The proposed approaches are designed by taking advantage of the best proposed approaches until day.

Chapter 2

Single-Solution Based Metaheuristics for Solving Dynamic Vehicle Routing Problem

Contents

2.1 Introduction

In dynamic optimization, it appears that any self-contained research work has to deal with three issues: adaptation to environmental changes, benchmarking, and performance measures.
The adaptation of environmental changes refers to the ability of the algorithm to continue the search for new optima if the environment shifts. Hence, the optimization algorithm has to track a moving optimum through time as closely as possible, rather than just find a single good solution. Thus, once the algorithm starts to converge around some optimal or near optimal solution, it should not lose its ability to continue the search of new optima if environmental change occurs.
A class of approaches that seems to be interesting for the dynamic optimization problems is the Single-solution based metaheuristics (S-metaheuristics).

They could be viewed as walks through neighborhoods or search trajectories over the search space of the problem at hand. These trajectories are performed by iterative procedures that move from the current solution to another one in the search space.
One of the recent approaches in the filed of S-metaheuristic is Variable Neighborhood Search (VNS) [Hansen 1999]. It consists in adaptively changing the neighborhood in order to get different local optima and to escape from local optima. The process of changing neighborhoods corresponds to a diversification of the search. In particular, the choice of neighborhoods of increasing cardinality yields a progressive diversification. Moreover, a solution that is locally optimal on the search landscape with respect to a neighborhood is probably not locally optimal with respect to another neighborhood. The global optima will be one of the local optima of a given neighborhood. Different neighborhoods generate different landscapes, which is known as "One Operator, One Landscape" concept [Jones 1995]. The core idea is that the neighborhood structure determines the topological properties of the search landscape, i.e., each neighborhood defines one landscape. The properties of a landscape are in general different from those of other landscapes, therefore a search strategy performs differently on them. This specificity provides to VNS a serious ability and reactivity to track the shifting optimum in dynamic optimization problems.
In this chapter we propose to face the Dynamic Capacitated Vehicle Routing Problem (DCVRP) by using Variable Neighborhood based approach. Here, the objective function consists in minimizing the distance traveled by the vehicles which serve the customers (see equation 1.2). We believe that the characteristic of changing the neighborhood structure offers a powerful mechanism of adaptivity to the environmental changing. To quantify this adaptability, we measure several indicators based on Weicker's dynamic performance measures [Weicker 2002] on algorithm along the optimization process.
The remainder of this chapter is organized as follows: The fundamental bases of S-metaheuristics are presented in the Section 2.2. Section 2.4 introduces, our VNS-DVRP in an incremental manner, with the representation, the definition of the neighborhood structure, the incremental evaluation function, the determination of the initial solution, and the dedicated algorithm. Section 2.5 describes and explains how to simulate and to solve the dynamic problem. In Section 2.6, we discuss our experimental methodology, and provide an experimental analysis. Finally, we conclude with a summary of the main contributions reported in this chapter.

2.2 Single-Solution Based Metaheuristics

Single Solution Based Metaheuristics (S-metaheuristics) iteratively apply the generation and replacement procedures from the current single solution. In the generation phase, a set of candidate solutions is generated from the current solution s. This set $C(s)$ is generally obtained by local transformations of the solution. In the replacement phase, a selection is performed from the candidate solution set $C(s)$ to replace the current solution; that is, a solution $s' \in C(s)$ is selected to be the new solution. This process iterates until a given stopping criteria (Figure 2.1). The generation and the replacement phases may be memory-less. In this case, the two procedures are based only on the current solution. Otherwise, some history of the search stored in a memory can be used in the generation of the candidate list of solutions and the selection of the new solution. Popular examples of such S-metaheuristics are simulated annealing [Kirkpatrick 1983, Černỳ 1985], tabu search [Glover 1990], and variable neighborhood search [Hansen 1999]. Algorithm 1 illustrates the high-level template of S-metaheuristics. The common search concepts for all S-metaheuristics are the definition of the *neighborhood* structure and the determination of the *initial solution* [Talbi 2009].

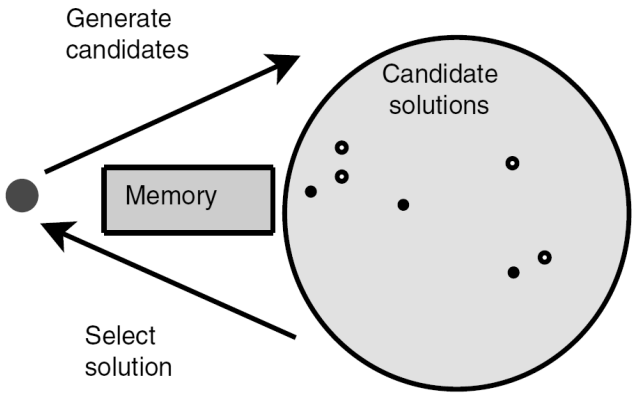


Figure 2.1: Main principles of single-solution based metaheuristics.

2.2.1 Neighborhood

The definition of the neighborhood is a required common step for the design of any S-metaheuristic. The neighborhood structure plays a crucial role in the performance of an S-metaheuristic. If the neighborhood structure is not adequate to the problem, any S-metaheuristic will fail to solve the problem.

Algorithm 1 High-level template of S-metaheuristics.

Input: Initial solution s_0 .
$t = 0$;
repeat
/ Generate candidate solutions (partial or complete neighborhood) from s_t /
Generate($C(s_t)$) ;
/ Select a solution from $C(s)$ to replace the current solution s_t /
s_{t+1} = Select($C(s_t)$) ;
$t = t + 1$;
until Stopping criteria satisfied
Output: Best solution found.

Definition 2.2.1 *A neighborhood function $\mathcal{N}$ is a mapping $\mathcal{N} : S \rightarrow 2^S$ that assigns to each solution s of S a set of solutions $\mathcal{N}(s) \subset S$. A solution s' in the neighborhood of s ($s' \in \mathcal{N}(s)$) is called a neighbor of s.*

A neighbor is generated by the application of a move operator m that performs a small perturbation to the solution s. The main property that must characterize a neighborhood is locality. Locality is the effect on the solution when performing the move (perturbation) in the representation. The neighborhood is said to have a strong locality, if when small changes are made in the representation, the solution is affected slightly. Hence, a S-metaheuristic will perform a meaningful search in the landscape of the problem.

Weak locality is characterized by a large effect on the solution when a small change is made in the representation. In the extreme case of weak locality, the search will converge toward a random search in the search space. The structure of the neighborhood depends on the target optimization problem. It has been first defined in continuous optimization.

Definition 2.2.2 *In a discrete optimization problem, the neighborhood $\mathcal{N}(s)$ of a solution s is represented by the set $\{s'/d(s', s) \leq \varepsilon\}$, where d represents a given distance that is related to the move operator.*

The neighborhood definition depends strongly on the representation associated with the problem at hand. Some usual neighborhoods are associated with traditional encodings. The natural neighborhood for binary representations is based on the Hamming distance. In general, a distance equal to 1 is used. Then, the neighborhood of a solution s consists in flipping one bit of the solution. For a binary vector of size n, the size of the neighborhood will be n. The Hamming neighborhood for binary encodings may be extended to any discrete vector representation using a given alphabet . Indeed, the substitution can be generalized by replacing the discrete value of a vector element

by any other character of the alphabet. If the cardinality of the alphabet is k, the size of the neighborhood will be $(k-1) \times n$ for a discrete vector of size n. For permutation-based representations, a usual neighborhood is based on the swap operator that consists in exchanging (or swapping) the location of two elements s_i and s_j of the permutation. For a permutation of size n, the size of this neighborhood is $n(n-1)/2$. This operator may also be applied to any linear representation. Figure 2.3 shows the neighborhood associated with a combinatorial optimization problem using a permutation encoding. The distance is based on the swap move operator. Once the concept of neighborhood has been defined, the local optimality property of a solution may be given.

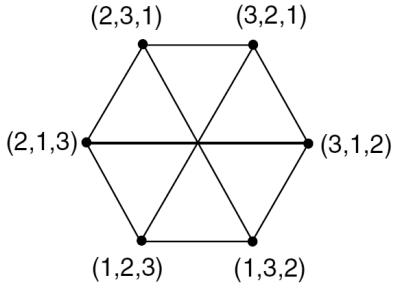


Figure 2.2: An example of neighborhood for a permutation problem of size 3. For instance, the neighbors of the solution (2, 3, 1) are: (3, 2, 1), (2, 1, 3), and (1, 3, 2).

Definition 2.2.3 ***Local optimum.*** *Relatively to a given neighboring function $\mathcal{N}$, a solution $s \in S$ is a local optimum if it has a better quality than all its neighbors; that is, $f(s) \leq f(s')$ for all $s' \in \mathcal{N}(s)$. For the same optimization problem, a local optimum for a neighborhood $\mathcal{N}_1$ may not be a local optimum for a different neighborhood $\mathcal{N}_2$.*

2.2.2 Initial Solution

Two main strategies are commonly used to generate the initial solution: a random and a greedy approach. There is always a trade-off between the use of random and greedy initial solutions in terms of the quality of solutions and the computational time. The best answer to this trade-off will depend mainly on the efficiency and effectiveness of the random and greedy algorithms at hand, and the S-metaheuristic properties. For instance, the larger is the neighborhood, the less is the sensitivity of the initial solution to the performance of the S-metaheuristics. Generating a random initial solution is a quick operation,

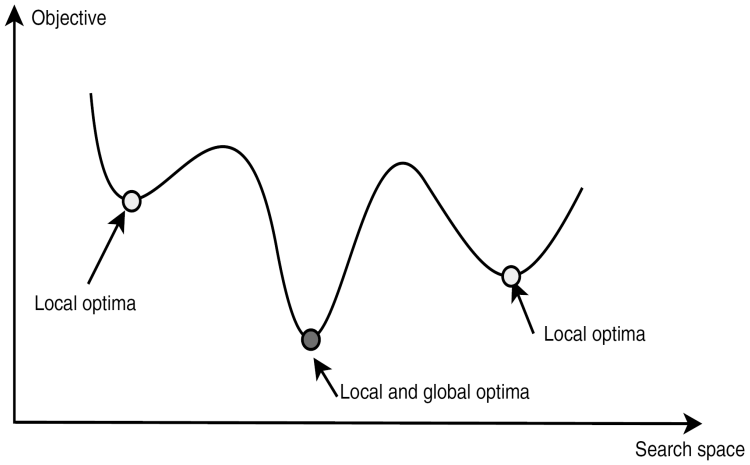


Figure 2.3: Local optimum and global optimum in a search space. A problem may have many global optimal solutions.

but the metaheuristic may take much larger number of iterations to converge. To speed up the search, a greedy heuristic may be used. Indeed, in most of the cases, greedy algorithms have a reduced polynomial-time complexity. Using greedy heuristics often leads to better quality local optima. Hence, the S-metaheuristic will require, in general, less iterations to converge toward a local optimum. Some approximation greedy algorithms may also be used to obtain a bound guarantee for the final solution. However, it does not mean that using better solutions as initial solutions will always lead to better local optima.

2.3 S-Metaheuristics for DVRP: Literature Review

Tabu Search metaheuristic is one of the most popular S-metaheuristic in solving DVRP. Gendreau *et al.* in [Gendreau 1999] propose a parallel implementation of Rochat and Taillard approach for solving DVRP [Rochat 1995]. It is based on adaptive memory which stores elite solutions discovered in previous searches and combines them to produce new initial solutions. The neighborhood is generated with Cross exchange operator. It is basically a chain exchange procedure, where two segments of variable length are taken from different routes and moved from one route to another.

Mitrović -Minić *et al.* [Mitrović-Minić 2004a] deal with the Dynamic Pickup and Delivery Problem with Time Windows (DPDVRPTW) and chose the cheapest insertion heuristic as a constructive method. The cheapest

insertion procedure is applied to new requests accumulated over a given time period and allows to determine the overall best insertions for the locations of a request before its insertion. Before insertion, these requests are sorted in increasing order of slack time. The slack time of a request is equal to the difference between the total time available to serve the request and the direct travel time between its pickup and delivery locations. The Tabu search is applied after the constructive procedure with neighborhoods defined by means of ejection chains.

Hanshar *et al.* [Hanshar 2007] introduce two operators to generate the neighborhood structure of their Tabu Search; inversion operator and λ-exchange operator [Osman 1993], each one was applied according to some probability.

Besides, Attanasio *et al.* [Attanasio 2004] present a parallel tabu search for the Dynamic Dial-a-Ride problem (DDARP). An initial solution is obtained by randomly assigning requests to routes while satisfying the constraints related to the problem. The neighborhood of a solution s is generated by the relocate operator which makes up all solutions reachable from s by simply moving a customer visit from one route to another.

A Greedy Randomized Adaptive Search Procedure (GRASP) [Montemanni 2005b] have been also implemented for dealing with DVRP. Initial tours are generated by iteratively selecting the next customers to visit at random among the feasible ones. Once a complete solution is available, it is tentatively improved using a local search procedure.

2.4 Variable Neighborhood Search for Dynamic Vehicle Routing Problem

In this section, we present our VNS-DVRP approach. The adaptation of the different component for DVRP is described and examined.

2.4.1 Variable Neighborhood Search (VNS)

Variable neighborhood search has been recently proposed by P. Hansen and N. Mladenović [Hansen 1999]. The basic idea of VNS is to successively explore a set of predefined neighborhoods to provide a better solution. It explores either at random or systematically a set of neighborhoods to get different local optima and to escape from local optima. VNS exploits the fact that using various neighborhoods in local search may generate different local optima and that the global optima is a local optima for a given neighborhood (Figure 2.4). Indeed, different neighborhoods generate different landscapes [Jones 1995].

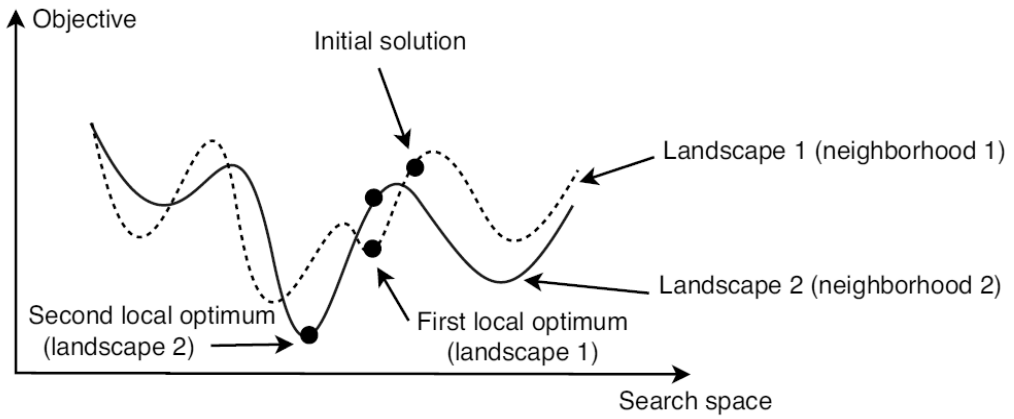


Figure 2.4: Variable neighborhood search using two neighborhoods. The first local optimum is obtained according to the neighborhood 1. According to the neighborhood 2, the second local optimum is obtained from the first local optimum.

VNS is a stochastic algorithm in which, first, a set of neighborhood structures $\mathcal{N}_k(k = 1, ..., n)$ are defined. Then, each iteration of the algorithm is composed of three steps: shaking, local search and move. At each iteration, an initial solution is shaken from the current neighborhood $\mathcal{N}_k$. For instance, a solution s' is generated randomly in the current neighborhood $\mathcal{N}_k(s)$. A local search procedure is applied to the solution s' to generate the solution s''. The current solution is replaced by the new local optima s'' if and only if a better solution has been found (i.e., $f(s'') < f(s)$). The same search procedure is restarted from the solution s'' in the first neighborhood $\mathcal{N}_1$. If no better solution is found (i.e., $f(s'') \geq f(s)$), the algorithm moves to the next neighborhood $\mathcal{N}_{k+1}$, randomly generates a new solution in this neighborhood, and attempts to improve it. Let us notice that cycling is possible (i.e., $s'' = s$) (Figure 2.5). Algorithm 2 presents the template of the basic VNS algorithm.

The design of the VNS algorithm is mainly related to the selection of neighborhoods for the shaking phase. Usually, nested neighborhoods are used, where each neighborhood $\mathcal{N}_k(x)$ contains the previous one $\mathcal{N}_{k1}(x)$: A compromise must be found between intensification of the search and its diversification through the distribution of work between the local search phase and the shaking phase. An increased work in the local search phase will generate better local optima (more intensification), whereas an increased work in the shaking phase will lead to potentially better regions of the search space (more diversification).

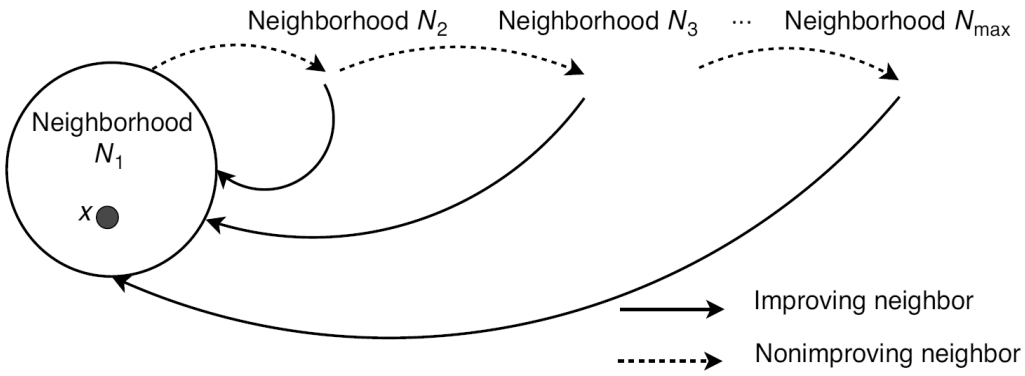


Figure 2.5: The principle of the variable neighborhood algorithm.

Algorithm 2 Template of the basic variable neighborhood search algorithm.

Input: a set of neighborhood structures $\mathcal{N}_k$ for k = 1, . . . , k_{max} for shaking.
$s = s_0$; / Generate the initial solution /
repeat
  $k = 1$;
  repeat
    Shaking: pick a random solution s' from the k^{th} neighborhood $\mathcal{N}_k(s)$ of s ;
    $s'' = localsearch(s)$;
    if $f(s'') < f(s)$ **then**
      $s = s''$;
      Continue to search with $\mathcal{N}_1$; $k = 1$;
    else
      $k = k + 1$;
    end if
  until $k = k_{max}$
until Until Stopping criteria
Output: Best found solution.

2.4.2 DVRP Solution's Representation

The representation design consists in finding a suitable mapping between problem and algorithmic feasible solution. Since the customers are unknown beforehand and arrive through the time, we propose a dedicated encoding of solution to dynamic vehicle routing problems. The representation is dynamic in the sense that it has a variable length and extends while customer requests appear. Our representation allows the insertion of dynamic customers in the already planned routes. Given that the problem is dynamic and customer requests arrive along time, it is necessary to have some information about

the state of each customer (visited/ unvisited) and its time of service. Thus, routes distinguish between strictly fixed assignments and tentative proposals. We notice that, according to the problem description, uncommitted requests always correspond to the last part of the route. On the other side, for each vehicle we need some information as its current position in the service region, its remaining capacity, its traveled distance, and its status (committed/not committed). We propose a discrete representation which expresses the route of m vehicles over the n customers to serve. The solution consists in a set of partial routes, where there exists pending customers that have been newly added to the day's schedule, but not yet assigned to any vehicle and committed customers that have already been visited by a given vehicle. A solution to the problem is represented by a set of routes $S = \{R_1...R_p...R_q...R_m\}$, where R_k is the set of customers serviced by the vehicle V_k. The representation of each route R_k is a permutation of n customers as follows:

$$R_k : (c_0, c_1, c_2, ..., c_i, ..., c_n, c_{n+1}) \tag{2.1}$$

For each customer c_i, we assign the following information:

- (x_i, y_i): coordinates of the customer c_i.
- s_i: boolean variable which indicates if the customer c_i has been already served or not.
- t_i: processing time of the customer c_i (time in which the customer is served).

Furthermore, for each route R_k served by the vehicle v_k, we keep some information:

- (x_j, y_j): coordinates of the vehicle v_k.
- cap_k: remaining capacity of the vehicle v_k.
- $dist_k$: distance traveled by the vehicle v_k.
- $commit_k$: boolean variable which indicates if the vehicle v_k has been committed or not.

Therefore, this representation provides more flexibility and abstraction to several heuristics (see Section 2.4.3) since they can understand partial states of the problem using this information, no matter how complicated the current scenario may seem. It offers and contributes in a clear manner to design dynamic vehicle routing problem solutions.

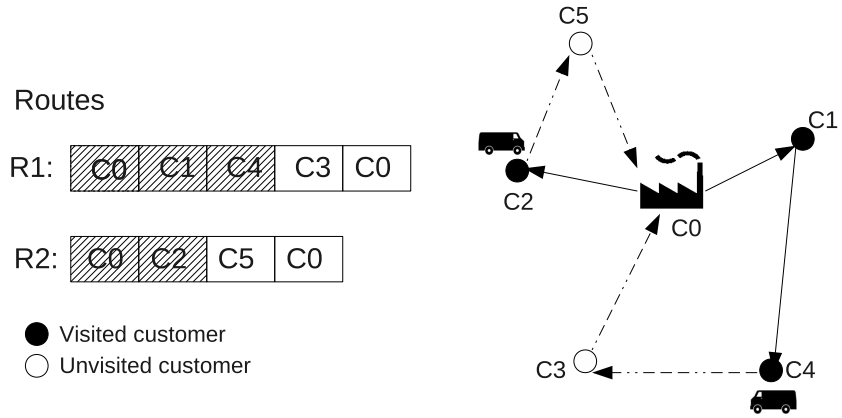


Figure 2.6: The solution's representation of the partial state of the problem. The cross-hatched customers represent the visited part, those without hatching are the unvisited part.

2.4.3 Neighborhood

In order to adapt VNS for a particular problem, it is necessary to define the set of neighborhood structures and to establish the local search procedure that will be applied to the solutions. Both our neighborhoods and the local search are related to move operators specific to the VRP. We propose four different neighborhoods $\mathcal{N}_k(s)$ for our VNS algorithm. The neighborhoods are defined as follows:

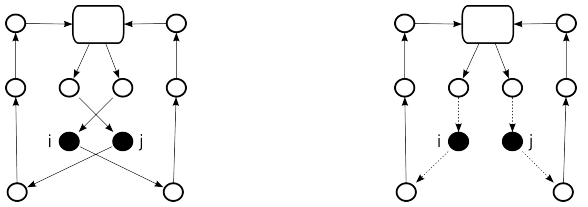
1. $\mathcal{N}_1(s)$ is the set of solutions which results of exchange operator. It consists in swapping any two customers in the solution s. The arcs $(i-1,i)$, $(i,i+1)$, $(j-1,j)$, and $(j,j+1)$ are replaced by $(j-1,i)$, $(i,j+1)$, $(i-1,j)$ and $(j,i+1)$. The size of this neighborhood is $n(n-1)/2$ where n represents the number of customers (see Figure 2.7).

2. $\mathcal{N}_2(s)$ is the set of solutions which results of λ-Interchange [Osman 1993] operator with $\lambda = 1$. The λ-Interchange operator we use is based on the interchange of all the possible combinations for up to λ customers between sets of routes. Hence, this method results in customers either being shifted from one route to another, or being exchanged between routes. The mechanism can be described as follows. New neighboring solutions can be obtained by applying λ-Interchange between a pair of routes R_p and R_q by replacing each subset of customers $S_1 \subseteq R_p$ of size $|S1| \leq \lambda$ with any other one $S2 \subseteq R_q$ of size $|S2| \leq \lambda$. This way, we get two new routes $R'_p = (R_p - S_1) \cup S_2$ and $R'_q = (R_q - S_2) \cup S_1$, which are parts of the new solution $S' = \{R_1 \dots R'_p \dots R'_q \dots R_m\}$. Then, λ-Interchange neighborhood selects two subsets of customers (with car-

dinality less than or equal to λ) from two different routes and exchanges them considering all possible insertion positions of both routes, resulting in a neighboorhood size $n^{4\lambda}$. If it is the required that the nodes be inserted in the position of the removed nodes, the size reduces to $n^{2\lambda}$. Since the size of λ-Interchange neighboorhood is relatively large even for small values of λ, in the literature λ rarely exceeds 2. For $\lambda = 1$, it results in customers either being shifted from one route to another for the $(1, 0)$ or $(0, 1)$ moves, or being exchanged between both routes for the $(1, 1)$ move (see Figure 2.8). The insertion of a customer is done using the cheapest cost insertion (*i.e.* the position that minimizes the cost of the insertion) [Funke 2005, Gendreau 2010].

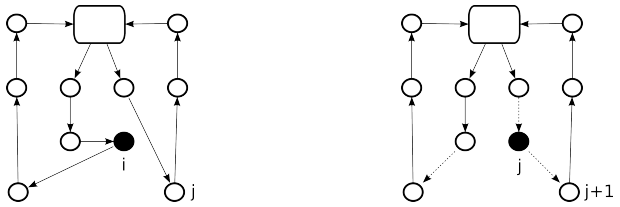
3. $\mathcal{N}_3(s)$ is the set of solutions which results of applying 2-Opt [Lin 1965] to any subroute of the solution s. The 2-Opt operator reverses a subroute of a given route R_k by selecting two arcs (i_1, i_2) and (j_1, j_2) and substituting them by (i_1, j_1) and (i_2, j_2) (see example in Figure 2.9). The already traveled segment of the tour is left untouched. The heuristic is operated on each route and only for the segments with unserved customers. The neighborhood for the 2-Opt operator is represented by all the permutations obtained by removing two edges. The size of the neighborhood for the 2-opt operator is $(n(n1)/2)n$; all pairs of edges are concerned except the adjacent pairs.

4. $\mathcal{N}_4(s)$ is the set of solutions which results of using 2-Opt* [Potvin 1995] in any two subroutes of the solution s. The 2-Opt* operator selects two arcs $(i_1, i_2) \in R_\alpha$ and $(j_1, j_2) \in R_\beta$ and constructs two new arcs so that (i_1, j_2) and (j_1, i_2) (see an example in Figure 2.10). The size of the 2-Opt* neighborhood is quadratic.



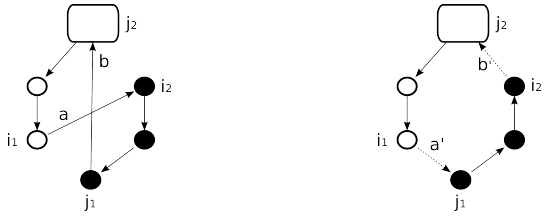
(a) Before (b) After

Figure 2.7: Example of the exchange operator. Two customers i, j from different routes are simultaneously placed into the other routes.



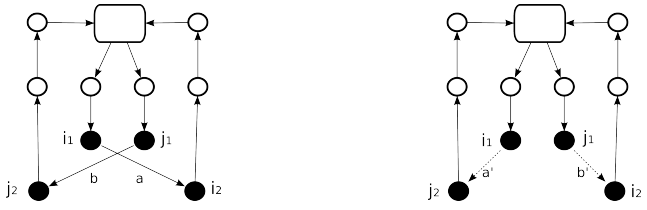
(a) Before (b) After

Figure 2.8: Example of the λ-interchange operator ($\lambda = 1$). In this example, it plays the role of relocate operator. The edges $(i-1, i)$, $(i, i+1)$ and $(j-1, j)$ are replaced by $(i-1, i+1)$, $(j-1, i)$ and $(i, j+1)$, i.e., customer i from the origin route is placed into the destination route.



(a) Before (b) After

Figure 2.9: Example of the 2-Opt operator applied to arcs a and b in one single route.



(a) Before (b) After

Figure 2.10: Example of the 2-Opt* operator applied to arcs a and b belonging to two different routes.

Given that the constraints are not enforced at this stage; this means that a solution s' picked up from the neighborhood does not need to comply with the capacity and depot working day restrictions. A repair procedure makes this new solution feasible before its evaluation. This repair procedure is necessary since the neighborhood operators can generate unfeasible solutions.

For that, some customers are shifted from the unfeasible tour in order to fill the constraints related to the depot time window and the capacity of the vehicles. The procedure consists in choosing the customer(s) c_i which minimizes the cost of its insertion into another tour comparatively to its extraction. The cost of the extraction is $CE = d_{(i-1),(i+1)} - (d_{(i-1),i} + d_{i,(i+1)})$, where the cost of the insertion is $CI = (d_{j,i} + d_{i,(j+1)}) - d_{j,(j+1)}$. Then, the customer c_i is inserted into another tour by following the Chepeast Insertion Heuristic (CIH). This heuristic is illustrated in Figure 2.11 and described in the following steps [Kindervater 1989]:
(i) Start with a tour consisting of a given vertex i and a self-loop.
(ii) Find a vertex not on the tour which can be inserted between two neighboring vertices i and j such that the distance $d_{ik} + d_{kj} - d_{ij}$ is minimal.
(iii) Insert this vertex between two neighboring vertices on the tour. If the tour still incomplete (some vertices remain) go to the step (ii).
Furthermore, the complexity of the cheapest insertion is $O(n^3)$ but with careful programming it can be $O(n^2 log(n))$ [Frieze 1982]. In brief, the repairing procedure is repeated until the tour becomes feasible and all the extracted customers are inserted into other tours. The procedure follows the steps described in the Algorithm 3:

Algorithm 3 Repairing procedure for Vehicle Routing Problem

Input: an unfeasible tour R_k of a solution s.
$R'_k := R_k$;
while R'_k is unfeasible due to vehicle capacity or depot time window **do**
  Remove a customer c_i from the tour R'_k.
  Insert the customer c_i into another route of s by using the Chepeast Insertion Heuristic.
end while
Output: a feasible tour R_k

2.4.4 Initial Solution

Constructive or greedy algorithms start from an empty solution and construct a solution by assigning values to one decision variable at a time, until a complete solution is generated. They gradually build a feasible solution while keeping an eye on solution cost, but do not contain an improvement phase. Route construction heuristics select nodes (or arcs) sequentially until a feasible solution has been created. Nodes are chosen based on some cost minimization criterion, often subject to the restriction that the selection does not create a violation of vehicle capacity or time window constraints. Sequential methods construct one route at a time, while parallel methods build several

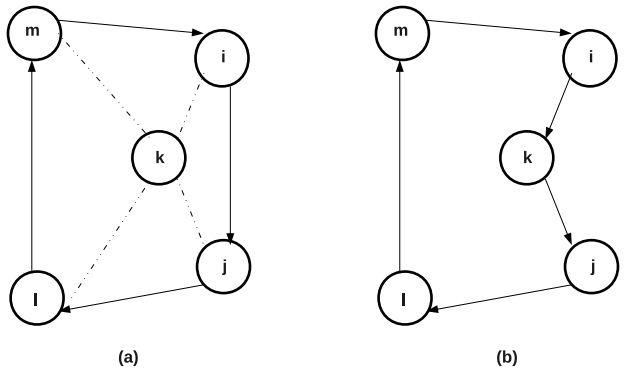


Figure 2.11: Cheapest Insertion Heuristic: (a) Initial subtour with potential insertions.(b) The new subtour after applying the heuristic.

routes simultaneously. The Savings heuristic, originally developed by Clarke & Wright [Clarke 1964] for the classical VRP, is probably the best-known route construction heuristic. It is a saving-based method which merges two routes into a single route by considering the saving distance between the depot and the nodes connected to the depot. It begins with a solution in which every customer is supplied individually by a separate route. Combining the two routes serving respectively customers i and j results in a cost savings of $s_{ij} = d_{i0} + d_{0j} - d_{ij}$. It is summarized in selecting the arc (i, j) linking customers i and j with maximum S_{ij} subject to the requirement that the combined route is feasible. With this convention, the route combination operation is applied iteratively. In combining routes, one can simultaneously form partial routes for all vehicles or sequentially add customers to a given route until the vehicle is fully loaded. The savings heuristic is illustrated in Figure 2.12. In our work, we use the enhanced savings heuristic proposed by Yellow [Yellow 1970]. It has a form $s_{ij} = d_{i0} + d_{0j} - \gamma\, d_{ij}$, where γ is a route shape parameter, and takes values in $[0, 1]$. More γ is larger, more the emphasis is put on the distance between the vertices to be connected. Golden *et al.* [Golden 1977] report that using $\gamma = 0.4$ or 1 yields good solutions taking into account the number of routes and the total length of the solution. The same heuristic is followed to insert dynamic customers in the solution: a partial solution including only new customers is built using the Savings algorithm and these routes are added to the current solution.

2.4.5 Evaluation of the Neighborhood

Often, the evaluation of the objective function is the most expensive part of a local search algorithm and more generally for any metaheuristic. A naive

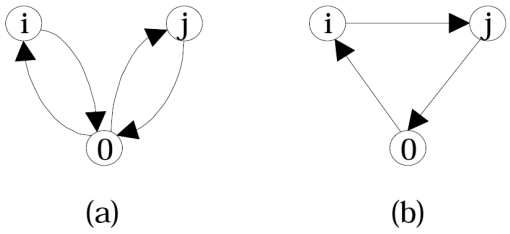


Figure 2.12: The savings heuristic. In the left part, customers i and j are served by separate routes; in the right part, the routes are combined by inserting customer j after i.

exploration of the neighborhood of a solution s is a complete evaluation of the objective function for every candidate neighbor s of $\mathcal{N}(s)$. A more efficient way to evaluate the set of candidates is the evaluation $\triangle(s, m)$ of the objective function when it is possible to compute, where s is the current solution and m is the applied move. This is an important issue in terms of efficiency and must be taken into account in the design of an S-metaheuristic. It consists in evaluating only the transformation (s, m) applied to a solution s rather than the complete evaluation of the neighbor solution $f(s') = f(s \oplus m)$. The definition of such an incremental evaluation and its complexity depends on the neighborhood used over the target optimization problem. This is an important issue in terms of efficiency and must be taken into account in the design of high-achieving metaheuristics especially in dynamic optimization context [Talbi 2009].
Let us present an incremental evaluation of the 2-Opt operator applied to a vehicle tour as it is described in the Figure 2.9. The incremental evaluation can be stated as follows:

$$\Delta f = d_{i_1,i_2} + d_{j_1,j_2} - (d_{i_1,j_1} + d_{i_2,j_2}) \tag{2.2}$$

Thus, for an improving neighbor, we have $\Delta f > 0$ which means that $(d_{i_1,i_2} + d_{j_1,j_2}) > (d_{i_1,j_1} + d_{i_2,j_2})$.

2.4.6 VNS-DVRP Algorithm

The VNS algorithm as applied to DVRP is given in the Algorithm 4. First, the VRP instance that corresponds to the set of customers who appear in the last time slice is given to the algorithm as input data. An initial or partial solution is built according to the constructive Saving heuristic. Then, the algorithm proceeds by selecting a solution from the current neighbor, applies a local search and repair the resulting solution whether it does not fill the

feasibility constraints (vehicle capacity, and depot time window). If the local optimum is better than the incumbent, the algorithm moves there($s := s''$), and continue the search with N_1 ($k := 1$); otherwise the algorithm switches to the next neighbor ($k := k + 1$).

Algorithm 4 VNS for the DVRP.

INPUT: VRP instance which corresponds to the set of customers who are known at the time step T_s
A set of neighborhood structures $\mathcal{N}_k$ for $k = 1, \ldots, k_{max}$ that will be used in the search.
if $T_s = 0$ **then**
  /* First instance */
  s := buildSavingsInitialSolution()
else
  if (Orders are waiting to be scheduled) **then**
    s := getLastTimeSliceSolution()
    s' := buildSavingsPartialSolution(*New customer orders*)
    s := merge(s, s')
  end if
end if
repeat
  $k := 1$
  repeat
    /* Select one solution from the current neighborhood */
    s':= pickRandom($\mathcal{N}_k(s)$)
    /* Apply local search */
    s'' := applyLocalSearch(s')
    s'' := repair(s'') /* In case when solution is unfeasible */
    /* Update current solution and/or neighborhood*/
    if $f(s'') < f(s)$ **then**
      $s := s''$
      $k := 1$
    else
      $k := k + 1$
    end if
  until $k = k_{max}$ {k_{max} is the number of neighborhoods}
until Termination conditions not met
Output: Solution of the partial state of the problem.

The local search results of a consecutively combining of four local search operators: λ-Interchange with $(1, 1)$ moves, λ-Interchange with $(1, 0)$ moves, 2-Opt and 2-Opt*. The motivation here is giving VNS a method which explicitly exchanges subroutes instead of merely single customers; besides, 2-Opt*

is a suitable operator for DVRP, since it allows exchanging the remainder of two routes while not affecting already committed customers. For each heuristic, all possible moves are checked and the best one is performed, i.e. the one which reduces the solution cost the most.

2.5 Simulation and Solving Framework

The strategy that we propose to simulate and solve the problem is based on the works of Montemanni *et al.* [Montemanni 2005b] and Hanshar and Ombuki-Berman [Hanshar 2007]. Basically, the system performs three main tasks, detailed as follows:

1. It obtains new requests which appear during the working day T.
2. It solves the instance of the problem.
3. It updates routes and sends vehicles to serve customers.

The architecture of this simulation framework is given in the Figure 2.13. It consists of two main components to handle the new orders and optimize the current set of routes. The first component, called *Event Manager*, carries out three main tasks: Managing customer requests, assigning orders to specific vehicles, and creating static VRP-like instances. On the other hand, the second component corresponds to the optimization algorithm which solves successive VRP-like instances created by the Event Manager (see Section 2.4 for further details on the optimization algorithm). In the following section, we describe the Event Manager component in details.

2.5.1 Event Manager

The *event manager* serves as an interface between the arrival of new orders and the optimization procedure. Based on the division of the working day into n_{ts} discrete time slices of equal length T/n_{ts}, where T is the length of the working day (as shown in Figure 2.14). The event scheduler creates partial static VRP-like instances and runs in sequence the solving algorithm on these instances (Figure 2.14). Furthermore, it considers/assumes the implicit existence of a centralized dispatcher to communicate next destinations to vehicles, called commitment phase. We assume that the commitment cannot be retracted, i.e. once an order is committed to a driver, this assignment cannot be changed. However, our approach constantly provides a solution covering all the known orders. Among these orders, the assignment of those not yet committed, can be retracted (freely reallocated to other routes or positions of the route).

The idea of dividing the working day into several discrete time slices has been proposed by Kilby *et al.* [Kilby 1998]. The goal is to limit the time given to each partial static problem, hence providing an orderly way to service new requests. Another strategy proposes to include each new customer as soon as its request is received [Gendreau 1999], which may be necessary when urgent requests must be served. However, Montemanni *et al.* [Montemanni 2005b] already showed that restarting the optimizer each time a new request occurs does not necessarily lead to better results on this problem.

The first partial static problem created for the first time slice (*i.e.* at the

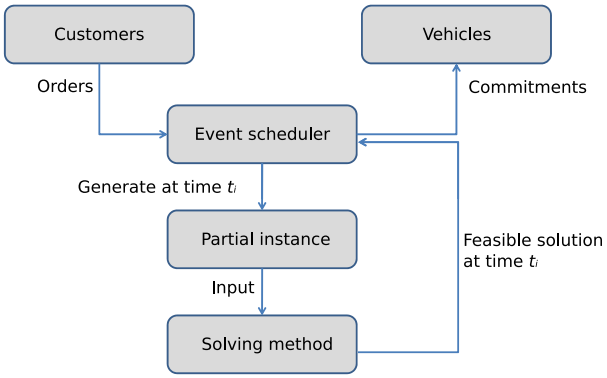


Figure 2.13: Simulation and solving framework for the Dynamic Vehicle Routing Problem (DVRP).

beginning of the working day) consists of all orders remaining from the previous working day. These customers can be considered being left unattended the day before and are known as static customers. The total number of them is determined using the degree of dynamism parameter (*dod*).

The next partial problems will consider all orders received during the previous time slice as well as those that have not been committed to drivers yet. In addition to the parameter n_{ts}, our approach considers two additional parameters which can be defined by user: the *cut-off time* T_{co} and *advanced commitment time* T_{ac}. The cut-off time allows the system to postpone, to the next day, requests which have been received after T_{co}. These requests are considered static for the next day. Furthermore, the idea behind T_{co} is to enable the dispatcher and vehicles to finish the service into the depot time window by avoiding the overload of orders at the end of the working day T. The advanced commitment T_{ac} time permits the system to communicate with the vehicles before leaving the last served location. The main idea is to give

the drivers enough reaction time to process orders. In our simulation, each vehicle starts from the location of the last customer committed to it, with a starting time corresponding to the end of the servicing time for this customer, and with a remaining capacity which equals the capacity left after serving all customers previously committed to it.

At the end of each time slice, the solutions found for the corresponding partial problem are examined, and customer orders c_i with a processing time t_i within the next time slice must be committed to their respective vehicles. Note that the t_i is the moment in which c_i should be served according to its position in the route; it is calculated as the departure time from the previous customer c_{i-1} plus the distance between c_{i-1} and c_i. An exception to this commitment strategy is represented by return journeys to the depot, which happens only in two circumstances: when all the customers have been served, or when the vehicle has used all its capacity. In practice, a vehicle will wait at its last committed customer if neither of the two conditions described above are satisfied. The event manager is presented in Algorithm 5. We recall that the working day T is split into n_{ts} time slices, each one with $T_{ts} = T/n_{ts}$ duration. The set *UnServedOrds* initially contains the orders known from the previous day. The variable T_{step} is initialized to 0, while at the beginning of the working day the location of all the vehicles is set at the depot. A partial problem (*PartialProblem*) is created in each loop step and solved with the procedures that will be described in Section 2.4.

The appropriate commitments (*CommOrds*) are done accordingly to the solution of *PartialProblem*. Customers whose processing times cover the following time windows $[T_{step}, T_{step}+T_{ts}[$ are associated with vehicles. Then, *UnServedOrds* regroups pending orders (orders assigned to a preemptive route, but not committed to a vehicle yet, and those that appeared during the last time slice).

Starting positions, capacities and travel times of the vehicles are updated. The algorithm loops until $UnServedOrds \neq \emptyset$. When all customers are serviced, the vehicles return to the depot.

2.5.2 Vehicle Schedule and Waiting Strategy

Branke *et al.* [Branke 2005b] and Bent *et al.* [Bent 2007] examine the problem of finding an optimal waiting schedule for the vehicles to maximize the probability that a new customer can be incorporated into one of the constructed tours. Branke proves that in the case of several vehicles, waiting may be beneficial because it allows vehicle to remain at strategically favorable locations.

In our approach, the event manager integrates the waiting strategy in the vehicle scheduling. It is based on the recognition that in some circumstances,

Algorithm 5 Event Manager Procedure

$T_{step} := 0$;
The starting position of each vehicle is set at the depot;
$UnServedOrds :=$ orders left the previous day generated according to *dod* (degree of dynamism);
while $UnServedOrds \neq \emptyset$ **do**
  $PartialProblem :=$ problem with orders in $UnServedOrds$;
  Run the solving method on $PartialProblem$;
  $CommOrds :=$orders with processing time $t_i \in [T_{step}, T_{step} + T_{ts}[$;
  Commit orders in $CommOrd$;
  $T_{step} := T_{step} + T_{ts}$; // Update the simulation time
  $UnServedOrds := UnServedOrds \setminus CommOrds$;
  $UnServedOrds := UnServedOrds \bigcup \{$orders appeared during the last time slice $[T_{step} - T_{ts}, T_{step}]\}$;
  Update vehicle positions, capacities, and travel times;
end while
Send all vehicles back to the depot;

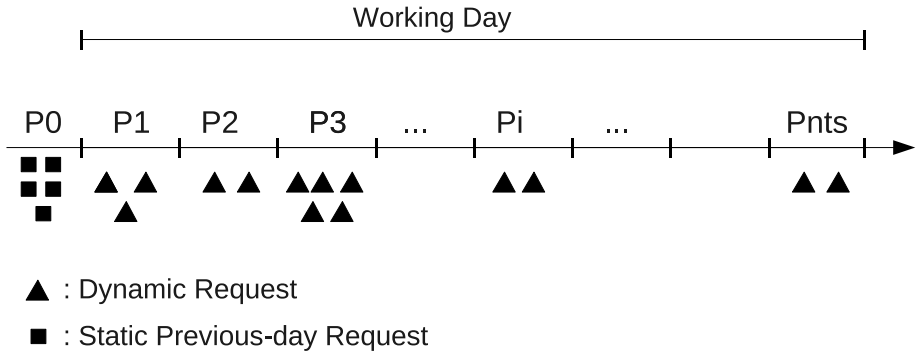


Figure 2.14: Strategy to tackle dynamic instances: A sequence of VRP-like problems.

it might not be desirable to rush to the first real customer, one should anticipate the change by trying to maintain flexibility. Such flexibility can be maintained by having the vehicles wait at appropriate locations in their tours. Instead, if the vehicle waits after servicing a customer at its current position, new requests will have a chance to appear in the same area and be taken into consideration.

This strategy improves the probability of being able to serve additional customers, while reducing the average length of the detour that is necessary to serve them. It is particularly useful for instances in which the objective is the minimization of traveling times, and it is relatively easy to serve all the

customers.

2.6 Experimental Results and Discussion

This section is devoted to the experimental evaluation of VNS-DVRP algorithm.

The adopted benchmarks will be described in Section 2.6.1. The results achieved by the algorithm will be presented and compared with state-of-the-art metaheuristics 2.6.2 and 2.6.3. A study on varying the degree of dynamism is given in Section 2.6.4. Section 2.6.5 assesses the dynamic performances of our algorithm.

2.6.1 Benchmark Description

Our experiments are based on the benchmark data set proposed by Kilby *et al.* [Kilby 1998] and described in Section 1.6. They were derived from publicly available VRP benchmark from three separate VRP sources, namely Taillard [Taillard 1993] (13 instances), Christophides and Beasley[Christofides 1984] (7 instances) and Fisher *et al.* [Fisher 1995] (2 instances). The number of customers to serve can be inferred from the name of the instance. For example, f71 corresponds to Fisher's instance with 70 customers to serve and one single depot. In our experiments, we deal with pickup instances. The driver of the vehicle is not concerned with what he is transporting, but only the quantity that he will pick from the customer. The data set consists of numerous types of service areas, some with uniformly distributed customers, others with clustered customers and a few of them have mixed and irregular distributions. Some properties of these data sets are shown in Table 2.1.

Data set	♯ Customers	Topology and distribution
Christofides	50–199	Customers are either uniformly distributed around the service area or uniformly spread in clusters
Taillard	75–150	Customers present mixed uniform and clustered distributions around the service area
Fisher	71–134	Most of the customers are centralized in the surroundings of the depot. As customers go away this area, they considerably decrease in an irregular manner

Table 2.1: Data sets: Features and properties.

As described in Section 2.5, the simulation of the instance working day requires three parameters: the number of time slots n_{ts}, cut-off time T_{co} and advanced commitment time T_{ac}. In order to compare our algorithm with

previous works, we have divided the working day into $n_{ts} = 25$ time slots. Montemanni *et al.*[Montemanni 2005b] have tested several values for this parameter (10, 25, and 50) on different instances, and found that setting $n_{ts} = 25$ leads to the best trade-off between reactivity to dynamic events and accurate optimization of the static VRP-like problems. Furthermore, we have set the cut-off time $T_{co} = 0.5 \times T$, where T is the length of the working day. Thus, orders which arrive after $T_{co} \times T$ are postponed to the following day, while those that arrive before this time are considered dynamics. Given that the customers arrive according to uniform distribution over the working day, with $T_{co} = 0.5$, the half of customers is considered dynamic, while the other part is static. This corresponds to a $dod = 0.5$. It is important to note that, in some cases, the insertion of customers is not necessary, as the system does not receive events in some time slots. The hardware platform for the experiments was an Intel Core 2 Quad 2.6 GHz machines with 4 GB of memory.

2.6.2 Comparison with State-of-the-Art Metaheuristics

A comparison of the solution quality in terms of minimizing travel distances (costs) is done between our VNS-DVRP and other metaheuristics proposed in literature. The metaheuristics are Montammani *et al.*'s Ant System (AS) [Montemanni 2005b], and Hanshar *et al.*'s [Hanshar 2007] Genetic Algorithm (GA) and Tabu Search (TS).
In order to obtain significant results and carry out this comparison, our approach has been executed 30 times for each instance. In this case, we have simulated the working day T into n_{ts} time slots, where $n_{ts} = 25$. VNS-DVRP algorithm is launched at the beginning of each time slice. The stopping criterion is a fixed number of evaluations. We have fixed the number of evaluations at 5000 evaluations per time slice. Thus, the entire problem will be solved into a number of evaluations equals to 125000 ($25 \times 5000 = 125000$).

The state-of-the-art approaches use the CPU time as stopping criterion. Indeed, for each instance, ACS [Montemanni 2005b] was simulated with 1500 seconds (25 minutes) of CPU time (giving a maximum of 60 seconds for each single optimization) on a Pentium IV 1.4 GHz. For their part, TS and GA algorithms [Hanshar 2007] were run on a Pentium IV 2.8 GHz with a simulation time of 750 seconds (12.5 minutes) of CPU time, *i.e.*, 30 seconds for the optimization of each VRP-like instance.
The choice of using a fixed number of evaluations is motivated by the fact that CPU time is highly dependent on hardware, and not a suitable standard to compare with existent algorithms. However, with the fixed number of evaluations, we guarantee that our algorithm runs in less time than the state-of-the-art algorithms.

Table 2.2 gives the best and average distances of the VNS-DVRP over the 21 Kilby's instances. The results are compared with AS [Montemanni 2005b], GA, and TS, both proposed in [Hanshar 2007]. We highlight the best found solutions into dark shaded cells, and the average results are marked into light shaded cells.

We can see that VNS is very competitive comparatively to other algorithms. It outperforms the other metaheuristics on 3 instances, and gives new best-so-far solutions on these instances. The average of the relative error for the total best results is 1.68%. When we compare the class of S-metaheuristics, our VNS outperforms Hanshar *et al.*'s TS [Hanshar 2007] on 8 instances. Furthermore, VNS outperforms the P-metaheuristic AS on 18 instances and GA on 5 instances.
VNS is able to find high quality solution on Christophides and Beasley instances. These instances are characterized by a uniform distribution of customers in the service area, or uniform spread in clusters.

It is also significant to notice that each AS execution lasts 25 minutes on a Pentium IV 1.5 GHz and each GA and TS execution lasts 12.5 minutes on a Pentium IV 2.8 GHz, which results in a total execution time of 525 and 262.5 minutes respectively. These execution times can be normalized according to the processor used in our case. For that purpose, we use the set of Geekbench benchmarks [Gee 2010]. Geekbench provides a comprehensive set of benchmarks engineered to quickly and accurately measure processor and memory performance. Thereby, when comparing with VNS execution times, the normalized times are 62.08 minutes in the case of AS and 73.5 minutes for GA and TS. The execution of VNS is less time-consuming than GA and comparable with AS, which denotes that our algorithm is reactive and is able to reach competitive solutions in a short time.

2.6.3 Performance on Large Scale Instances

We propose here a study on the performance of VNS on large scale instances *k-series*) generated for dynamic vehicle routing problems and described in Section 1.6. The aim is to deal with instances larger than the standard Kilby's problem set [Kilby 1998] (21 instances, where the number of customers ranges between 50 and 385, although in the literature only instances up to 199 are solved).
Three instances have been chosen for this experiment; *k100*, *k250*, and *k500*, where the number of customers can be inferred from its name. As *k-series* has different variants, we choose to indicate if the instance is with a single

[1] For this instance. the plan of the service area is in a scale 10 times larger than the Fisher's instance.

Table 2.2: Numerical results obtained by our VNS compared to AS, GA, and TS.

Instances	Metaheuristics								
	VNS			AS [Montemanni 2005b]		GA [Hanshar 2007]		TS [Hanshar 2007]	
	Best	Averg.	Time	Best	Averg.	Best	Averg.	Best	Averg.
c50	599.53	653.84	0.75	631.30	681.86	570.89	593.42	603.57	627.90
c75	981.64	1040.00	1.22	1009.36	1042.39	981.57	1013.45	981.51	1013.82
c100	1022.92	1087.18	2.63	973.26	1066.16	961.10	987.59	997.15	1047.60
c100b	866.71	942.81	1.65	944.23	1023.60	881.92	900.94	891.42	932.14
c120	1285.21	1469.24	3.63	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12
c150	1334.73	1441.37	6.22	1345.73	1455.50	1348.88	1386.93	1318.22	1401.06
c199	1679.65	1769.95	10.72	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43
f71	304.32	325.18	1.5	311.18	358.69	301.79	309.94	280.23	306.33
f134[1]	15680.05	16522.18	1.43	15135.51	16083.56	15528.81	15986.84	15717.90	16582.04
tai75a	1806.81	1954.25	1.0	1843.08	1945.20	1782.91	1856.66	1778.52	1883.47
tai75b	1480.70	1560.71	0.68	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72
tai75c	1621.03	1746.07	0.98	1574.98	1653.58	1440.54	1501.91	1406.27	1527.72
tai75d	1446.50	1541.98	0.87	1472.35	1529.00	1399.83	1422.27	1430.83	1453.56
tai100a	2250.50	2462.50	2.33	2375.92	2428.38	2232.71	2295.61	2208.85	2310.37
tai100b	2169.10	2319.72	2.18	2283.97	2347.90	2147.70	2215.93	2219.28	2330.52
tai100c	1490.58	1557.81	1.67	1562.30	1655.91	1541.28	1622.66	1515.10	1604.18
tai100d	1969.94	2100.38	2.08	2008.13	2060.72	1834.60	1912.43	1881.91	2026.76
tai150a	3479.44	3680.35	6.32	3644.78	3840.18	3328.85	3501.83	3488.02	3598.69
tai150b	2934.86	3089.57	5.23	3166.88	3327.47	2933.40	3115.39	3109.23	3215.32
tai150c	2674.29	2928.77	4.65	2811.48	3016.14	2612.68	2743.55	2666.28	2913.67
tai150d	2954.64	3147.38	4.33	3058.87	3203.75	2950.61	3045.16	2950.83	3111.43
Total	50033.15	53341.24	62.08	50876.23	53794.02	49202.73	51089.37	49987.8	52725.85

or multi-depots. In this work, we deal with single depot instances. Thus, the instance *k100* means that there are 100 customers to serve and one single depot. Each instance contains the length of the working day, the occurrence time of each request, the time required to serve these requests, and the number of the available vehicles.
We use the standard way to measure the degree of dynamism in the system: we define the *dod* depending on the proportion of unknown customers, whereas Montemanni *et al.* [Montemanni 2005b] propose to split the working day in two halves of equal length and consider the customer orders that arrive during the second half as static.

In order to deal with this problem, we split the working day T in n_{ts} time slices and collect the requests at the end of each time slot. These requests constitute the new VRP-like instance of the problem. We set n_{ts} to 25 as tested by Montammani *et al.* [Montemanni 2005b]. The degree of dynamism *dod* is fixed to 0.5; this means that a half of the customers is considered as static, while the other half is dynamic. The optimization begins to plan routes with the known static customers at time $t = 0$. The stopping criterion is a fixed number of evaluations. In Table 2.3, we give the number of evaluations

assigned to each instance in dynamic case.

For the static instances, the results are reported after the convergence of the algorithm. We have performed 30 independent runs of each experiment. The results are shown in Table 2.4, which includes the best achieved fitness, the average, the standard deviation, as well as the running time for each instance and each algorithm measured in minutes.

Table 2.3: Number of evaluations assigned to each instance as stopping criterion with $n_{ts} = 25$.

Instances	Single time slice	Complete problem
k100	600	$25 \times 600 = 15000$
k250	600	$25 \times 600 = 15000$
k500	1200	$25 \times 1200 = 30000$

Table 2.4: Solutions obtained by VNS on static and dynamic instances.

Instance	Solution	Static	Dynamic	Time
k100	Best	**1448.18**	**1874.37**	0.74
	Avrg	1529.49	2084.47	
	Std-Dev	36.71	102.14	
k250	Best	**5869.38**	**6845.82**	13.23
	Avrg	6187.80	7251.54	
	Std-Dev	270.88	249.44	
k500	Best	**18582.83**	**24082.73**	130.83
	Avrg	20108.49	24939.88	
	Std-Dev	1457.51	520.99	

Figure 2.15 shows the main trace of VNS-DVRP over the three instances *k100*, *k250*, *k500*. The lower bounds have been computed running our algorithms on the static sub-instance which results of each time slice. These bounds are not attainable by the dynamic algorithms in any case, but we find them useful as reference values for the behavior of our algorithms. They will be useful for the dynamic performance assessment (see Section 2.6.5). It details the track of optima through time by our algorithm. It considers both changing environments; every 600 evaluations for *k100* and *k250* instances and 1200 evaluations for the *k500* instance.

The performance of the tracking depends both on the speed and the severity of environmental changes. Given that the environment is changing, if the solution handled by VNS is distant from the new optimal solution, we can observe a sudden deterioration in the performance of our algorithm. This deterioration depends on the severity of change. Indeed, if new customers appear in a region which is not covered by the assigned vehicles, or if their orders need a quantity that cannot be satisfied by the assigned vehicle; vehicle

detour routes will be planned or new vehicle assignments from the depot will take place in order to respond to these new demands. Therefore, this leads to increase the cost of the solution.

2.6.4 Study on Varying the Degree of Dynamism

We have performed a study on the behavior of our algorithms in relation to different degrees of dynamism. The *dods* take their values in range $[0.5, 1]$. If the *dod* is 0.5, the problem is semi-dynamic, while with a *dod* equal to 1, the problem is completely dynamic. We have done experiments only on the *k-series* instances. The aim is to study the *dod* effect on the quality of the obtained solutions in term of minimizing the fitness function, and the average of the served customers during the working day.

For each instance, 30 runs of VNS are considered. Table 2.5 reports the results obtained by our VNS algorithm with different degrees of dynamism. It indicates the best found solutions, the average, and the percentage as well as the range of served customers.
We can see that when we increase the degree of dynamism, the percentage of served customers starts to decrease. Indeed, since vehicles have to return to the depot at the end of the working day T, dynamic customers that arrive late at the end of the day remain unserved.

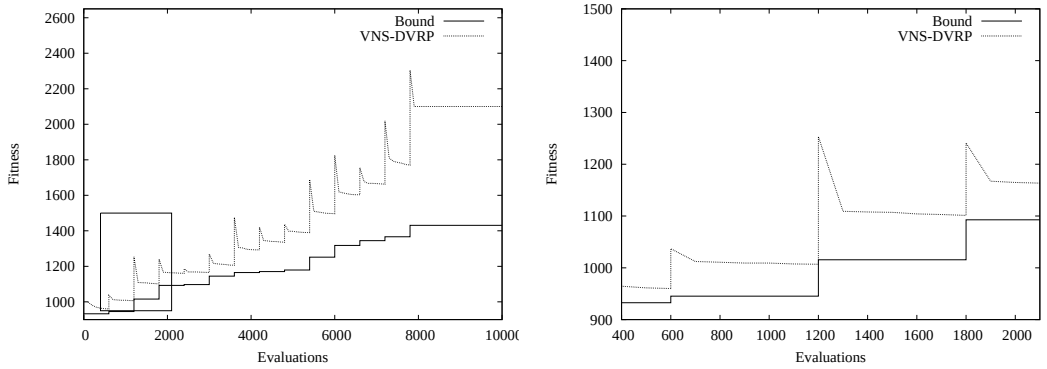
When *dod* =0.6, we observe that almost customers are served for the instances *k100* and *k500*, and totally served for the instance *k250*. However, the total traveled distance is relatively big comparatively to the case when all customers are served ($dod = 0.5$). This can be explained by the fact that when new customers arrive late, some vehicle detours in the existing routes are necessary to respond to these new demands, which leads to an increasing in the total traveled distance.

In the pure dynamic case ($dod = 1$), less than three-quarters of the total number of customers is served for the three instances. This case is closely similar to an emergency scenario, where the priority is to be available for the demands.

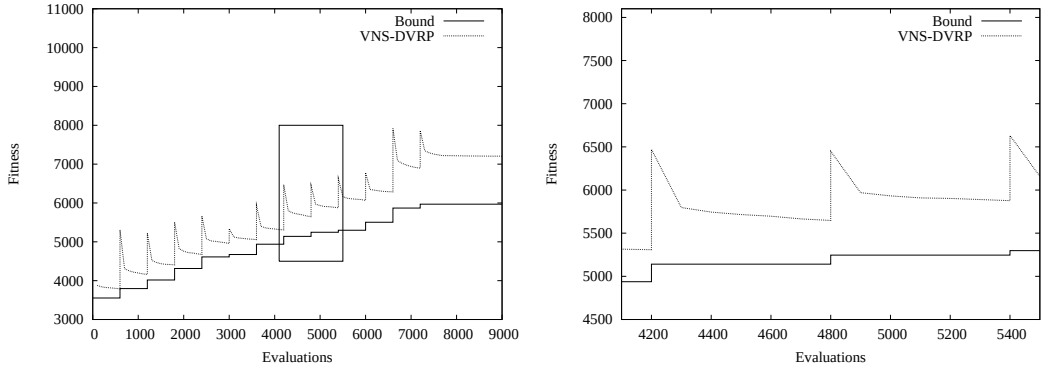
2.6.5 Dynamic Performance Assessment

To assess the dynamic performance measures of our VNS-DVRP, we propose to evaluate the adaptivity of the algorithm according to Weicker's indicators [Weicker 2002] seen in Section 1.5. These measures regroup accuracy, stability and reactivity of the algorithm.

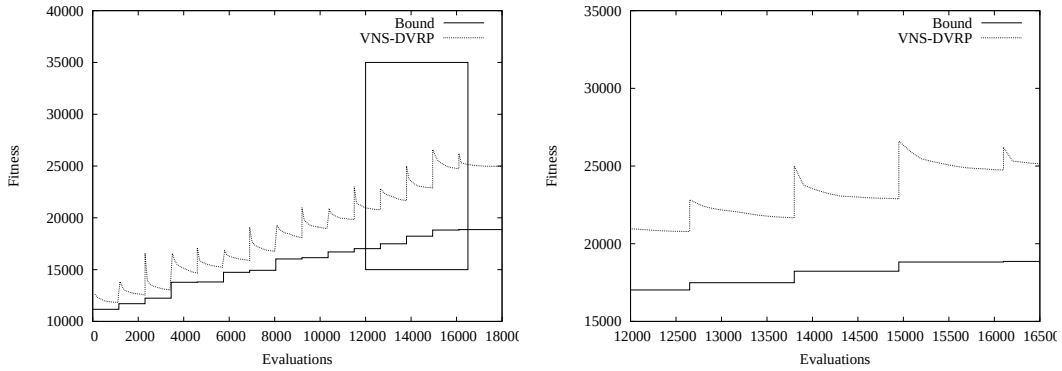
Kilby's instances: For the classical Kilby's instances, we have computed the accuracy at the end of the working day T. Table 2.6 shows the



(a)*k100* instance.



(b)*k250* instance.



(c)*k500* instance.

Figure 2.15: Evolution of VNS-DVRP algorithm mean trace for each instance and the optimum value for each time slice. Each square on the left figure is enlarged in the right figure.

Table 2.5: Solutions obtained by VNS with different degrees of dynamism.

Dods	Inst.	Best	Avrg.	Custom.	Range
0.5	*k100*	1874.37	2084.47	**100%**	**[100-100]**
	k250	6845.82	7251.54	**100%**	**[250-250]**
	k500	24082.73	24939.88	**100%**	**[500-500]**
0.6	*k100*	2313.35	2571.78	99.9%	**[99-100]**
	k250	7361.08	7781.98	**100%**	**[250-250]**
	k500	27354.50	28861.12	99.0%	[493-498]
0.7	*k100*	2436.85	2680.72	95.5%	[94-96]
	k250	8239.43	9244.82	99.6%	[248-250]
	k500	26101.17	28209.95	96%	[478-482]
0.8	*k100*	2214.75	2647.02	88.5%	[87-91]
	k250	8666.36	9365.09	93.8%	[232-236]
	k500	24327.26	27185.13	90.14%	[449-451]
0.9	*k100*	2348.79	2647.33	79.7%	[77-82]
	k250	8184.28	9098.23	86.1%	[212-218]
	k500	23242.42	25640.75	82.41%	[410-413]
1	*k100*	2289.09	2581.81	70.3%	[68-74]
	k250	7567.24	9010.27	76.5%	[188-194]
	k500	22147.62	23764.54	73.09%	[362-366]

accuracy of our algorithms over the 21 instances. It reports the average distances and the lower bounds Min_F^T (*i.e.* best known solutions) for each instance by considering the whole set of customers as static. These solution can be found in literature[2]. From Table 2.6, we see that our VNS-DVRP has an accuracy equal to 0.86 that denotes that it is able to produce good solutions on the conventional dynamic benchmarks. It gets the best accuracy on 6 instances. Its performance is similar to tabu search algorithm which is known as one of the most competitive S-metaheuristics for this problem. However, GA holds the best accuracy with 0.87, since it is a P-metaheuristics, it has the capability to visit more solutions during its search which is a significant advantage.

K-series instances: Table 2.7 shows the accuracy and stability over the three *k-series* instances on different time slices, and the average on the whole working day. These results are plotted in Figure 2.16. As we introduce the new orders to the system at the beginning of each time slot, the algorithm stats reacts immediately and starts its search in order to find the new optimum. So, we have excluded ε-reactivity from this analysis since it provides no significant results (it is always equal to one).

It is interesting here to pay attention to the different behaviors of our algorithm on the three instances. From the accuracy results, we can see that the size of the instance affects differently the performance of

[2] `http://neo.lcc.uma.es/radi-aeb/WebVRP/`

our algorithms. The performance of VNS on the instance *k100* is less effective (0.79) compared to the rest of the instances. Whereas in the medium and bigger instances, VNS performs well with an accuracy that ranges $[0.83 - 0.86]$.
With respect to stability, the average stability values for VNS ranges between 0.168 and 0.177 which is quite stable for a metric which ranges in the interval $[0, 1]$.

Table 2.6: Accuracy of the different metaheuristics on the Kilby's instances.

Instance	Min_F^T	VNS		AS [Montemanni 2005b]		GA [Hanshar 2007]		TS [Hanshar 2007]	
		Best	Accuracy	Best	Accuracy	Best	Accuracy	Best	Accuracy
c50	**521**	599.53	0.87	631.3	0.83	**570.89**	**0.91**	603.57	0.86
c75	**832**	981.64	**0.85**	1009.36	0.82	981.57	0.85	**981.51**	**0.85**
c100	**817**	1022.92	0.80	973.26	0.84	**961.1**	**0.85**	997.15	0.82
c100b	**820**	**866.71**	**0.95**	944.23	0.87	881.92	0.93	891.42	0.92
c120	**1042.11**	**1285.21**	**0.81**	1416.45	0.74	1303.59	0.80	1331.22	0.78
c150	**1028.42**	1334.73	0.77	1345.73	0.76	1348.88	0.76	**1318.22**	**0.78**
c199	**1291.45**	1679.65	0.77	1771.04	0.73	1654.51	**0.78**	1750.09	0.74
f71	**237**	304.32	0.78	311.18	0.76	301.79	0.79	**280.23**	**0.85**
f134	**11620**	15680.05	0.74	**15135.51**	**0.77**	15528.81	0.75	15717.9	0.74
tai75a	**1618.36**	1806.81	0.90	1843.08	0.88	1782.91	**0.91**	**1778.52**	**0.91**
tai75b	**1344.64**	1480.7	0.91	1535.43	0.88	1464.56	**0.92**	**1461.37**	**0.92**
tai75c	**1291.01**	1621.03	0.80	1574.98	0.82	1440.54	0.90	**1406.27**	**0.92**
tai75d	**1365.42**	1446.5	0.94	1472.35	0.93	**1399.83**	**0.98**	1430.83	0.95
tai100a	**2041.33**	2250.5	0.91	2375.92	0.86	2232.71	0.91	**2208.85**	**0.92**
tai100b	**1940.61**	2169.1	0.89	2283.97	0.85	**2147.7**	**0.90**	2219.28	0.87
tai100c	**1406.2**	**1490.58**	**0.94**	1562.3	0.90	1541.28	0.91	1515.1	0.93
tai100d	**1581.25**	1969.94	0.80	2008.13	0.79	**1834.6**	**0.86**	1881.91	0.84
tai150a	**3055.23**	3479.44	0.88	3644.78	0.84	**3328.85**	**0.92**	3488.02	0.88
tai150b	**2656.47**	2934.86	**0.91**	3166.88	0.84	**2933.40**	**0.91**	3109.23	0.85
tai150c	**2341.84**	2674.29	0.88	2811.48	0.83	**2612.68**	**0.90**	2666.28	0.88
tai150d	**2645.39**	2954.64	**0.90**	3058.87	0.86	**2950.61**	**0.90**	2950.83	**0.90**
Average	**1976.03**	2382.53	0.86	2422.68	0.83	**2342.99**	**0.87**	2380.37	0.86

2.7 Conclusion

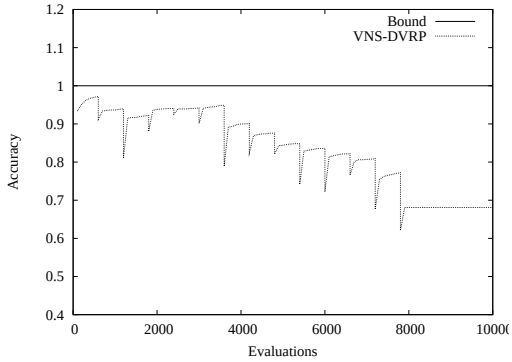
This chapter presented a VNS based approach for solving Dynamic Vehicle Routing Problem. The interest of this approach consists in the ability of shifting from a neighborhood to another one throughout the optimization process. This ability offers an adaptive mechanism for tracking the optimum during the environmental changes. For this proposal, different neighborhoods have been integrated to increase the efficiency of the approach.
Our experiments have been validated on a conventional set of benchmarks as well as a new set of large scale benchmark instances that we have proposed.

Table 2.7: Accuracy and stability of VNS on the dynamic *k-series* instances over different time slices.

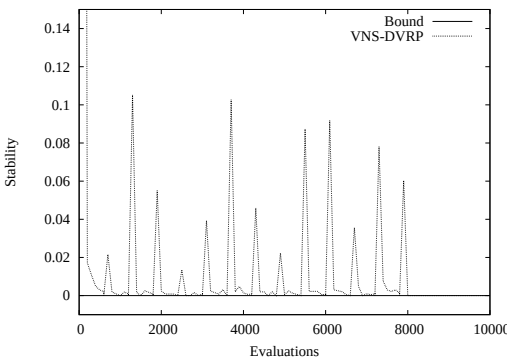
Instance	Time slice	Accuracy	Stability
k100	0	0.972	0.933
	5	0.949	0.039
	10	0.822	0.092
	15	0.681	0.000
	20	0.681	0.000
	25	0.681	0.000
	Avg.	0.797	0.177
k250	0	0.937	0.916
	5	0.924	0.036
	10	0.875	0.055
	15	0.829	0.000
	20	0.829	0.000
	25	0.829	0.000
	Avg.	0.866	0.168
k500	0	0.944	0.885
	5	0.928	0.068
	10	0.819	0.054
	15	0.779	0.000
	20	0.779	0.000
	25	0.779	0.000
	Avg.	0.832	0.168

In order to evaluate the dynamic performance of our approach, several indicators have been used to this end. Weicker's measures allow to assess the accuracy, the stability, and the reactivity of an algorithm throughout the optimization process. Our approach provides very competitive results comparatively to the other state-of-the-art metaheuristics, and Weicker's indicators demonstrated the high adaptivity and stability of our algorithm.

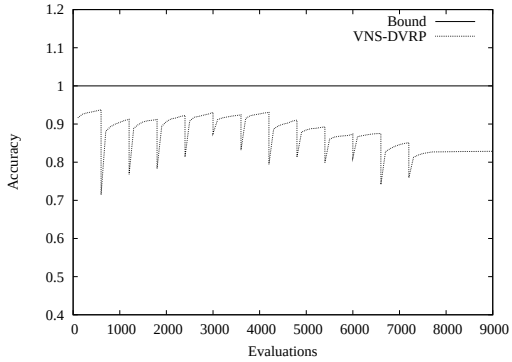
Population-based metaheuristics tend to be more effective in terms of diversification than single solution based ones. However, in terms of intensification search areas, the latter group is known to be more effective. In general, the degree of success of these methods on a given problem depends largely on their ability to strike a balance between exploration and exploitation. The ability of the population-based metaheuristics to sample the search space and the fact that they simultaneously manipulate a group of solutions increase their potential for dynamic problems. Techniques which exploit these qualities are reviewed in the next chapter.



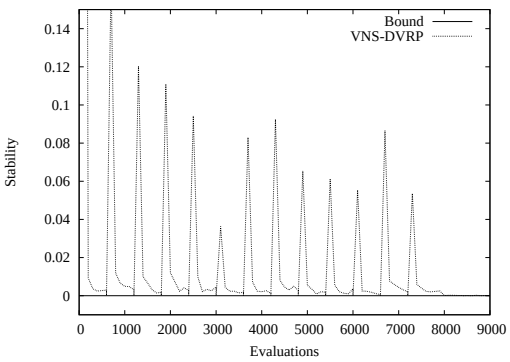
(a) Accuracy over 25 time slices for *k100*.



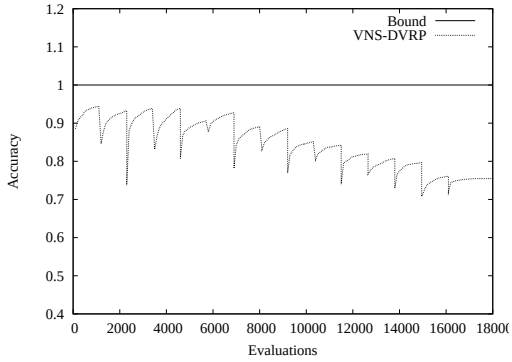
(b) Stability over 25 time slices for *k100*.



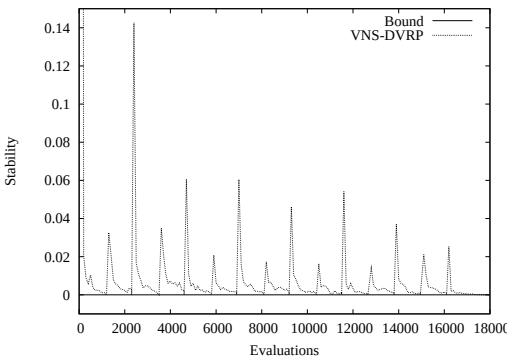
(c) Accuracy over 25 time slices for *k250*.



(d) Stability over 25 time slices for *k250*.



(e) Accuracy over 25 time slices for *k500*.



(f) Stability over 25 time slices for *k500*.

Figure 2.16: Evolution of accuracy and stability of VNS-DVRP across time slices for each instance.

Chapter 3

Population Based Metaheuristics for Solving Dynamic Vehicle Routing Problem

Contents

3.1 Introduction

As mentioned earlier, environmental changes in real life typically do not alter the problem completely, but affect only some part of the problem at a time. For example, for the dynamic vehicle routing problem in which customer orders arrive progressively over the working day, not all the customer orders are canceled in the same time, not all vehicles break down at once, weather changes affect only parts of road, etc. Thus, after an environmental change, there remains some information from the past that can be used in the future. Such problems call for a methodology to track their optimal solutions through time. The required algorithm should not only be capable of tackling combinatorial problems, but should also be adaptive to changes in the environment.

Population based metaheuristics exhibit a number of potential advantages for such purposes. In recent years, there have been increasing interest in using Particle Swarm Optimization (PSO) in dealing with dynamic optimization problems [Eberhart 2001, Carlisle 2002, Li 2006, Blackwell 2007]. There are several characteristics inherent and attributed to PSO that encourage their use for dynamic problems. The underlying principle of PSO is based on swarm intelligence, and hence they are expected to be capable of self-organization and able to adapt to environmental changes. In addition, PSO has proved to be suitable for dynamic environments due to their ability to store and exploit previous solutions. One of the most appealing features for dynamic environments is that, at any given instant, PSO deals with a population of solutions rather than a single solution. Hence, even if the environment changes, it is likely that some solutions in the population remain feasible and retain some of their good quality. Thus, by using PSO, it is possible to formulate general techniques to address the dynamic issues of the vehicle routing problem. For this purpose, we present in this chapter an Adaptive Particle Swarm Optimization for solving Dynamic Vehicle Routing Problem. The underlying idea is to reuse the best solutions gathered in the past that could be beneficial for tracking the shifting optimum over the time. This chapter starts with an introduction to the population-based metaheuristics and the common concepts related to these approaches. In Section 3.3 the standard particle swarm optimization approach is exposed. While the Section 3.4 reports an overview of some P-metaheuristics applied on DVRP. Section 3.5 presents our Adaptive Particle Swarm Optimization (APSO) for solving DVRP. It covers particle's representation, particle movements, memory mechanism, and hybridization with heuristics. In Section 3.7 our approach is tested on a conventional set of benchmarks as well as a new set of large scale instances. In addition, the dynamic performances of our algorithm are assessed and discussed. Finally, we conclude this chapter with a summary and introduce the future step of this thesis.

3.2 Common Concepts for Population-Based Metaheuristics

Population-based metaheuristics (P-metaheuristics) start from an initial population of solutions. Then, they iteratively apply the generation of a new population and the replacement of the current population (see Figure 3.1). In the generation phase, a new population of solutions is created. In the replacement phase, a selection is carried out from the current and the new populations. This process iterates until a given stopping criteria. The generation and the

replacement phases may be memoryless. In this case, the two procedures are based only on the current population. Otherwise, some history of the search stored in a memory can be used in the generation of the new population and the replacement of the old population. Most of the P-metaheuristics are nature-inspired algorithms. Popular examples of P-metaheuristics are evolutionary algorithms, ant colony optimization, scatter search, particle swarm optimization, bee colony, and artificial immune systems. Algorithm 6 illustrates the high-level template of P-metaheuristics [Talbi 2009].

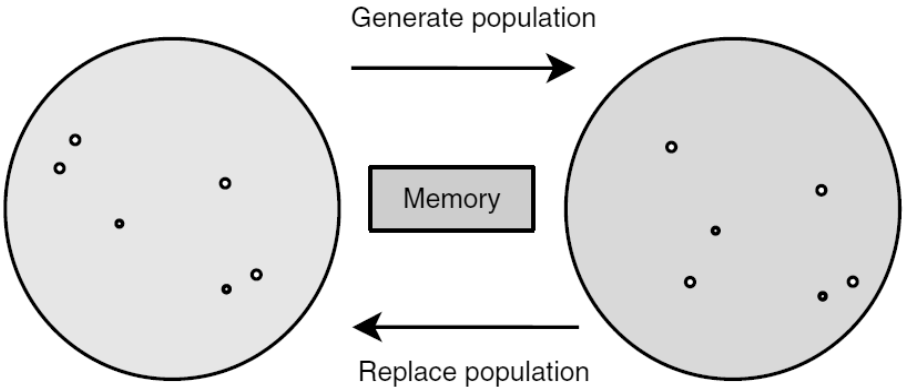


Figure 3.1: Main principles of P-metaheuristics.

Algorithm 6 High-level template of P-metaheuristics.

$P := P_0$; / Generation of the initial population /
$t := 0$;
repeat
  Generate(P'_t); / Generation a new population /
  P_{t+1} := Select-Population ($P_t \cup (P'_t)$); / Select new population /
  $t := t + 1$;
until Stopping criteria satisfied
Output: Best solution(s) found.

P-metaheuristics differ in the way they use the search memory during the search and in generation and selection procedures.

- **Search memory:** The memory of a P-metaheuristic represents the set of information extracted and memorized during the search. The content of this memory varies from a P-metaheuristic to another one (Table 3.2). In most of the P-metaheuristics such as evolutionary algorithms and scatter search, the search memory is limited to the population of solutions. In ant colonies, the pheromone matrix is the main component of

the search memory, whereas in estimation distribution algorithms, it is a probabilistic learning model that composes the search memory.

- **Generation:** In this step, a new population of solutions is generated. According to the generation strategy, P-metaheuristics may be classified into two main categories (Figure 3.2):
  - **Evolution based:** In this category of P-metaheuristics, the solutions composing the population is selected and reproduced using variation operators (e.g., mutation, recombination) acting directly on their representations. A new solution is constructed from the different attributes of solutions belonging to the current population. Evolutionary algorithms and scatter search represent well-known examples of this class of P-metaheuristics.
  - **Blackboard based:** Here, the solutions of the population participate in the construction of a shared memory. This shared memory will be the main input in generating the new population of solutions. The recombination in this class of algorithm between solutions is indirect through this shared memory. Ant colonies and estimation distribution algorithms belong to this class of P-metaheuristics. In the former strategy, the shared memory is represented by the pheromone matrix, while in the latter strategy, it is represented by a probabilistic learning model. For instance, in ant colonies, the generated solutions by ants will affect the generation of solutions by future ants via the pheromone. Indeed, the previously generated solutions participate in updating the pheromone.
- **Selection:** The last step in P-metaheuristics consists in selecting the new solutions from the union of the current population and the generated population. The traditional strategy consists in selecting the generated population as the new population. Other strategies use some elitism in the selection phase where they provide the best solutions from the two sets. In blackboard-based P-metaheuristics, there is no explicit selection. The new population of solutions will update the shared search memory (e.g., pheromone matrix for ant colonies and probabilistic learning model for estimation of distribution algorithms), which will affect the generation of the new population.

As for S-metaheuristics, the search components that allow to define and differentiate P-metaheuristics have been identified. The common search concepts for P-metaheuristics are the determination of the initial population and the generation of the new one.

P-metaheuristic	Search Memory
Evolutionary algorithms	Population of individuals
Scatter search	Population of solutions, reference set
Ant colonies	Pheromone matrix
Estimation of distribution algorithms	Probabilistic learning model
Particle swarm optimization	Population of particles, best global and local solutions
Bee colonies	Population of bees
Artificial immune systems: clonal selection	Population of antibodies

Table 3.1: Search Memories of Some P-Metaheuristics.

3.2.1 Initial Population

Due to the large diversity of initial populations, P-metaheuristics are naturally more exploration search algorithms whereas S-metaheuristics are more exploitation search algorithms. The determination of the initial population is often disregarded in the design of a P-metaheuristic. Nonetheless, this step plays a crucial role in the effectiveness of the algorithm and its efficiency [Talbi 2009]. In the generation of the initial population, the main criterion to deal with is diversification. If the initial population is not well diversified, a premature convergence can occur for any P-metaheuristic. For instance, this may happen if the initial population is generated using a greedy heuristic or a S-metaheuristic (e.g., local search, tabu search) for each solution of the population.

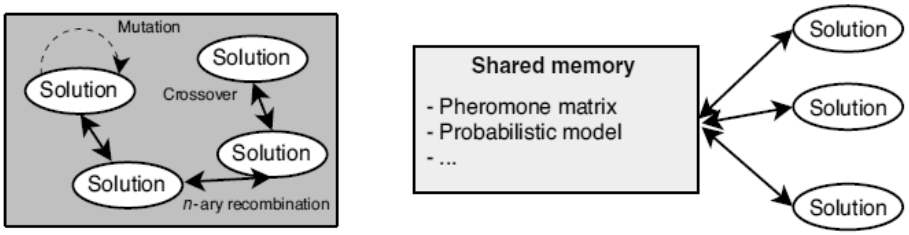


Figure 3.2: Evolution based versus blackboard based strategies in P-metaheuristics.

In some P-metaheuristics such as scatter search [Cung 1997b, Cung 1997a], the diversification criterion is explicitly taken into account in the generation of the initial population. Some diversification criteria are optimized in the generation of the initial population such as maximizing the minimum distance between any two solutions of the initial population: $Max_{i=1,n}(Min_{j=1,i-1}\{d_{ij}\})$ where d_{ij} represents the distance in the decision

space between two solutions i and j and n is the size of the population. Strategies dealing with the initialization of the population may be classified into four categories: random generation, sequential diversification, parallel diversification, and heuristic initialization. They may be analyzed according to the following criteria: diversity, computational cost, and quality of the solutions (Table 3.2). Sequential and parallel diversification strategies provide in general the best diversity followed by the quasi-random strategy. The heuristic initialization provides in general better solutions in terms of quality, but with the expense of a higher computational cost and a reduced diversity. This will depend on the fitness landscape of the tackled optimization problem.

Strategy	Diversity	Computational	Cost Quality of Initial Solutions
Pseudo-random	++	+++	+
Quasi-random	+++	+++	+
Sequential diversification	++++	++	+
Parallel diversification	++++	+++	+
Heuristic	+	+	+++

Table 3.2: Analysis of the different initialization strategies. The evaluation is better with more sign (+).

3.3 Particle Swarm Optimization

Particle swarm optimization is a stochastic population-based metaheuristic inspired from swarm intelligence [Kennedy 2001]. It mimics the social behavior of natural organisms such as bird flocking and fish schooling. Indeed, in those swarms, a coordinated behaviors using local movements emerge without any central control. Originally, PSO has been successfully designed for continuous optimization problems. Its first application to optimization problems has been proposed in [Kennedy 1995]. In the basic model, a swarm consists of N particles flying around in a D-dimensional search space. Each particle i is a candidate solution to the problem, and is represented by the vector x_i in the decision space. A particle has its own position and velocity, which means the flying direction and step of the particle. Optimization takes advantage of the cooperation between the particles. The success of some particles will influence the behavior of their peers. Each particle successively adjusts its position x_i toward the global optimum according to the following two factors: the best position visited by itself ($pbest_i$) denoted as $p_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$ and the best position visited by the whole swarm ($gbest$) (or $lbest$, the best position for a given subset of the swarm) denoted as $p_g = (p_{g1}, p_{g2}, \ldots, p_{gD})$.

The vector $(p_g - x_i)$ represents the difference between the current position of the particle i and the best position of its neighborhood. A neighborhood must be defined for each particle. This neighborhood denotes the social influence between the particles. There are many possibilities to define such a neighborhood, traditionally, two methods are used:

- *gbest* method: In the global best method, the neighborhood is defined as the whole population of particles (Fig. 3.3).

- *lbest* method: In the local best method, a given topology is associated with the swarm. Hence, the neighborhood of a particle is the set of directly connected particles. The neighborhood may be empty so the particles are isolated (i.e., $\varphi_2 = 0$). Figure 3.3 shows three different topologies: complete graph, ring world graph, and small world graph. This model is similar to social science models based on population members mutual imitation, where a stabilized configuration will be composed of homogeneous subpopulations.

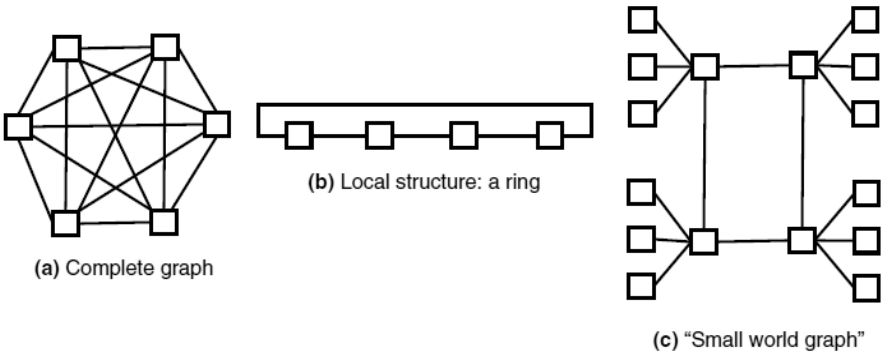


Figure 3.3: Neighborhood associated with particles. (a) *gbest* Method in which the neighborhood is the whole population (complete graph). (b) *lbest* Method where a non complete graph is used to define the neighborhood structure (e.g., a ring in which each particle has two neighbors). (c) Intermediate topology using a small world graph.

  According to the neighborhood used, a leader (i.e., *lbest* or *gbest*) represents the particle that is used to guide the search of a particle toward better regions of the decision space. A particle is composed of three vectors:

  - The x-vector records the current position (location) of the particle in the search space.

- The p-vector records the location of the best solution found so far by the particle.
- The v-vector contains a gradient (direction) for which particle will travel in if undisturbed.
- Two fitness values: The x-fitness records the fitness of the x-vector, and the p-fitness records the fitness of the p-vector.

A particle swarm may be viewed as a cellular automata where individual cell (particles in PSO) updates are done in parallel; each new cell value depends only on the old value of the cell and its neighborhood, and all cells are updated using the same rules. At each iteration, each particle will apply the following operations:

- **Update the velocity:** The velocity that defines the amount of change that will be applied to the particle is defined as:

$$v_i(t) = v_i(t-1) + \varphi_1 \times r_1(p_i - x_i(t-1)) + \varphi_2 \times r_2(p_g - x_i(t-1)) \quad (3.1)$$

where r_1 and r_2 are two random variables in the range $[0, 1]$. The constants φ_1 andφ_2 represent the learning factors. They represent the attraction that a particle has either toward its own success or toward the success of its neighbors. The parameter φ_1 is the cognitive learning factor that represents the attraction that a particle has toward its own success. The parameter φ_2 is the social learning factor that represents the attraction that a particle has toward the success of its neighbors. The velocity defines the direction and the distance the particle should go (see Figure 3.4). This formula reflects a fundamental aspect of human sociality where the socialpsychological tendency of individuals emulates the successes of other individuals.

$$v_i(t) = \omega \times v_i(t-1) + \varphi_1 \times r_1(p_i - x_i(t-1)) + \varphi_2 \times r_2(p_g - x_i(t-1)) \quad (3.2)$$

The elements of v_i are limited to a maximal value $[V_{max}, +V_{max}]$ such as the system will not explode due to the randomness of the system. If the velocity v_i exceeds Vmax (resp. V_{max}), it will be reset to V_{max} (resp. V_{max}). In the velocity update procedure, an inertia weight ω is generally added to the previous velocity [Shi 1998]: The inertia weight w will control the impact of the previous velocity on the current one. For large values of the inertia weight, the impact of the previous velocities will be much higher. Thus, the inertia weight represents a trade-off between global exploration and local exploitation. A large inertia weight encourages global exploration (i.e., diversify the search in the whole

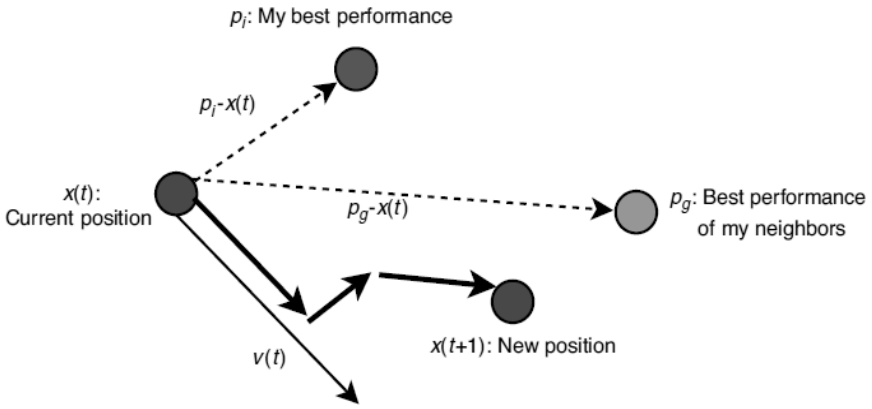


Figure 3.4: Movement of a particle and the velocity update.

search space) while a smaller inertia weight encourages local exploitation (i.e., intensify the search in the current region).

- **Update the position:** Each particle will update its coordinates in the decision space. Then it moves to the new position.

$$x_i(t) = x_i(t-1) + v_i(t) \tag{3.3}$$

- **Update the best found particles:** Each particle will update (potentially) the best local solution $pbest_i$ and the best global solution *gbest* of the swarm is updated.
  Hence, at each iteration, each particle will change its position according to its own experience and that of neighboring particles. As for any swarm intelligence concept, agents (particles for PSO) are exchanging information to share experiences about the search carried out. The behavior of the whole system emerges from the interaction of those simple agents. In PSO, the shared information is composed of the best global solution *gbest*. Algorithm 7 summarizes the template for the PSO algorithm.

3.4 P-Metaheuristics for DVRP: Literature Review

Several P-metaheuristic algorithms have been proposed to solve the dynamic version of the vehicle routing problem. Essentially, Ant Colony System (ACO) and Evolutionary Algorithms (EAs) constitute the major contributions in this application filed.

Algorithm 7 Template of the particle swarm optimization algorithm.

```
Random initialization of the whole swarm ;
for all particles i do
   Initialize x_i and v_i;
   Evaluate f(x_i);
   p_i := x_i;
   p_g := argmin{f(p_i)};
end for
repeat
   for all particles i do
      /* Update velocities*/
      v_i(t) := ω × v_i(t − 1) + φ_1 × r_1(p_i − x_i(t − 1)) + φ_2 × r_2(p_g − x_i(t − 1))
      /*Move to the new position:*/
      x_i(t) := x_i(t − 1) + v_i(t);
      Evaluate (f(x_i));
      if f(x_i) < f(p_i) then
         p_i := x_i ;
      end if
      if f(x_i) < f(p_g) then
         p_g := x_i;
      end if
   end for
until Stopping criteria
```

3.4.1 Ant Colony Optimization (ACO)

For ACO, the approach tends to take advantage of the old information gathered from the previous search by modifying the pheromone matrix. Tian et *al.* [Tian 2003] and Montemanni et *al.* [Montemanni 2005b] propose a new pheromone initialization for new requests that is better than re-optimization process. The aim is to reuse some characteristics of previous good solutions which have an attractive trail in the pheromone matrix. The updating of the pheromone matrix consists in the reinforcement of the trails of these solutions. The initialization of the pheromone matrix is done by solving the partial problem with a greedy insertion algorithm, which is a nearest neighbor heuristic.
Tian et *al.* [Tian 2003] hybridize their algorithm with a 2-Opt heuristic. The hybridization takes place after the updating of the pheromone matrix. Montemani et *al.* [Montemanni 2005b] use a very simple greedy algorithm to improve their solutions. It consists in iteratively selects a customer and tries to move it into another position within its tour or within another tour. Furthermore, Jin et *al.* [Jun 2008] hybridize ACO with EA for a

multi-objective DVRPTW. The authors use the EA for initializing the pheromone matrix as well to optimize it at each iteration.

3.4.2 Evolutionary Algorithms (EAs)

On the Evolutionary algorithms side, for their GA-based system for the DVRP, Hanshar and Ombuki-Berman [Hanshar 2007] and Wang et *al.* [Wang 2007] initialize the population of chromosome randomly. Hanshar generates a random permutation of size n for each chromosome where n is the number of customers left over from the day before.
A dynamic programming is applied by Jih et *al.* [Jih 1999] to generate the initial population instead of creating it randomly. Furthermore, Branke et *al.* [Branke 2005b] propose to seed the population with some heuristics consisting in letting the vehicles wait at suitable locations during their tours, thus influencing the position of the vehicles at the time when the new customer arrives. The objective is to maximize the probability that the additional customer can be integrated into one of tours without violating time constraints.
Different reproduction operators have been proposed in literature. As crossovers: Best-Cost Route Crossover (BCRC) [Hanshar 2007], Partially Mapped Xover (PMX) [Housroum 2006], two-point crossover [Branke 2005b], order crossover (OX), uniform order-based crossover (UOX), merge cross #1 (MX1) and merge cross #2 (MX2) in [Jih 1999].
For mutation, Housroum et *al.* [Housroum 2006] suggest Or-Opt, 1-Opt, and swap. While Hanshar [Hanshar 2007] use the inversion operator. Wang et *al.* [Wang 2007] use relocate operator which consists in changing the assignment of unserved customers to another vehicle. Jih et *al.* [Jih 1999] propose three operators that are: swap, inversion, and re-arrangement.

3.5 Particle Swarm Optimization for DVRP

In this section, we present our PSO-DVRP approach. The different components of our algorithm and their adaptation for DVRP are studied.

3.5.1 Particle Representation

Particle Swarm Optimization is an approach has been used widely in continuous optimization problem, but its adaptation to discrete combinatorial problem remains still difficult. How to encode a schedule is one of the key issues in successfully applying PSO to the DVRP, namely, finding a suitable mapping from problem solution to PSO particle.

3.5.2 Review of Literature

Concerning Vehicle Routing Problems, in the literature, various representations for the particles were proposed.

In [Chen 2006a], the authors propose a hybrid algorithm for solving the capacitated VRP. This algorithm uses Discrete Particle Swarm Optimization (DPSO) combined with simulated annealing (SA). In the encoding process, a particle is represented as a 2D array (double array). Each vector of this representation is a vector of $n \times m$ dimensions, in which n customers have to be served by m vehicles. Every particle is composed of m sections and every section has n discrete points corresponding to the customers. The first dimension in the 2D array of the particle is an $n \times m$ dimension vector in which each entry corresponds to customer. The second dimension is also an $n \times m$ dimension vectors, where every position takes 0 or 1. If the value is 1, it represents that the corresponding customer is served by the relevant vehicle, otherwise it is not served by this vehicle. The position of each particle indicates the relevant sequence of the customers served by each vehicle.

Zhu et al. [Zhu 2006], propose a particle swarm optimization for the VRPTW. The suggested particle coding for this problem is an indirect discrete combinatorial coding. A route for n customers and m vehicles can be presented as a 2-D vector of $n + k - 1$ dimensions. The first array defines the customer or the center nodes (depots) to visit, while on the second vector each dimension defines the sequence of the corresponding customers or the depot into the route. The vehicle routes are retrieved after a decoding the particle representation.

Furthermore in [Wang 2006], the authors introduce a PSO for the Open Vehicle Routing Problem (OVRP). The PSO encoding method is based on real number encoding. For n customers each particle is encoded as a real number vector with n dimensions. The integer part of each dimension or element in the vector represents the vehicle. Thus, the same integer part represents the customer in the same vehicle. The fractional part represents the sequence of the customer in the vehicle route. When the particle position is decoded, the customer is assigned to the vehicle corresponding to the closet integer part. The order of visits is given according to the sorting of the fractional part.

Recently, in [Ai 2009], the authors propose a formulation of the vehicle routing problem with simultaneous pickup and delivery (VRPSPD) and a PSO algorithm to solve it. The solution representation for the VRPSPD with n customers and m vehicles is a $n + 2m$ dimensional particle. Each particle dimension is encoded as a real number. It consists of two parts: the first part relates to customers and the second part concerns vehicles. The first part of the representation consists of n dimensions of particle with each dimension

assigned to a customer.
It is required to set the priority of customers in order to insert them in the existing routes. A random key with n elements is applied. The smaller value of the dimension corresponds to the higher priority of the customer. The second part of the representation is based on the idea of vehicle route orientation. Route orientation of a vehicle is defined as a point in the service map that represents a certain area in which the vehicle is most likely to serve. Consequently, a vehicle route will tend to aggregate around its corresponding route orientation. The decoding method is done by transforming the particle to a priority list of customers to enter the route and a priority matrix of vehicles to serve each customer.

3.5.3 Proposed Representation

We can see that higher dimension is presented in the encoding methods described in the above section and in some cases, dimensions should be rounded to the closest integer number and sorted. It operates with difficulty and consumes much CPU time during the decoding step. Meanwhile, if the position presents the infeasible solution, a repair procedure is necessary.

In our work, to escape coding and decoding phases we reuse the same representation as in VNS approach. It consists in discrete representation which expresses the routes of m vehicles over the n customers to serve. A solution to the problem is represented by a set of routes $S = \{R_1...R_p...R_q...R_m\}$, where R_k is the set of customers serviced by the vehicle V_k. The representation of each route R_k is a permutation of n customers as follows:

$$R_k : (c_0, c_1, c_2, ..., c_i, ..., c_n, c_{n+1}) \tag{3.4}$$

For each customer c_i, we assign the following information:

- (x_i, y_i): coordinates of the customer c_i.
- s_i: Boolean variable which indicates if the customer c_i has been already served or not.
- t_i: processing time of the customer c_i (time in which the customer is served).

Furthermore, for each route R_k served by the vehicle v_k, we keep some information:

- (x_j, y_j): coordinates of the vehicle v_k.
- cap_k: remaining capacity of the vehicle v_k.

- $dist_k$: distance traveled by the vehicle v_k.
- $commit_k$: Boolean variable which indicates if the vehicle v_k has been committed or not.

With this representation, a simple evaluation of the population is provided since the evaluation is carried out only on the unvisited segments of routes.

3.5.4 Velocity vector

The velocity vector v_i of the particle is a vector of n dimensions. It is initialized for each new dimension by a random number between $[1, m]$, where m is the number of the planned routes in the current position of the particle, with the possibility to create a new route for a customer if the cost of this latter is less than its insertion into the existing ones. Each dimension i designs the vehicle route in which the customer c_i is assigned (See Figure 3.5). Each element of v_i is bounded on the range $[-m, m]$. If the velocity v_i exceeds $-m$ (resp. m), it is reset to 1 (resp. m). Thus, it allows to keep a control on excessive wandering of particles outside the neighborhood of the current position. The velocity equations are slightly modified to take into account the nature of the problem. Indeed, in the equation 3.1 the particle positions (p_i, p_g) are replaced with their respective routes index.

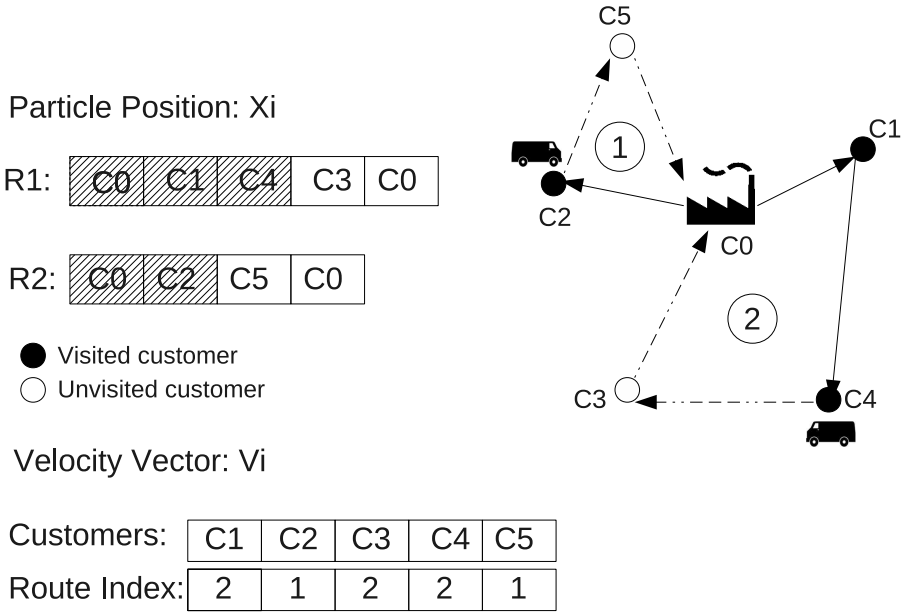


Figure 3.5: Particle position and its velocity vector for PSO-DVRP.

3.5.5 Particle Movement

It is very clear after all the existing literature on VRP that local search is almost mandatory to achieve results of high quality [Rochat 1995, Prins 2004]. This is why we consider the particle's movement as local optimization step for each generation.
Depending on the velocity vector, the particles move after the updating of their current position. The particle's movement is summarized in shifting customers from one route to another according to the new velocity vector. Each customer is reinserted into another route according to the cheapest insertion heuristic, *i.e.*, at a location which minimizes the overall cost of the entire tour (see Figure 2.11). This requires computing the cost of inserting each customer at each location in the route. When it is cheaper to insert an uninserted customer on an empty route rather than an existing route, the customer will be inserted on the new route.
The Figure 3.6 gives an example of the particle position updating. The movement related to the particles is very similar to the λ-Interchange local optimization method that is one of the most successful techniques in the past years [Osman 1993]. This latter is based on the analysis of all the possible combinations for up to λ customers between sets of routes. Hence, this method results in customers either being shifted from one route to another, or being exchanged between routes. The mechanism can be described as follows:
A solution to the problem is represented by a set of routes $S = \{R_1, \ldots, R_p, \ldots, R_q, \ldots, R_k\}$, where R_i is the set of customers served in route i. Thus, new neighboring solutions can be obtained after applying λ-Interchange between a pair of routes R_p and R_q ; to do so, it replaces each subset of customers $S_1 \subseteq R_p$ of size $|S_1| \leq \lambda$ with any other one $S_2 \subseteq R_q$ of size $|S_2| \leq \lambda$. This way, we obtain two new routes $R'_p = (R_p - S_1) \cup S_2$ and $R'_q = (R_q - S_2) \cup S_1$, which are part of the new solution $S' = \{R_1, \ldots, R'_p, \ldots, R'_q, \ldots, R_k\}$. The constraints which are related to the vehicle capacity and depot time-window are relaxed when the particle moves, and a repairing procedure to get feasible position is applied after each movement. The repairing heuristic follows the Algorithm 3 described in the Section 2.4.3.

3.5.6 Swarm Initialization

The initial population is obtained by generating a permutation of customers according to the nearest neighborhood greedy heuristic. The algorithm first starts a route with a random client and repeatedly visits the nearest client

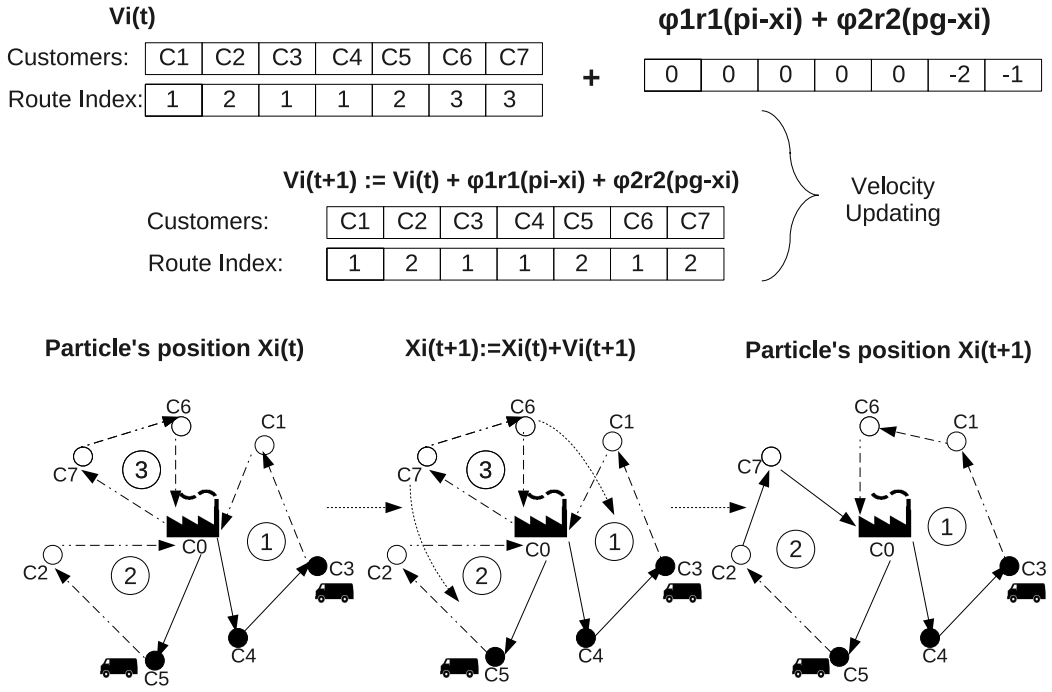


Figure 3.6: Illustration of a particle's movement: the velocity is updated followed by the particle's current position.

until the vehicle constraint (capacity, depot time window) cannot be satisfied. New routes are built until all clients have been visited. The initial population is constituted by the set of vehicle routes which serve customers who were left over from the day before.

3.5.7 Adaptive Memory Mechanism

One could deal with the non-stationarity of VRP by regarding each change as the arrival of a new optimization problem that has to be solved from scratch. However, this simple approach needs a lot of computational time when there is a change. In dynamic environment, the quick response to requests is very important and the immediate requests should be dealt with in real-time. The re-optimization strategy is not a good choice.
Furthermore, when dealing with real-world problems it is rarely the case that the exact same solution will receive the identical fitness at a later stage. However, the dynamic change may cause the optima to be in the neighborhood of an old solution more often. Indeed, unless the change in the problem is extremely strong, probably much effort could be saved and better solution

quality achieved by using an optimization algorithm that is capable of continuously adapting the solution to a changing environment, reusing the information gained in the past. If the change is relatively small, the new optimal solution must be related to the old one. If relative great changes have taken place, the old solution may be a near optimum for the new. Therefore, it is greatly beneficial to make the best of the old information.
A number of authors have addressed the issue of transferring information from the old environment to the new environment by enhancing the optimization algorithm with memory that might allow it to store good (possibly partial) solutions and reuse them later as necessary.
This memory may be implicit by a redundant genome representation, such as diploid chromosomes [Goldberg 1987, Lewis 1998] or explicit by storing and retrieving candidate solutions from a separate memory [Ramsey 1993, Louis 1996, Branke 1999a].
The following subsection gives a brief review of memory-based approaches for DOPs:

1. **Implicit Memory:**

   An algorithm that uses representations containing more information than necessary to define the phenotype (i.e. redundant representations) basically has some memory where good (partial) solutions may be stored and reused later as necessary. We call this kind of memory implicit because it is left to the algorithm to find a way to use it appropriately. The most prominent approach to redundant representations seems to be diploidy. Goldberg and Smith [Goldberg 1987] report on experiments with using diploidy and dominance. Since it is not clear beforehand which allele value (*e.g.* 0 or 1) should be dominant at a particular gene position, Goldberg and Smith favor a triallelic scheme where an allele can take on one of three values O,recessive I, and dominant 1. Tested on a time-varying knapsack problem, they report better adaptive qualities than with a simple GA. Another approach is to use multiploid representations [Lewis 1998], where the genes determining one trait are added in order to determine the phenotypic trait. The phenotypic trait becomes 1 a certain threshold is exceeded and 0 otherwise. The results produced by multiploid representations so far indicate that they are useful in periodic environments where it is enough to remember a few states and important to be able to return to previous states quickly.

2. **Explicit Memory:** The main idea with explicit memory is that remembering old solutions can turn out to be an advantage later on in a dynamic fitness landscape.

Different approaches have been reported in the literature using explicit memory. Louis and Xu [Louis 1996] study scheduling and re-scheduling by means of restarting the EA with individuals evolved by a related problem. Whenever a change occurred, the EA is restarted and the population is initialized with a seed from the old run and the rest randomly. The authors conclude from the experiments that a seed of 5-10 % from the old run produce better and faster results than running the EA with a totally random initialized population after a change occurred.
Ramsey and Greffenstette [Ramsey 1993] introduce an EA model that stored good candidate solutions for a robot controller in a permanent memory together with information about the robot environment. The idea is that if the robot environment becomes similar to a stored environment instance the corresponding stored controller solution is reactivated. For this, they use a simulator to train good strategies for robot movement and obstacle avoidance. In the article the authors report that their technique prevents premature convergence by a higher level of diversity and yielded significant improvements. The only drawback of this approach is that it assumes that the similarity of the robot environment is measurable. Another approach is introduced by Trojanowski and Michalewicz [Trojanowski 1997], in which each individual remembers some of its ancestors solutions. After a change in the environment, the current solution and the memory solutions are re-evaluated and the best solution becomes the active solution, keeping the other solutions in memory. The size of the memory is fixed and individuals from the first generation start with an empty memory buffer. For each of the following generations the parent solution is stored in memory and if the memory is already full the oldest memory solution is removed.
Further Eggermont and colleagues [Eggermont 2001] suggest an EA model, which focuses on a shared memory instead of a local memory, only available to the individual. They implement the model for a bit representation based on a real numerical representation by Branke [Branke 1999a]. In this approach the best individuals from some of the generations are stored in a shared memory. The memory starts out empty and is filled throughout the run. The size of the memory is fixed and different approaches of replacement strategies, when storing individuals, are tested, such as replacing individuals by their age or their contribution to diversity and fitness. Branke [Branke 1999a] and Eggermont et al. [Eggermont 2001] find significant improvement compared to approaches without memory on dynamic test problems.
For the Dynamic Vehicle Routing Problem, Y. Rochat and E. Taillard [Rochat 1995] propose a Tabu Search enhanced with adaptive mem-

ory. This latter stores the routes of the best solutions visited during the search. New solutions are then created by combining routes taken from different solutions of this memory. The combination is applied in similar way than a crossover operator found in Genetic Algorithm. New solution produced by the tabu search is included in the memory if it is not filled yet, or the new solution is better than the worst solution stored in memory, in which case the latter is removed.

3.6 Adaptive Particle Swarm Optimization

We propose here an Adaptive Particle Swarm Optimization (APSO). This approach is closely related to the strategies presented in the section above. In PSO, this memory is intrinsic to the algorithm since each particle is defined by its current position and its best position. Furthermore, particles are related to their best neighborhood positions. These positions represent the set of candidate solutions stored by the particles throughout the search. For the adjustment to the new environment, our algorithm selects the best positions found so far in the population and re-positioning the particles in the search space according to these positions. The result of this mechanism is that the stored candidate solutions will produce outposts at different locations (see Figure 3.7). If the optima returns to the same proximity in the search space the memory points can self-adjust to the translocated optima. After a change in the environment, the current position and the best particle position are re-evaluated and the best solution becomes the active current position. The algorithm of the Adaptive Particle Swarm is summarized in the Algorithm 8.

3.6.1 APSO-DVRP Algorithm

The pseudo-code of the APSO algorithm for solving DVRP is presented in Algorithm 9. The algorithm has as an input a problem represented by a series of static instances, since the working day is split into n time slices corresponding to the different sub-problems. The new customer requests are inserted in the existing routes according to the nearest neighborhood greedy heuristic. Then, the best visited solutions by particles are used for repositioning the particles in the search space of this new environment. The algorithm iterates the main loop of PSO approach by updating the velocity of the particles and their positions. After a time slot, new orders arrive and must be included in the scheduling. Therefore, the algorithm resumes with solving a new instance

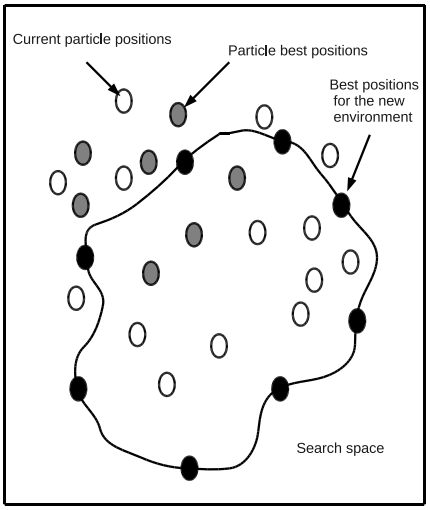


Figure 3.7: Movement of memory points. The curve is an example of a dynamic changing optimum. The black circles are the best memory points for the new environment, the light gray scaled circles are the current positions of particles and the gray circle is the best position found so far by particles.

of the problem. The termination criterion is the end of the journey and the way back of vehicles to the depot.

3.6.2 Hybridization with Heuristics

Over the last years, interest in hybrid metaheuristics has risen considerably among the combinatorial optimization research community, as best results for academic and industrial problems are usually obtained by hybrid algorithms. Indeed, two competing goals govern the design of a metaheuristic: exploration and exploitation. Exploration is needed to ensure that every part of the space is searched enough to provide a reliable estimate of the global optimum. Exploitation is important since the refinement of the current solution will often produce a better solution. P-metaheuristics such as PSO are powerful in the exploration of the search space in the sense they try to (optimize globally) and weak in the exploitation of the solutions found. Therefore, most efficient P-metaheuristics have been coupled with S-metaheuristics or advanced heuristics, which are powerful optimization methods in terms of exploitation (optimize locally). The two classes of algorithms have complementary strengths and weaknesses. We use a Low-level Teamwork Hybrid (LTH) for our algorithm. In [Talbi 2009], Talbi describes this class of hybrids as algorithms in which a given heuristic or S-metaheuristic is embedded into a P-metaheuristic.

Algorithm 8 Template of the Adaptive Particle Swarm Optimization Algorithm

INPUT Dynamic problem P that corresponds to a series of n-static problems $(P = (I_0, t_0, \Delta_0), (I_1, t_1, \Delta_1), \ldots, (I_k, t_k, \Delta_k), \ldots, (I_n, t_n, \Delta_n))$. Each sub-problem I_k occurs at time t_k and spends a certain duration Δ_k.
/* Random initialization of the whole swarm */
for each particles i **do**
  Initialize x_i, v_i;
  Evaluate $f(x_i)$;
  $p_i := x_i$;
  $p_g := argmin\{f(p_i)\}$;
end for
$k := 0$; /* First sub-problem */
repeat
  /* Change in the environment (new static instance (I_k, t_k, Δ_k)) */
  for each particles i **do**
    Evaluate $(f(x_i), f(p_i))$;
    /* Reuse the best solutions found previously by the particles */
    $x_i :=$ AdjustPosition(x_i, p_i);
    /* Update swarm attractor */
    $p_g := argmin\{f(p_i)\}$;
  end for
  repeat
    for each particles i **do**
      /* Update velocities */
      $v_i(t) := \omega \times v_i(t-1) + \varphi_1 \times r_1(p_i - x_i(t-1)) + \varphi_2 \times r_2(p_g - x_i(t-1))$;
      /* Move to the new position */
      $x_i(t) := x_i(t-1) + v_i(t)$;
      Evaluate $(f(x_i))$;
      if $f(x_i) < f(p_i)$ **then**
        $p_i := x_i$;
      end if
      if $f(x_i) < f(p_g)$ **then**
        $p_g := x_i$;
      end if
    end for
  until new change of the environment
  $k := k + 1$;
until termination criterion reached

We have used the 2-Opt heuristic as a local search for APSO. The heuristic is applied for each particle after its movement. For more details on this heuristic see the Section 2.4.3.

Algorithm 9 Pseudo-Code of APSO for the DVRP

INPUT Dynamic Vehicle Routing Problem P that corresponds to a series of n-static instances ($P = (I_0, t_0, \Delta_0), (I_1, t_1, \Delta_1), \ldots, (I_k, t_k, \Delta_k), \ldots, (I_n, t_n, \Delta_n)$). The working day is split into n time slices. Each sub-problem I_k corresponds to the customer orders that arrive at the time slice t_k and which spends a duration Δ_k.

$k := 0$; /*First instance*/
repeat
  for each particle i **do**
    /*(Re)Build swarm according to the new customer orders of the sub-problem I_k */
    $x_i :=$ GreedyInsertion(x_i, *New customer orders of* I_k);
    $p_i :=$ GreedyInsertion(p_i, *New customer orders of* I_k);
    Initialize v_i;
    Evaluate $(f(x_i), f(p_i))$;
    /*Reuse the best solutions stored into the population memory for the repositioning of the particles*/
    $x_i :=$ AdjustPosition(x_i, p_i);
    /* Update swarm attractor */
    $p_g := argmin\{f(p_i)\}$;
  end for
  repeat
    for Each particle i **do**
      /* Update velocity */
      $v_i(t) := v_i(t-1) + \varphi_1 \times r_1(p_i - x_i(t-1)) + \varphi_2 \times r_2(p_g - x_i(t-1))$;
      /* Move to the new position */
      $x_i(t) := x_i(t-1) + v_i(t)$;
      Evaluate $(f(x_i))$;
      /* Update personal best */
      if $f(x_i) < f(x_i)$ **then**
        $p_i := x_i$;
      end if
      /* Update global best */
      if $f(x_i) < f(p_g)$ **then**
        $p_g := x_i$;
      end if
    end for
  until (new change in the problem) /* new time slot is reached */
  $k := k + 1$; /* new instance */
until (the end of the working day T ($k == n$))

3.7 Experimental Results and Discussion

For the experimental validation, we report the performance of APSO on conventional benchmarks and compare our algorithm with state-of-the-art algorithms in Section 3.7.1. A study on varying the degree of dynamism is given in Section 3.7.3. While, Section 3.7.4 assesses the dynamic performances of our algorithm with respect to Weicker's measures.

3.7.1 Comparison with State-of-the-Art Metaheuristics

Similar to VNS's experimental protocol, the solving strategy consists in dividing the working day in a fixed number of time slots $n_{ts} = 25$. Each slot has a duration of T/n_{ts}, where T is the length of the working day. An event manager collects the orders at each slot and generates a static VRP-like instance. Then, the instances are solved by the algorithm. Furthermore, the cut-off time T_{co} is set to 0.5. Demands which arrive after $T_{co} \times T$, are postponed to the following day and are considered statics in the problem, while those that arrive before this time are considered dynamics. Since the demands arrive uniformly, half of customers arrive before the middle of the working day which leads to a situation where the degree of dynamism *dod* is 0.5. More details on the solving strategy could be found in Section 2.5. The APSO algorithm has been implemented using the ParadisEO[1] framework [Talbi 2009]. The experiments are run on Intel Xeon 3 GHz with 2 GB memory.
Our experiments are based on the benchmark data set proposed by Kilby *et al.* [Kilby 1998] and extended by Montemanni *et al.* [Montemanni 2005b][2]. They constitute 21 instances and are derived from the conventional available VRP benchmark data, namely Taillard [Taillard 1993] (13 instances), Christophides and Beasley [Christofides 1984] (7 instances) and Fisher *et al.* [Fisher 1995] (2 instances). The number of customers ranges in $[50, 199]$ and the service area may consist of uniformly distributed customers, clustered customers, or a combination of both (semi-clustered instances). Further details about area topologies can be found in the Section 2.6.1.
A comparison of the solution quality in terms of minimizing travel distances/costs is done between our APSO, VNS and other metaheuristics proposed previously in literature. These metaheuristics are Montammani *et al.*'s Ant System (AS) [Montemanni 2005b], and Hanshar et *al.*'s [Hanshar 2007] Genetic Algorithm (GA) and Tabu Search (TS). As stopping criterion of the algorithm, we have fixed the number of evaluations to 5000 evaluations per time slot. Thus, the entire problem will be solved into a number of evalua-

[1] `http://paradiseo.gforge.inria.fr`
[2] `http://www.fernuni-hagen.de/WINF/inhalte/benchmark_data.htm`

tions equals to 125000 ($25 \times 5000 = 125000$). For the algorithm parameters, we have fixed the inertia weight ω to 1, φ_1 and φ_2 take their values in the range [0.5-1].
For each instance, 30 runs of our algorithms have been considered. For the approaches of literature, the stopping criterion has been fixed to execution time duration. Indeed, ACS [Montemanni 2005b] allows 60 seconds as an algorithm stopping criterion, which leads to a 1500 seconds to the optimization process on a Pentium IV 1.4 GHz. On the other side, TS and GA algorithms [Hanshar 2007] dedicate 30 seconds for each slot, which constitute 750 seconds of CPU Time for the algorithm optimization on a Pentium IV 2.8 GHz. Table 3.3 shows and compares the results obtained by the different metaheuristics. The best, the average distances, and running time in minutes of our algorithms are reported. We highlight the best found solutions into dark shaded cells, and the average results are marked in light shaded cells.

From the Table 3.3, we can see that APSO finds five new best solutions on Kilby's instances. These solutions concern all the classes of instances and customer distribution topology: uniform (c75, c120 and c199), cluster (f71) and mix between uniform and cluster (tai75b). The error relative to the total best results is 2% comparatively to GA.
VNS gives two best solutions comparatively to other algorithms. While, GA [Hanshar 2007] outperforms the other metaheuristics over 9 instances. Finally, AS and TS provide respectively 1 and 4 of the best solutions on Kilby's instances.
It is also important to notice that each AS execution lasts 25 minutes in a Pentium IV 1.5 GHz and each GA and TS execution lasts 12.5 minutes in a Pentium IV 2.8 GHz, which results in a total execution time of 525 and 262.5 minutes respectively. These execution times can be normalized according to the processor used in each case. For that purpose, we used a set of benchmarks [Gee 2010] which allow to quickly and accurately measure and compare processors and memory performances.
When comparing with APSO (113.07 minutes), AS normalized time is 115.63 minutes, while GA and TS normalized time is 151.73 minutes, both of them slower than APSO.

Table 3.3: Numerical results obtained by $APSO$ and VNS compared to AS, GA, and TS.

Instances	Metaheuristics											
	$APSO$			VNS			AS [Montemanni 2005b]		GA [Hanshar 2007]		TS [Hanshar 2007]	
	Best	Averg.	Time	Best	Averg.	Time	Best	Averg.	Best	Averg.	Best	Averg.
c50	575.89	632.38	1.65	599.53	653.84	0.75	631.3	681.86	**570.89**	593.42	603.57	627.90
c75	**970.45**	1031.76	2.49	981.64	1040	1.22	1009.36	1042.39	981.57	1013.45	981.51	1013.82
c100	988.27	1051.5	5.81	1022.92	1087.18	2.63	973.26	1066.16	**961.10**	987.59	997.15	1047.60
c100b	924.32	964.47	3.58	**866.71**	942.81	1.65	944.23	1023.6	881.92	900.94	891.42	932.14
c120	**1276.88**	1457.22	6.88	1285.21	1469.24	3.63	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12
c150	1371.08	1470.95	11.74	1334.73	1441.37	6.22	1345.73	1455.5	1348.88	1386.93	**1318.22**	1401.06
c199	**1640.40**	1818.55	17.97	1679.65	1769.95	10.72	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43
f71	**279.52**	312.35	3.45	304.32	325.18	1.5	311.18	358.69	301.79	309.94	280.23	306.33
f134[3]	15875	16645.89	3.38	15680.05	16522.18	1.43	**15135.51**	16083.56	15528.81	15986.84	15717.9	16582.04
tai75a	1816.07	1935.28	1.66	1806.81	1954.25	1	1843.08	1945.2	1782.91	1856.66	**1778.52**	1883.47
tai75b	**1447.39**	1484.73	1.62	1480.7	1560.71	0.68	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72
tai75c	1481.35	1664.4	2.09	1621.03	1746.07	0.98	1574.98	1653.58	1440.54	1501.91	**1406.27**	1527.72
tai75d	1414.28	1493.47	2.12	1446.5	1541.98	0.87	1472.35	1529	**1399.83**	1422.27	1430.83	1453.56
tai100a	2249.84	2370.58	4.59	2250.5	2462.5	2.33	2375.92	2428.38	2232.71	2295.61	**2208.85**	2310.37
tai100b	2238.42	2385.54	4.43	2169.1	2319.72	2.18	2283.97	2347.9	**2147.70**	2215.93	2219.28	2330.52
tai100c	1532.56	1627.32	3.04	**1490.58**	1557.81	1.67	1562.3	1655.91	1541.28	1622.66	1515.1	1604.18
tai100d	1955.06	2123.9	4.42	1969.94	2100.38	2.08	2008.13	2060.72	**1834.60**	1912.43	1881.91	2026.76
tai150a	3400.33	3612.79	9.15	3479.44	3680.35	6.32	3644.78	3840.18	**3328.85**	3501.83	3488.02	3598.69
tai150b	3013.99	3232.11	9.1	2934.86	3089.57	5.23	3166.88	3327.47	**2933.40**	3115.39	3109.23	3215.32
tai150c	2714.34	2875.93	6.46	2674.29	2928.77	4.65	2811.48	3016.14	**2612.68**	2743.55	2666.28	2913.67
tai150d	3025.43	3347.6	6.95	2954.64	3147.38	4.33	3058.87	3203.75	**2950.61**	3045.16	2950.83	3111.43
Total	50190.87	53538.72	113.07	50033.15	53341.24	62.08	50876.23	53794.02	**49202.73**	51089.37	49987.8	52725.85

3.7.2 Large Scale Instances

To extend the analysis of the performance of our algorithm, we have proposed a set of large scale instances for the dynamic vehicle routing called *k-series*. A detailed description about these instances is given in the Section 1.6. These instances are *k100*, *k250*, and *k500*. We have performed 30 independent runs of each experiment. The results are shown in Table 3.4, which includes the best achieved fitness, the average, the standard deviation, as well as the running time for each instance and each algorithm measured in minutes. Dark shaded cells correspond to best found solutions, while best average results are in light shaded cells. In the dynamic case, the stopping criterion per sub-instance is fixed to $600 \times 25 = 15000$ evaluations for $k100$ and $k250$ instances and $1200 \times 25 = 30000$ evaluations for $k500$ instance. We recall that we keep the same solving strategy as in conventional benchmarks consisting in splitting the working day in 25 times slots and then solving successively each instance of the whole problem (more details in Section 2.6.3). In order to be able to compare our results accurately, we have also performed statistical significance tests. We use a Kolmogorov-Smirnov test to check whether distributions are normal or not and a Levene test to check the data homocedasticity (homogeneity of variances); if both tests are positive, ANOVA is used, otherwise we perform a Kruskal-Wallis test to compare the medians of the algorithms [Cohen 1995]. As a result, all our experiments have a confidence level of 95 % (p-value ≤ 0.05). Table 3.5 is marked with "+" sign if there are statistical differences between a certain pair of algorithms, and with a "−" sign otherwise.

Table 3.4: Solutions obtained by $APSO$ and VNS on static and dynamic instances.

Instance	Algorithm	Solution	Static	Dynamic	Time
k100	*APSO*	Best	1497.70	**1819.01**	6.15
		Avrg	1563.80	1871.25	
		Std-Dev	30.13	29.55	
	VNS	Best	**1448.18**	1874.37	0.74
		Avrg	1529.49	2084.47	
		Std-Dev	36.71	102.14	
k250	*APSO*	Best	6038.08	7658.27	41.89
		Avrg	6722.67	8194.08	
		Std-Dev	277.08	99.82	
	VNS	Best	**5869.38**	**6845.82**	13.23
		Avrg	6187.80	7251.54	
		Std-Dev	270.88	249.44	
k500	*APSO*	Best	20396.5	26347.8	223.29
		Avrg	21157.28	27592.34	
		Std-Dev	312.16	383.07	
	VNS	Best	**18582.83**	**24082.73**	130.83
		Avrg	20108.49	24939.88	
		Std-Dev	1457.51	520.99	

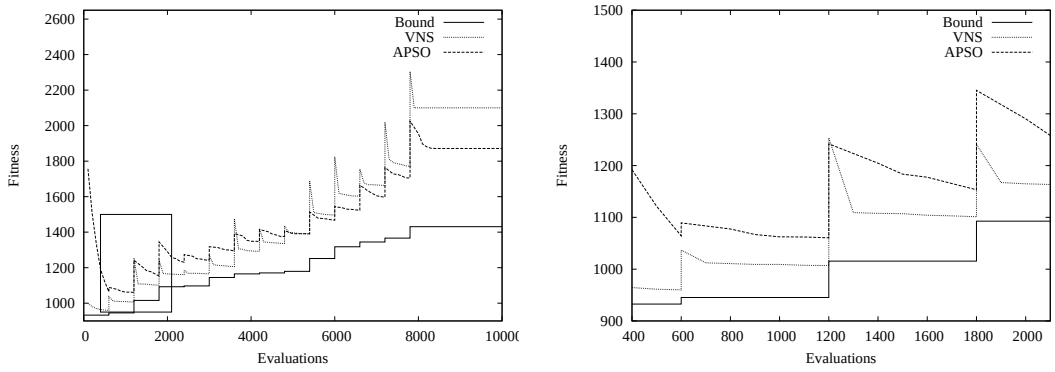
Table 3.5: Statistical results of comparing our algorithms with a multiple comparison test.

State	Instance	Algorithm	Test result	
			APSO	*VNS*
Static	*k100*	*APSO*	−	+
		VNS	+	−
	k250	*APSO*	−	+
		VNS	+	−
	k500	*APSO*	−	+
		VNS	+	−
Dynamic	*k100*	*APSO*	−	+
		VNS	+	−
	k250	*APSO*	−	+
		VNS	+	−
	k500	*APSO*	−	+
		VNS	+	−

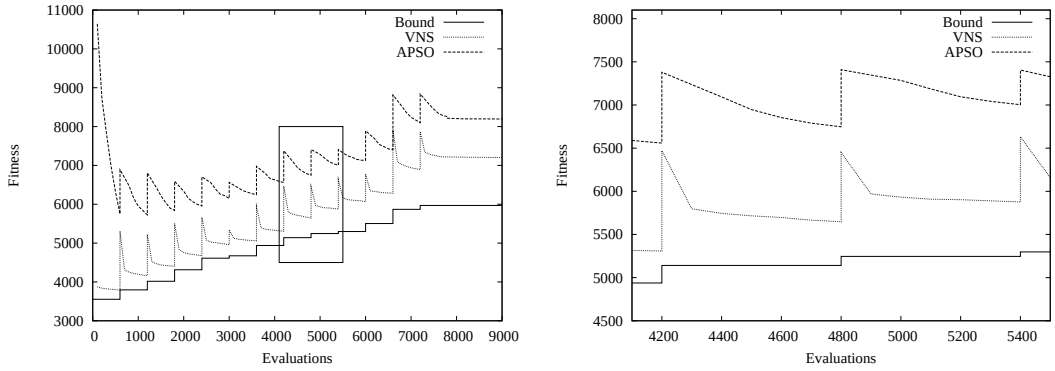
First of all, we study the behavior of our algorithms on the static problem, *i.e.* considering all the customers in the instance as static ($dod = 0$). VNS behaves significantly better than APSO for the three instances, as it finds the best solution in the three instances. Our statistical study (see Table 3.5) reflects that there is a statistical difference between APSO and VNS for the three instances. APSO obtains better results than VNS for *k100*, while it is outperformed by VNS in *k250* and *k500*. All these results are statistically significant, as shown in Table 3.5. Figure 3.8 shows detailed tracking of optima of APSO and VNS through time. Each deterioration indicates a changing of the environment caused by the arrival of new customer orders corresponding to those collected during the last time slice. An interesting issue is the fact that APSO achieves worse results in the dynamic case as well as in the static case for *k250* and *k500*. This is due to the fact that APSO has a slow evolution as shown in the Figure 3.8b and Figure 3.8c, which affects its performance. However, we should take into account the number of 600 and 1200 evaluations allowed for each time slot for the instance k250 and k500 respectively as a constraint of the changing environment.

Regarding the behavior of VNS algorithm, it outperforms APSO in the case of the two bigger instances (*k250* and *k500*). This can be observed in Figure 3.8b and Figure 3.8c, where VNS is able to converge much quicker and reaches better solutions. The situation is the opposite for the instance *k100* (see Figure 3.8a), in which APSO has a good performance from the beginning until the end of the simulation. It must also be noticed that the bound fitness values represented in Figure 3.8 have been computed by running our algorithms on the static instance (all customers are known beforehand) which results of each time slice. These lower bounds represent reference values for the tracking behavior of our algorithms. However, they are not attainable

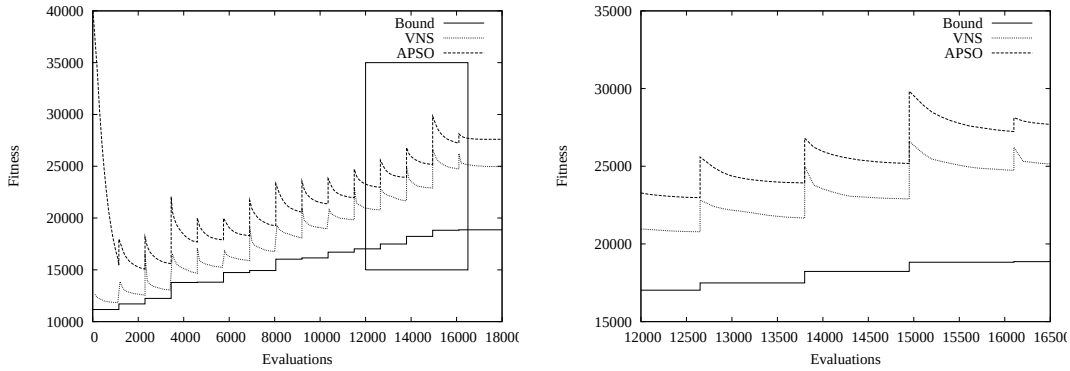
by the dynamic algorithms in any case.



(a) *k100* instance.



(b) *k250* instance.



(c) *k500* instance.

Figure 3.8: The evolution of each algorithm mean trace for each instance; each of them shows also the optimum value for each time slice as obtained by running our algorithms over the static subproblems. Each square on the left figure is enlarged in the right figure.

3.7.3 Study on Varying the Degree of Dynamism

We have performed a study on the behavior of our algorithms in relation to different degrees of dynamism. The *dods* take their values in range $[0.5, 1]$. If the *dod* is 0.5, the problem is semi-dynamic, while with a *dod* equal to 1, the problem is completely dynamic. We have done experiments only on the *k-series* instances. The aim is to present the *dod* effect on the quality of the obtained solutions in term of minimizing the fitness function, and the average of the served customers during the working day. For each instance, 30 runs of $APSO$ and VNS are considered. We keep the same solving strategy as described in the Section 3.7.2. Table 3.6 reports the obtained results on the different degrees of dynamism for the $APSO$ and VNS algorithms. It indicates the best found solutions, the average, and the percentage as well as the range of served customers. When we increase the degree of dynamism, it is easy to see that the percentage of served customers decreases. This is due to the fact that as the problem is bounded by the length of the working day T, the vehicles have to go back to the depot before its closing. In general, customers that arrive at the end of the working day are unserved. For a *dod* equal to 0.5, results are analyzed in Section 3.7.2. From a *dod* upper than 0.6, the percentage of served customers for APSO is better or equal to VNS percentage in all cases, except in the instance *k100* for a *dod* equal to 1. APSO algorithm is able to find solutions which cover more served customers. We can explain this by the diversity of the solutions brought by APSO as a population based metaheuristic. At the opposite VNS covers less customers leading to a situation where the traveled distance is lower than that of APSO.

3.7.4 Dynamic Performances Assessment

We have measured the adaptability of our algorithm according to Weicker's measures [Weicker 2002]. This adaptability covers different measures as accuracy, stability, and ε-reactivity. For the classical Kilby's instances, we have computed the accuracy at the end of the working day T. Table 3.7 shows the accuracy of our algorithms $APSO$ and VNS, compared to other metaheuristics (Ant System (AS), Genetic Algorithm (GA), and Tabu Search (TS)). The accuracy has been computed using the best known solutions of the static instances[4] as the bound to compute accuracy (Min_F^T in Equation 1.10). These best known solutions consider all customers to be static, and then are not feasible solutions for the DVRP. They can be seen as a bound for our algorithms. From Table 3.7, we infer that our algorithms have on average the same average accuracy at the end of the simulation. This accuracy is equal to 0.86 (being

[4] `http://neo.lcc.uma.es/radi-aeb/WebVRP/`

Table 3.6: Solutions obtained by $APSO$ and VNS over different degrees of dynamism.

Dod	Inst.	Algorithm	Best	Avrg.	Custom.	Range
0.5	*k100*	*APSO*	**1819.01**	**1871.25**	**100%**	**[100-100]**
		VNS	1950.47	2129.68	**100%**	**[100-100]**
	k250	*APSO*	7658.27	8194.08	**100%**	**[250-250]**
		VNS	**6903.29**	**7221.88**	**100%**	**[250-250]**
	k500	*APSO*	26347.80	27592.34	**100%**	**[500-500]**
		VNS	**24082.73**	**24939.88**	**100%**	**[500-500]**
0.6	*k100*	*APSO*	**2167.89**	**2295.47**	**100%**	**[100-100]**
		VNS	2313.35	2571.78	99.9%	[99-100]
	k250	*APSO*	8145.35	8706.67	**100%**	**[250-250]**
		VNS	**7361.08**	**7781.98**	**100%**	**[250-250]**
	k500	*APSO*	27535.21	**28761.64**	**99.4%**	**[495-498]**
		VNS	**27354.50**	28861.12	99.0%	[493-498]
0.7	*k100*	*APSO*	**2267.38**	**2491.69**	**96.7%**	**[96-98]**
		VNS	2436.85	2680.72	95.5%	[94-96]
	k250	*APSO*	8856.33	**9165.44**	**99.6%**	**[249- 250]**
		VNS	**8239.43**	9244.82	**99.6%**	[248-250]
	k500	*APSO*	27662.10	28477.72	**97.07%**	**[483-488]**
		VNS	**26101.17**	**28209.95**	96%	[478-482]
0.8	*k100*	*APSO*	**2141.93**	**2381.81**	**89.45%**	[89-90]
		VNS	2214.75	2647.02	88.5%	**[87-91]**
	k250	*APSO*	**8221.32**	**8914.49**	**94.32%**	**[235-237]**
		VNS	8666.36	9365.09	93.8%	[232-236]
	k500	*APSO*	25618.20	**27133.46**	**91.6%**	**[454-465]**
		VNS	**24327.26**	27185.13	90.14%	[449-451]
0.9	*k100*	*APSO*	**2070.50**	**2283.56**	79.32%	[79-80]
		VNS	2348.79	2647.33	**79.7%**	**[77-82]**
	k250	*APSO*	**7944.94**	**8459.35**	**86.5%**	**[215-218]**
		VNS	8184.28	9098.23	86.1%	[212-218]
	k500	*APSO*	24545.60	**25442.21**	**84%**	**[415-424]**
		VNS	**23242.42**	25640.75	82.41%	[410-413]
1	*k100*	*APSO*	**2028.51**	**2191.45**	69.29%	[68-71]
		VNS	2289.09	2581.81	**70.3%**	**[68-74]**
	k250	*APSO*	**7063.96**	**7727.19**	**76.96%**	[191-193]
		VNS	7567.24	9010.27	76.5%	**[188-194]**
	k500	*APSO*	21485.20	**22546.13**	**74%**	**[357-376]**
		VNS	**22147.62**	23764.54	73.09%	[362-366]

1.0 a perfect metric) which denotes that our algorithms are able to produce good solutions on the conventional dynamic benchmarks.

Table 3.8 shows the accuracy and stability over the three *k-series* instances on different time slices, and the average on the whole working day. These results are graphically represented in Figure 3.9. We have excluded ε-reactivity from this analysis since it provides no significant results (it is always equal to one). It is interesting here to pay attention to the different behaviors of our algorithms on the three instances. The accuracy results confirm numerically what we already explained in Section 3.7.2, the size of the instance affects differently the performance of our algorithms. In instance *k100*, the highest accuracy levels correspond to APSO; although VNS is better in the first time

Table 3.7: Accuracy of the different metaheuristics on the Kilby's instances.

Instance	Accuracy				
	APSO	*VNS*	*AS* [Montemanni 2005b]	*GA* [Hanshar 2007]	*TS* [Hanshar 2007]
c50	0.90	0.87	0.83	**0.91**	0.86
c75	**0.86**	0.85	0.82	0.85	0.85
c100	0.83	0.80	0.84	**0.85**	0.82
c100b	0.89	**0.95**	0.87	0.93	0.92
c120	**0.82**	0.81	0.74	0.80	0.78
c150	0.75	0.77	0.76	0.76	**0.78**
c199	**0.79**	0.77	0.73	0.78	0.74
f71	**0.85**	0.78	0.76	0.79	**0.85**
f134	0.73	0.74	**0.77**	0.75	0.74
tai75a	0.89	0.90	0.88	**0.91**	**0.91**
tai75b	**0.93**	0.91	0.88	0.92	0.92
tai75c	0.87	0.80	0.82	0.90	**0.92**
tai75d	0.97	0.94	0.93	**0.98**	0.95
tai100a	0.91	0.91	0.86	0.91	**0.92**
tai100b	0.87	0.89	0.85	**0.90**	0.87
tai100c	0.92	**0.94**	0.90	0.91	0.93
tai100d	0.81	0.80	0.79	**0.86**	0.84
tai150a	0.90	0.88	0.84	**0.92**	0.88
tai150b	0.88	0.91	0.84	**0.91**	0.85
tai150c	0.86	0.88	0.83	**0.90**	0.88
tai150d	0.87	**0.90**	0.86	**0.90**	**0.90**
Average	0.86	0.86	0.83	**0.87**	0.86

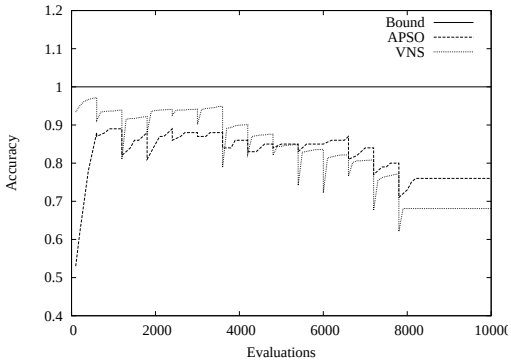
slices (0 to 5), APSO has a better adaptation from the 10th time slice until the end. VNS achieves the best accuracy for all time slices on the instances *k250* and *k500*, whereas APSO performances are poor due to its slow evolution comparatively to VNS, which adapts faster to the changes. Both the final fitness and the accuracy point to a better performance of APSO in *k100* and VNS in the larger *k250* and *k500*. APSO provides enough diversity to achieve better solutions on the smaller instance, while VNS profits from the fast convergence of trajectory based techniques. This is to be considered an essential issue in dynamic optimization due to the reduced available time in each time slice. With respect to stability, APSO is more stable than VNS. The difference between algorithms is noticeable in the three instances: the average stability values for APSO are always less than 0.1, while for VNS it ranges between 0.168 and 0.177 (quite stable for a metric which ranges in $[0, 1]$). This is caused by APSO being a population-based metaheuristic, which provides diversity and different types of solutions when a change occurs in the environment; this means APSO can choose from a wide range of solutions which one is more adequate in the next time slice. However, VNS provides a single solution at the end of each period; thus there is a steeper fitness variation between the end of a time slice and the beginning of the next one.

Table 3.8: Accuracy and stability of APSO and VNS on the dynamic *k-series* instances over different time slices.

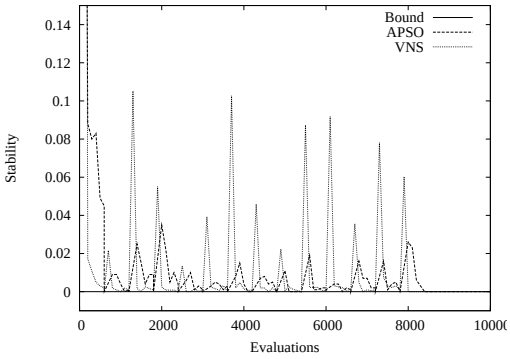
Instance	Time slice	Accuracy		Stability	
		APSO	VNS	APSO	VNS
k100	0	0.876	**0.972**	0.532	0.933
	5	0.885	**0.949**	**0.003**	0.039
	10	**0.865**	0.822	**0.004**	0.092
	15	**0.765**	0.681	**0.000**	**0.000**
	20	**0.765**	0.681	**0.000**	**0.000**
	25	**0.765**	0.681	**0.000**	**0.000**
	Avg	**0.820**	0.797	**0.090**	0.177
k250	0	0.618	**0.937**	**0.334**	0.916
	5	0.748	**0.924**	**0.015**	0.036
	10	0.743	**0.875**	**0.019**	0.055
	15	0.728	**0.829**	**0.000**	**0.000**
	20	0.728	**0.829**	**0.000**	**0.000**
	25	0.728	**0.829**	**0.000**	**0.000**
	Avg	0.716	**0.866**	**0.061**	0.168
k500	0	0.777	**0.944**	**0.277**	0.885
	5	0.805	**0.928**	**0.021**	0.068
	10	0.741	**0.819**	**0.019**	0.054
	15	0.704	**0.779**	0.021	**0.000**
	20	0.704	**0.779**	**0.000**	**0.000**
	25	0.704	**0.779**	**0.000**	**0.000**
	Avg	0.739	**0.832**	**0.048**	0.168

3.8 Conclusion

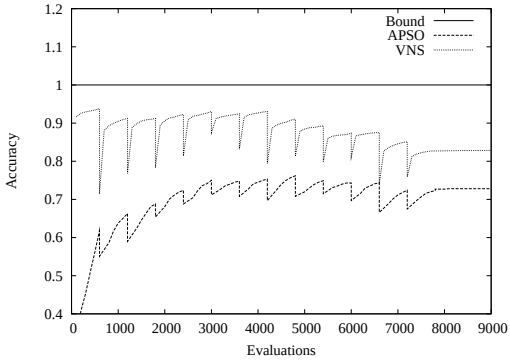
A population-based metaheuristic has been proposed for the DVRP in this chapter. This kind of approaches combine the advantages of manipulating several solutions simultaneously, robustness, and adaptability.
Particle Swarm Optimization takes its inspiration from swarm intelligence and have been successfully applied to most combinatorial problems and have the potential to be effective dynamic solvers. However, once a APSO converges or nearly converges around some solution, it may lose the ability to continue the search after an environment change. A simple remedy is to restart the algorithm after each change, but this may tend to be both expensive and ineffective since valuable information about the search history is discarded with every restart.
A key element in the successful dynamic solver is its ability to maintain diversity throughout the search process while retaining useful past information. This requirement adds another dimension to the traditional issue of balancing diversification and intensification. To enhance the performance of the standard APSO in dynamic environments, we adopt techniques that have been proven as successful methods for dynamic continuous optimization problems. It consists in reusing the information/solutions gathered previously by the particles, and reuse them when the change occurs for repositioning the par-



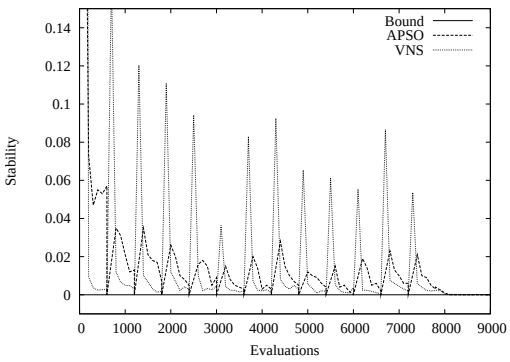
(a) Accuracy over 25 time slices for *k100*.



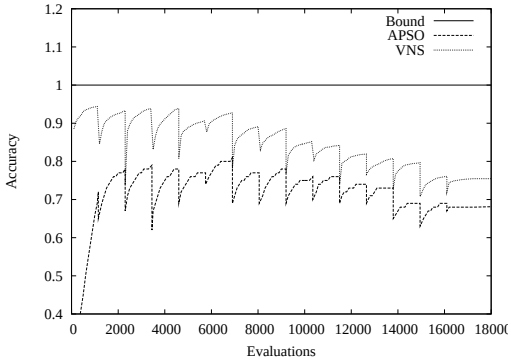
(b) Stability over 25 time slices for *k100*.



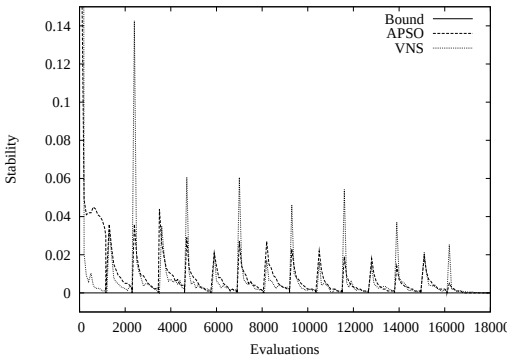
(c) Accuracy over 25 time slices for *k250*.



(d) Stability over 25 time slices for *k250*.



(e) Accuracy over 25 time slices for *k500*.



(f) Stability over 25 time slices for *k500*.

Figure 3.9: Evolution of accuracy and stability across time slices for each instance.

ticles in the search space. This allows a reactive response to the change and a better track for the shifting optimum since the dynamic change may cause the new optima to be in the neighborhood of an old one more often (the two problems are potentially very similar). Thus, storing old solutions can turn

out to be an advantage later in a dynamic fitness landscape. The experimental results showed that our Adaptive Particle Swarm Optimization is able to reach high quality solutions when compared to state-of-the-art metaheuristics, and introduce new best solutions for the DVRP. The dynamic performance measures reinforce these results, when the accuracy of the obtained solutions demonstrates its competitiveness. Another promising techniques for dynamic optimization problems are multiple population approaches. Their main role is to maintain enough diversity in the population. However, these schemes can be enhanced to respond better to the changing environment. These approaches are addressed in the next chapter.

Chapter 4

Multi-Population Based Metaheuristics for Solving Dynamic Vehicle Routing Problem

Contents

4.1 Introduction

Dynamic optimization methods have the main aim to continuously adapt the solution to a changing environment. Approaches try to deal with this changing by introducing a memory that stores the best solutions visited during the previous searches. These solutions are reevaluated and used to initialize the population when change happens in the environment.

The main problem with standard population-based metaheuristics used for dynamic optimization problems appears to be that P-Metaheuristics eventually converge to an optimum and thereby lose their diversity necessary for efficiently exploring the search space and consequently also their ability to adapt to a change in the environment when such a change occurs. For instance,

in Particle Swarm Optimization, if the swarm is converging, the attractors will be close to the optimum position and the swarm will shrink around the optimum due to the attractiveness of particles towards this optimum and the best solutions of the search space in general. When change occurs, if the optimum shifts within the shrinking swarm, then re-optimization will be efficient. However, if the optimum shift at is significantly far from the swarm, it is difficult to reactivate the tracking. Therefore, approaches should counterbalance the effect of diversity loss by maintaining diversity throughout the run. This may be achieved by a multi-population approach. In the multi-population approach, a part of the population clusters around any local optimum it may discover, and remains close to this optimum for further exploration. The remainder of the population continues to search for new local optima, and the process is repeated if any more local optima are found. To track the optimum in such an environment, the algorithm has to be able to follow a moving optimum, and to jump to another optima when the change occurs in a way that makes a previously local optima solution the new optimum. Furthermore, the subpopulation can exchange information during the search and be more reactive to the next change.

In this chapter, we investigate whether a multi-population metaheuristic might also be beneficial in dynamic vehicle routing problems. For this purpose, we elaborate a multiswarm APSO and evaluating the approach on parallel architecture. Parallelizing such a metaheuristics in real-time context is an important aspect due to the hard requirement on search time especially when we deal with dynamic problem in which changes occur in repeated manner and within short intervals. First of all, an overview of the existing multi-population approaches for dynamic optimization problems is given in Section 4.2. Section 4.3 presents the algorithmic design point of view of the parallel model used for our metaheuristic. Afterwards, the main concepts of the multi-swarm optimization is presented in Section 4.4, and is detailed for the DVRP. Section 4.5 deals with the implementation of our parallel metaheuristic. Finally, Section 4.6 reports the performance assessment of our multi-population metaheuristics on a variety of benchmark instances. The main dynamic as well as parallel performance indicators are also provided.

4.2 Multi-population Approaches for Dynamic Optimization Problems

It has been argued in the literature that continuous adaptation only makes sense when the landscapes before and after the change are sufficiently correlated, otherwise it would be at least as efficient to restart the search

from scratch [Branke 1999a]. The main problem with standard population-based metaheuristics used for dynamic optimization problems appears to be that P-metaheuristics eventually converge to an optimum and thereby lose their diversity necessary for efficiently exploring the search space and consequently also their ability to adapt to a change in the environment when such a change occurs. To counterbalance the effect of diversity loss, we can attempt to maintain diversity throughout the run. This may be achieved by a category of metaheuristics which are multi-population approaches. They have been applied essentially on continuous problems as Moving Peaks Problem (MPP) [Branke 1999a].

The underlying idea is to divide the search space into several parts, each explored by one of several subpopulations. A subpopulation continuously searches for new optima, while a number of other sub-populations try to exploit previously detected promising areas. Different multi-population approaches have been proposed for dynamic combinatorial problems. In the area of Evolutionary Algorithms (EAs), Oppacher and Wineberg in [Oppacher 1999] propose a Shifting Balance Genetic Algorithm (SBGA) that consists in dividing the EA population into one main population and a number of smaller colony subpopulations. The task of the main population is to exploit the best found optimum, while the colony populations are forced to explore the different areas of the fitness landscape. A repulsion mechanism is introduced whenever a colony population gets too close to the core population thus driving the colonies far from the core population. Periodically, the colonies update the core population by sending some emigrant solutions.

Another multi-population EA is the Self-Organizing Scouts (SOS) developed by Branke [Branke 2000]. The goal there is to have a number of sub-populations (scouts) watching over the best local optima. For that purpose, a part of the population is split-off when a local optimum is discovered, and remains close to this optimum for further exploration. The remainder of the population continues to search for new local optima that can appear when the environment changes, and the process is repeated if anymore local optima are found. A third multi-population approach is the Multinational Genetic Algorithm (MGA) proposed by Ursem [Ursem 2000]. It structures the population into subpopulations or nations using a procedure called *hill-valley detection*. For two points in the search space, a random sample of the line between these two end points is evaluated. The valley is detected if the fitness of the sample is lower than the fitness of the two end points.

This method is used to determine if an individual is not located on the same peak with the remaining of its population, and hence it should migrate to a different population. The procedure can also lead to the merging of two

populations if it finds that they are situated on the same peak. This detection algorithm works only on points between known optima, the remainder of the space remains unsampled unless by mutation. The frequent evaluations in this approach present its main disadvantage.
Another popular multi-population algorithm is Multi-Swarm Optimization. Different extensions of PSO which solve the problem of change detection and response have been suggested in literature.

In Charged Particle Swarm Optimization (CPSO) [Blackwell 2002], the particles have in analogy with electrostatics charge. Three types of particle swarm can be defined: neutral, atomic and fully-charged. The neutral swarm has no charged particles and is identical with the conventional PSO. Typically, in PSO, there is a progressive collapse of the swarm towards the best position, with each particle moving with reducing amplitude around the best position. This ensures good exploitation, but diversity is lost. However, in a swarm of charged particles, there is an additional collision avoiding acceleration by incorporating electrostatic repulsion between charged particles. This repulsion works against complete collapse and maintains population diversity, enabling the swarm to automatically detect and respond to change. In an *atomic* swarm, half of the particles are charged and the other half is neutral. Animations show that the charged particles orbit a collapsing nucleus of neutral particles, in a picture reminiscent of an atom. This type of swarm therefore balances exploration with exploitation. This approach was extended by the authors in [Blackwell 2004] to Multi-Quantum Swarm (MQS) by replacing the charged particles by quantum particles whose position is based on a probability function centered around the swarm attractor.

Besides, multi-swarm approach has been proposed by Parrott and Li [Parrott 2004]. There, the number and the size of swarms is adjusted dynamically by a speciation and crowding mechanisms called clearing for finding several optima in multimodal landscapes. While it splits up the swarm into several subswarms. This method relies on a speciation radius and has no further diversity mechanism.

In [Blackwell 2006], Blackwell et *al.* elaborate a multi-swarm PSO in which the main idea is to split the population of particles into a set of interacting swarms. These swarms interact locally by an exclusion parameter and globally through a new anti-convergence operator. In addition, each swarm maintains diversity either by using charged or quantum particles. Exclusion is a local interaction between swarms, aimed at ensuring swarm diversity: whenever two swarms are getting too close (one swarm's global best lies within from the other swarm's global best), a distance of the two swarms compete and the one with lower fitness is reinitialized. Besides, the anti-convergence mechanism reinitializes the worst of all swarms once

all swarms have converged. The approach has been tested on a variety of instances of the moving peaks benchmark [Branke 1999b].

4.3 Parallel Design of MP-Metaheuristics for Dynamic Optimization Problems

4.3.1 Interests

Dynamic optimization problems are often NP-hard and CPU time and/or memory consuming. Although the use of metaheuristics allows to significantly reduce the computational complexity of the search process, the latter remains time consuming for many problems in diverse domains of application, where the objective function and the constraints associated with the problem are resource (e.g., CPU, memory) intensive and the size of the search space is huge.

The fast development of technology in designing processors (*e.g.*, multicore processors, General-Purpose Processing on Graphics Processing Units, dedicated architectures), networks (*e.g.*, LAN, WAN, and optical networks), and data storage has made the use of parallel computing more and more popular. Such architectures represent an effective strategy for the design and implementation of parallel metaheuristics. Indeed, sequential architectures are reaching physical limitation (speed of light, thermodynamics). Nowadays, even laptops and workstations are equipped with multicore processors, which represent a given class of parallel architecture. Moreover, the cost/performance ratio is constantly decreasing. The proliferation of powerful workstations and fast communication networks have shown the emergence of clusters of processors (COWs), networks of workstations (NOWs), and large-scale network of machines (GRIDs) as platforms for high-performance computing. Parallel and distributed computing can be used in the design and implementation of multi-population metaheuristics (MP-Metaheuristics) for the following reasons:

- **Speed up the search:** One of the main goals of parallelizing a metaheuristic is to reduce the search time. This helps designing on-line and interactive optimization methods. This is a very important aspect for dynamic optimization class of problems where there are hard requirements on the search time since the problem changes constantly over time. This requirement is stronger when the degree of dynamism increases.

- **Improve the quality of the obtained solutions:** Parallel models for metaheuristics might allow to improve the quality of the search. In-

deed, exchanging information between cooperative metaheuristics will alter their behavior in terms of searching in the landscape associated with the problem. The main goal of a parallel cooperation between metaheuristics is to improve the quality of solutions. Both better convergence and improvement in the quality of solutions may happen.

- **Improve the robustness**: A parallel metaheuristic may be more robust in terms of solving in an effective manner different optimization problems and different instances of a given problem. Robustness may also be measured in terms of the sensitivity of the metaheuristic to its parameters.

- **Solve large-scale problems**: Parallel metaheuristics allow to solve large-scale instances of complex optimization problems. A challenge here is to solve very large instances that cannot be solved by a sequential machine.

4.3.2 Cooperative Parallel Model for MP-Metaheurisitcs

This section aims to present a structured vision of the parallel models and parallel implementations of multi-population metaheuristics. In literature we can find different models for the parallelization of metaheuristics. Talbi in [Talbi 2009] gives a classification based on the level of the parallelization. Three parallel models are defined: algorithmic level, iteration level, and solution level parallel level. The algorithmic and the iteration levels are independent to the problem, where the solution level is dependent to the problem.

In this thesis, we focus on the algorithmic level as it seems to be the closest to the multi-population approaches. In this parallel model, independent or cooperative self-contained metaheuristics are used. If the different metaheuristics are independent, the search will be equivalent to the sequential execution of the metaheuristics in terms of the quality of solutions. Besides, the cooperative model will alter the behavior of the metaheuristics and enable the improvement of the quality of solutions.

In the cooperative model for parallel metaheuristics, the different algorithms are exchanging information related to the search with the intent to compute better and more robust solutions. In designing a parallel cooperative model for any metaheuristics, the same design questions are addressed (see Figure 4.1.):

1. **The exchange decision criterion (when?):** The exchange of information between the metaheuristics can be decided either in a blind

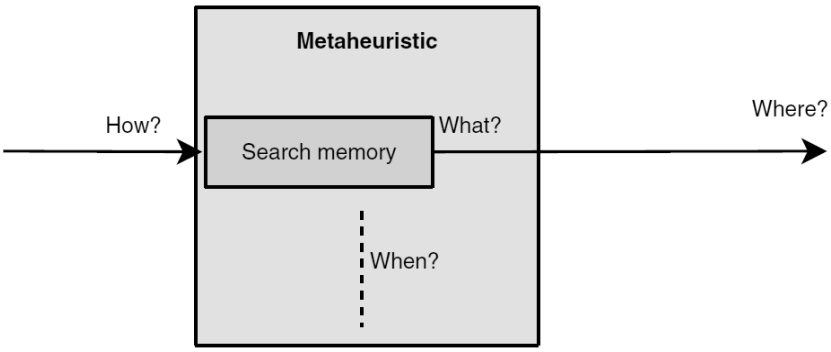


Figure 4.1: Design issues involved by the parallel algorithmic-level model for metaheuristics.

(periodic or probabilistic) way or according to an intelligent adaptive criterion. Periodic exchange occurs in each algorithm after a fixed number of iterations; this type of communication is synchronous. Probabilistic exchange consists in performing a communication operation after each iteration with a given probability. Conversely, adaptive exchanges are guided by some run-time characteristics of the search.

For instance, it may depend on the evolution of the quality of the solutions or the search memory. A classical criterion is related to the improvement of the best found local solution.

2. **The exchange topology (where?):** The communication exchange topology indicates for each metaheuristic its neighbor(s) regarding the exchange of information that is, the source/destination algorithm(s) of the information (Figure 4.2). Several works have been dedicated to the study of the impact of the topology on the quality of the provided results, and they show that cyclic graphs are better [Alba 2005]. The ring, mesh, and hypercube regular topologies are often used. The ring topology may be directional (*i.e.*, directed graph) or bidirectional (*i.e.*, undirected graph). In a hypercube of order k, there are $2k$ nodes, and each node has k neighbors. A complete graph or a random one can also be used. In a complete graph, every node is connected to all other nodes, while in a random graph, a node sends its information to a randomly selected subset of nodes. Different strategies may be used to determine random neighbors, for example, each node has exactly one neighbor that is chosen with equal probability. Whatever the topology, it is important to have a trade-off between the exploration of the search space (less communication and good diversification) and the exploitation of the global search information (more communication and good intensification).

3. **The exchanged information (what?):** This parameter specifies the information to be exchanged between the metaheuristics. In general, it may be composed of:

   - **Solutions:** This information deals with a selection of the generated and stored solutions during the search. In general, it contains elite solutions that have been found, such as the best solution at the current iteration, local best solutions, global best solution, neighborhood best solution, best diversified solutions, and randomly selected solutions. The quality of the solutions must also be sent so that the evaluation of the solutions is not recomputed in the destination metaheuristics. For S-metaheuristics such as local search, the exchanged information is generally the best found solution. For P-metaheuristics, the number of solutions to exchange may be an absolute value or a given percentage of the population. Any selection mechanism can be used to select the solutions. The most used selection strategy consists in selecting the best solutions for a given criteria (e.g., objective function of the problem, diversity, age) or random ones.

   - **Search memory:** This information deals with any element of the search memory that is associated with the involved metaheuristic. For tabu search, the exchanged information may be the short-term or long-term memories. For ant colonies (resp. estimation distribution algorithms), the information may be related to the pheromone trails (resp. the probability model).

4. **The integration policy (how?):** Similar to the information exchange policy, the integration policy deals with the usage of the received information. In general, there is a local copy of the received information. The local variables are updated using the received ones. For instance, the best found solution is simply updated with the global best between the local best solution and the neighboring best solution. For P-metaheuristics, any replacement strategy may be applied to the local population by the set of received solutions. For example, an elitist replacement will integrate the received k solutions by replacing the k worst solutions of the local population. In ant colonies, the local and the neighboring pheromone matrices may be aggregated in a linear manner.

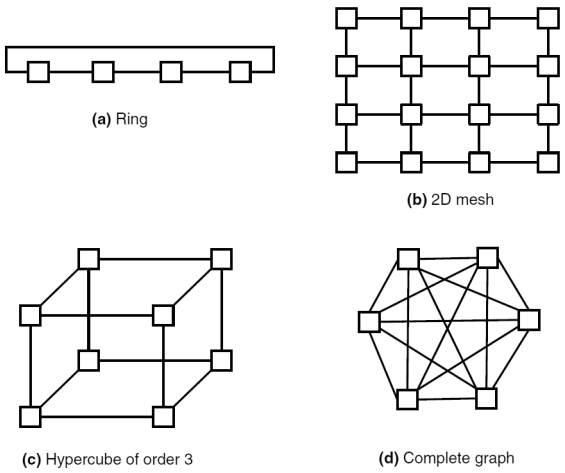


Figure 4.2: Some classical regular topologies for exchanging information.

4.4 Parallel Multi-Swarm Optimization for DVRP

Inspired by the multi-swarm approaches, we investigate in this section whether the parallel multi-swarm approach might also be beneficial in dynamic vehicle routing environments. Here, we use this general idea to maintain particles on several optima simultaneously, which should be helpful in our context. A part of the population clusters around any local optimum it may discover, and remains close to this optimum for further exploration. The remainder of the population continues to search for new local optima, and the process is repeated if any more local optima are found. This technique is expected to work well for a class of dynamic functions consisting of several optima, where the dynamism is expressed by small changes to the optima locations. These have been argued to be representative of real world problems [Branke 1999a] and therefore our problem. To track the optimum in such an environment, the algorithm has to be able to follow the shifting optimum, and to "jump" to another optimum when a change occurs in a way that makes a previously local optimal the new global optima.

Among the most widely known parallel algorithmic-level models for particle swarm optimization, we find the *island model*. In this well-known model, each node is responsible for the evolution of one sub-swarm. It executes all the steps of the algorithm from the velocity updating to the attractors updating of the subpopulation. Each island may use different parameter values and different strategies for any search component such as velocity updating, particle updating, and encodings.

After a given number of iterations (synchronous exchange) or when a condition holds (asynchronous exchange), the migration process is activated. Then, exchanges of some selected particles between sub-swarms are realized, and received particles are integrated into the local sub-swarm. The selection policy of emigrants indicates for each island in a deterministic or stochastic way the individuals to be migrated. The stochastic or random policy does not guarantee that the best individuals will be selected, but its associated computation cost is lower. The deterministic strategy (wheel, rank, tournament, or uniform sampling) allows the selection of the best individuals. The number of emigrants can be expressed as a fixed or variable number of particles, or as a percentage of particles from the swarm. The choice of the value of such parameter is crucial. Indeed, if the number of emigrants is low, the migration process will be less efficient as the islands will have the tendency to evolve in an independent way. Conversely, if the number of emigrants is high, the APSO is likely to converge to the same solutions. The replacement/integration policy of immigrants indicates in a stochastic or deterministic way the local individuals to be replaced by the newcomers. The objective of the model is to delay the global convergence and encourage diversity (This paradigm is illustrated in Figure 4.3.).

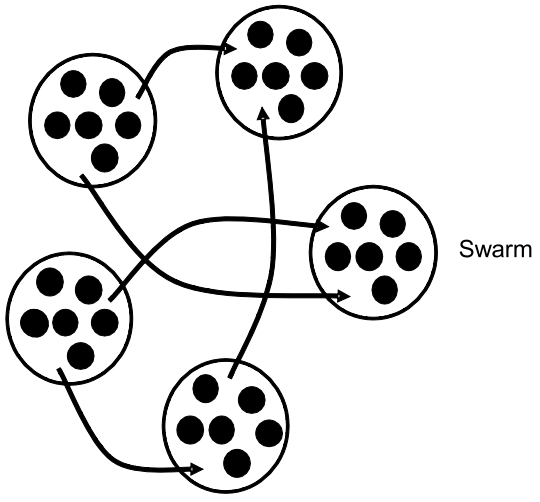


Figure 4.3: Parallel insular model for multi-swarm.

Based on the considerations above, we propose a parallel multiswarm variant of APSO that we call Multi-Adaptive Particle Swarm Optimization (MAPSO). The algorithm is presented in Algorithm 10. It starts with the initialization of the population of particles and then iterates a main loop with three stages: Test for function change, reusing of solutions that belong

to the adaptive memory and the updating of the personal and swarm attractors. When a change in the environment is detected, all the particles are re-positioned according to adaptive memory mechanism and evaluated. Consequently, the personal attractor is updated. Then, the particles move through the search space according to the equations 3.2 and 3.3 , and update their both attractors if necessary at each generation. When the migration criterion is filled, the exchange of particles is carried out between the swarm and its neighbor(s) according to the exchange topology.

For the dynamic vehicle routing problem, as previously mentioned, at each time step, the VRP like-instance that corresponds to the set of customers who arrived in the last time slot is given to the algorithm as an input data.
An initial population is built according to a greedy neighbor heuristic. Then, the algorithm proceeds by updating the particles velocity as well as their positions. The particles move in the search space seeking a better position. The global attractor which corresponds to the particle with the best position (in terms of fitness) is updated. When the migration criterion is reached after a number of iterations, a selection is performed on the population and a set of the best particles (*i.e.* those with the best positions) is chosen for migration toward a neighbor population defined by the topology of the multi-swarm. In the other part, the immigrant particles coming from another population are received and integrated into the local population by replacing the worst particles (*i.e.* those with the less good positions in terms of fitness).

4.5 Parallel Implementation of MP-Metaheuristics

Efficient implementation of parallel metaheuristics is a complex task that depends on the type of the parallel architecture used. In order to implement efficiently our parallel multi-swarm optimizer, we choose ParadisEO software framework to design and implement the parallel and distributed model for our metaheuristic. ParadisEO[1] is a framework dedicated to the reusable design of parallel hybrid metaheuristics by providing a broad range of features, including EAs, local search methods, parallel and distributed models, different hybridization mechanisms, etc. The rich content and utility of ParadisEO increases its usefulness.

ParadisEO is a C++ LGPL white-box open source framework, based on a clear conceptual separation of the metaheuristics from the problems they are intended to solve. This separation, and the large variety of implemented

[1] http://paradiseo.gforge.inria.fr

optimization features, allow a maximum code and design reuse. Changing existing components and adding new ones can be easily done, without impacting the rest of the application. ParadisEO is one of the rare frameworks that provide the most common parallel and distributed models, portable on distributed-memory machines and shared-memory multiprocessors, as they are implemented using standard libraries such as MPI, PVM and PThreads. The models can be exploited in a transparent way - one has just to instantiate its associated ParadisEO components. The user has the possibility of choosing, by a simple instantiation, the MPI or the PVM for the communication layer. The models have been validated on academic and industrial problems, and the experimental results demonstrate their efficiency [Talbi 2009].

The architecture of ParadisEO is layered as it is illustrated in Figure 4.4. From a top-down view, the first level supplies the optimization problems to be solved using the framework. The second level represents the ParadisEO framework, including optimization solvers, embedding single and multicriterion P/S metaheuristics (evolutionary algorithms, particle swarm optimization, variable neighborhood search, etc.). The third level provides interfaces for MPICH-G2 based programming. The fourth and lowest level supplies communication and resource management services. The implementation relies on invariant elements provided by the ParadisEO framework, providing support for the insular model approach, as well as for distributed and parallel aspects concerning other models as the parallel population evaluation. In this context, deployment related aspects are transparent, the focus being oriented on the application-specific elements. The main steps to be performed, in order to configure the environment and to deploy the algorithm, consist in specifying the individuals encoding, the specific operators and the fitness function. Furthermore, elements concerning selection mechanisms and replacement strategies must be specified, along with configuration parameters (number of individuals, number of generations, etc.).

4.6 Experimental Results and Discussion

The underlying support for performing the experiments was GRID5000[2], a French nation-wide experimental grid, connecting several sites which host clusters of PCs interconnected by RENATER[3] (the French academic network). GRID5000 is promoted by CNRS, INRIA and several universities[4].

[2]GRID5000 web site: https://www.grid5000.fr

[3]Rseau National de Tlcommunications pour la Technologie, lEnseignement et la Recherche - http://www.renater.fr

[4]CNRS - http://www.cnrs.fr/index.html; INRIA - http://www.inria.fr.

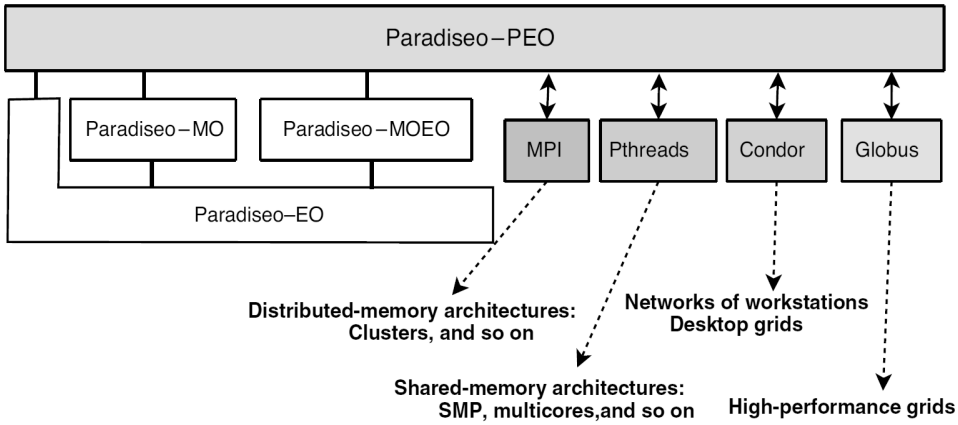


Figure 4.4: The layered architecture of ParadisEO.

At this time the GRID is gathering more than 2932 processors representing 7468 cores with around 2.5 Tb of cumulated memory and more than 100 Tb of non-volatile storage capacity. Inter-connections sustain communications of 10 Gbps. The initial target point was to achieve 5000 processors for 2007 in the platform. It has been reframed at 5000 cores, and was reached during winter 2008-2009, regrouping nine centers at the beginning and eleven since 2011: Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis, Toulouse, Reims, Luxembourg.

The GRID is designed to allow a per reservation utilization of the resources - no interferences may occur during the experiments, the allocation of the resources being associated only with the user which requested the reservation. The demanded resources are completely available during the entire experimentation time, unless in exceptional events occur.

For the experimental validation of our approach, we evaluate the computational results of MAPSO-DVRP algorithm. Therefore, we report its performance comparatively to other metaheuristics on conventional benchmarks in Section 4.6.1, and with a study on varying the number of subpopulations on large scale instances in Section 4.6.2. Section 4.6.3 assesses the dynamic performances of MAPSO. In addition, we evaluate in Section 4.6.4 the scalability of our algorithm by measuring its parallel performances.

4.6.1 Comparison with State-of-the-Art Metaheuristics

We compare the quality of the solutions obtained by MAPSO-DVRP with the best reported results from the state-of-the-art in dynamic vehicle routing problems. We use the conventional Kilby's benchmarks [Kilby 1998] summarized in 21 dynamic instances derived from three well-known data sets as

benchmarks: Christofides and Beasley [Christofides 1984] (7 instances), Taillard [Taillard 1993] (12 instances) and Fisher [Fisher 1995](2 instances). The data sets consist of numerous types of service areas, some with uniformly distributed customers, others with clustered customers and a few of them have mixed and irregular distributions. Further details and properties of the instances can be reviewed in Table 2.1.

We follow the solving strategy consisting in dividing the working day into several time slots and by solving the instance corresponding to the set of customers who appeared the last time slot. Then, vehicles are committed according to the solution provided by our algorithm. More details concerning the solving strategy are given in the Section 2.5. MAPSO population has been divided over 8 subswarms, where the migrations are performed inside a ring topology, each algorithm having a source island for receiving individuals and a destination island for sending the emigrant individuals. Another element with important consequences over the algorithm is the asynchronous migration parameterization. Frequent migrations may result in a premature convergence while distant migrations fall in the opposite case (the algorithms having independent evolutions). For our case, 5% of the population migrate at each 1000 evaluations, in an asynchronous manner (migrations occur at different times, depending on the evolution of the algorithm). The choice of asynchronous communication model is related to the fact that the speedup performance of this model is expected to be higher than synchronous models. Indeed, in the synchronous model, the evolution process is often hanging on powerful machines waiting the less powerful ones to complete their computation. On the other hand, this model is not fault tolerant as wasting a metaheuristic implies the blocking of the whole model in a volatile environment. A stochastic tournament selection strategy is being applied for selecting the emigrant individuals while the immigrant discard the worst individuals in the target population. The algorithm iterates for 5000 evaluations for each sub-problem. Then, the stopping criterion for the entire simulation is $25 \times 5000 = 125000$. We reuse the same parameters of APSO algorithm described in Section 3.7.1. Table 4.1 summarizes MAPSO parameters used for solving Kilby instances. Thirty trials of our algorithms have been considered.

Table 4.2 summarizes the average and best results found by MAPSO-DVRP and both VNS, APSO presented previously in the chapter 2, and 3 respectively. In addition, we report the results obtained by state-of-the-art metaheuristics; Ant System (AS) [Montemanni 2005b], Genetic Algorithm (GA) and Tabu Search (TS) [Hanshar 2007]. The best found solutions are highlighted into dark shaded cells, and the average results are marked in light shaded cells.

Table 4.2 shows that our algorithm is able to provide high quality solutions.

Table 4.1: Algorithm parameters for the multi-swarm metaheuristic.

Parameter type	Default value	Range
Number of swarms	8	–
Population size	100	–
Migration topology	ring	–
Migration frequency	1000 evaluations	–
Migration size	5% of population size	–
φ_1	–	0.5-1.0
φ_2	–	0.5-1.0
Stopping criterion	5000 evaluations for each sub-problem	25×5000=125000 evaluations for the whole problem

It outperforms the other metaheuristics, and gives 15 new best solutions out of the 21 Kilby's instances. Our algorithm provides also the shortest average for the traveled distance on 16 instances. The improvement provided by our algorithm on average ranges between 3.51% and 9.75% compared to the tested metaheuristics on these instances. The average of the relative error for the best results is 1.56%.

Algorithm 10 Cooperative Multi-Adaptive Particle Swarm Optimization (MAPSO) for Dynamic Optimization Problem

// Initialization
for Each swarm s_i **do**
  for Each particle j **do**
    Initialize v_j^i , $x_j^i = p_j^i$;
    Evaluate $f(p_j^i)$;
  end for
end for
repeat
  repeat
    // Test for Change
    for Each swarm s_i **do**
      //Evaluate function at swarm attractor of swarm s_i
      Evaluate $f(p_g^i)$;
      if new value is different from last iteration **then**
        for Each particle j of swarm s_i **do**
          //Evaluate each particle
          Evaluate $f(x_j^i, p_j^i)$;
          //Reuse the best solutions found previously by the particles (adaptive memory)
          x_j^i := AdjustPosition(x_j^i, p_j^i);
        end for
        //Update swarm attractor
        $p_g^i := argmin\{f(p_j^i)\}$;
      end if
    end for
    //Update velocities
    $v_j^i(t) := \omega \times v_j^i(t-1) + \varphi_1 \times r_1(p_j^i - x_j^i(t-1)) + \varphi_2 \times r_2(p_g^i - x_j^i(t-1))$;
    //Move to the new position
    $x_j^i(t) := x_j^i(t-1) + v_j^i(t)$;
    // Update Attractor
    Evaluate $f(x_j^i)$;
    if $f(x_j^i) < f(p_j^i)$ **then**
      $p_j^i := x_j^i$;
    end if
    if $f(x_j^i) < f(p_g^i)$ **then**
      $p_g^i := x_j^i$;
    end if
  until migration criterion reached
  // Communication Step
  Migration of local particles toward the neighbor swarm;
  Integration of the received particles in the population;
until certain criterion is reached

Table 4.2: Numerical results obtained by MAPS0 compared to APSO, VNS, AS, GA and TS.

Instances	Metaheuristics											
	MAPSO		APSO		VNS		AS [Montemanni 2005b]		GA [Hanshar 2007]		TS [Hanshar 2007]	
	Best	Averg.	Best	Averg.	Best	Averg.	Best	Averg.	**Best**	Averg.	Best	Averg.
c50	571.34	610.67	575.89	632.38	599.53	653.84	631.30	681.86	**570.89**	**593.42**	603.57	627.90
c75	**931.59**	**965.53**	970.45	1031.76	981.64	1040.00	1009.36	1042.39	981.57	1013.45	981.51	1013.82
c100	**953.79**	**973.01**	988.27	1051.5	1022.92	1087.18	973.26	1066.16	961.10	987.59	997.15	1047.60
c100b	**866.42**	**882.39**	924.32	964.47	866.71	942.81	944.23	1023.60	881.92	900.94	891.42	932.14
c120	**1223.49**	**1295.79**	1276.88	1457.22	1285.21	1469.24	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12
c150	**1300.43**	**1357.71**	1371.08	1470.95	1334.73	1441.37	1345.73	1455.50	1348.88	1386.93	1318.22	1401.06
c199	**1595.97**	**1646.37**	1640.40	1818.55	1679.65	1769.95	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43
f71	287.51	**296.76**	**279.52**	312.35	304.32	325.18	311.18	358.69	301.79	309.94	280.23	306.33
f134	15150.5	16193	15875.00	16645.89	15680.05	16522.18	**15135.51**	16083.56	15528.81	**15986.84**	15717.90	16582.04
tai75a	1794.38	**1849.37**	1816.07	1935.28	1806.81	1954.25	1843.08	1945.20	1782.91	1856.66	**1778.52**	1883.47
tai75b	**1396.42**	**1426.67**	1447.39	1484.73	1480.70	1560.71	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72
tai75c	1483.10	1518.65	1481.35	1664.4	1621.03	1746.07	1574.98	1653.58	1440.54	**1501.91**	**1406.27**	1527.72
tai75d	**1391.99**	**1413.83**	1414.28	1493.47	1446.50	1541.98	1472.35	1529.00	1399.83	1422.27	1430.83	1453.56
tai100a	**2178.86**	**2214.61**	2249.84	2370.58	2250.50	2462.50	2375.92	2428.38	2232.71	2295.61	2208.85	2310.37
tai100b	**2140.57**	2218.58	2238.42	2385.54	2169.10	2319.72	2283.97	2347.90	2147.70	**2215.93**	2219.28	2330.52
tai100c	**1490.4**	**1550.63**	1532.56	1627.32	1490.58	1557.81	1562.30	1655.91	1541.28	1622.66	1515.10	1604.18
tai100d	1838.75	1928.69	1955.06	2123.9	1969.94	2100.38	2008.13	2060.72	**1834.60**	**1912.43**	1881.91	2026.76
tai150a	**3273.24**	**3389.97**	3400.33	3612.79	3479.44	3680.35	3644.78	3840.18	3328.85	3501.83	3488.02	3598.69
tai150b	**2861.91**	**2956.84**	3013.99	3232.11	2934.86	3089.57	3166.88	3327.47	2933.40	3115.39	3109.23	3215.32
tai150c	**2512.01**	**2671.35**	2714.34	2875.93	2674.29	2928.77	2811.48	3016.14	2612.68	2743.55	2666.28	2913.67
tai150d	**2861.46**	**2989.24**	3025.43	3347.6	2954.64	3147.38	3058.87	3203.75	2950.61	3045.16	2950.83	3111.43
Total	**48104.13**	**50349.66**	50190.87	53538.72	50033.15	53341.24	50876.23	53794.02	49202.73	51089.37	49987.8	52725.85

4.6.2 Study on Varying the Number of Sub-Populations

We propose here to study the impact of the number of the interacting sup-populations on the optimization process. For this purpose, from a fixed number of individuals in the whole population, we vary the number of sub-populations involved in the search from 2 to 8 sub-populations. Therefore, three algorithms are proposed; $MAPSO_2$ with two sub-populations, $MAPSO_4$ with four sub-populations, and $MAPSO_8$ with eight sub-populations. In the same manner as in the previous chapters, our analysis covers a large set of instances that we have defined for the vehicle routing problems. This set is called *k-series* and includes three instances which are *k100*, *k250*, and *k500*. The whole population size is 100 particles, which is divided according the number of sup-populations in the proposed approaches. The stopping criterion is fixed to 600 per time slot, thus, $600 \times 25 = 15000$ evaluations for $k100$ and $k250$ instances, and $1200 \times 25 = 30000$ evaluations for the $k500$ instance. The solving algorithm runs over 25 time slots. Each slot corresponds to a static VRP-like instance. We have performed 30 independent trials of each experiment. The results are shown in Table 4.3, which indicates the best achieved fitness, the average and the standard deviation. Dark shaded cells correspond to the best found solutions, while best average results are in light shaded cells. In order to be able to compare our results accurately, we have also performed statistical significance tests. Kruskal-Wallis test has been applied to compare the medians of the algorithms [Cohen 1995]. As a result, all our experiments have a confidence level of 95 % (p-value ≤ 0.05). Table 4.4 is marked with "+" sign if there are statistical differences between a certain pair of algorithms, and with a "−" sign otherwise.

Table 4.3: Solutions obtained by MAPSO on dynamic *k-series* instances.

Algorithm	Instance	Algorithm	Solutions		
			Best	Avreg.	Dev
MAPSO	*k100*	APSO	1819.01	1871.25	29.55
		$MAPSO_2$	1786.61	1885.26	51.67
		$MAPSO_4$	1762.54	**1866.20**	53.15
		$MAPSO_8$	**1755.84**	1884.16	52.8
	k250	APSO	7658.27	8194.08	99.82
		$MAPSO_2$	7126.92	7356.59	125.6
		$MAPSO_4$	6855.76	7078.03	63.15
		$MAPSO_8$	**6756.19**	**7039.29**	115.75
	k500	APSO	26347.8	27592.34	383.07
		$MAPSO_2$	24200.00	25830.51	926.49
		$MAPSO_4$	23377.40	24018.08	343.77
		$MAPSO_8$	**23189.00**	**23888.58**	465.77

Table 4.4: Statistical results of comparing our algorithms with a multi-comparer test.

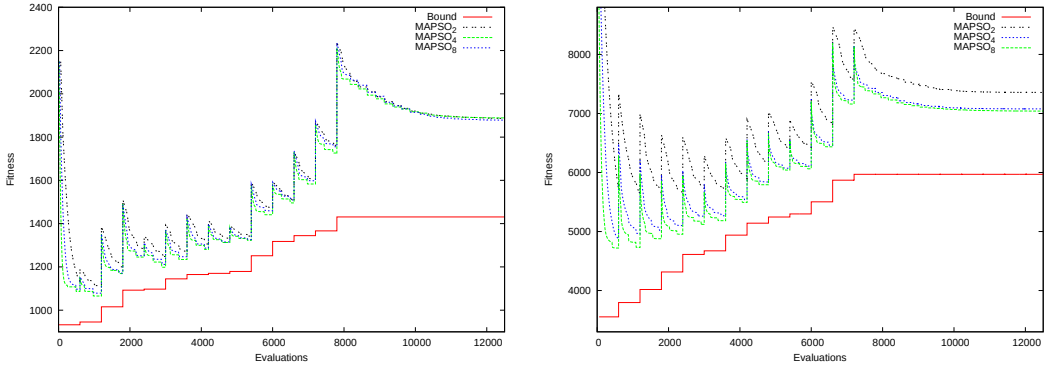
Instances	Algorithms	$MASPO_2$	$MASPO_4$	$MASPO_8$
k100	$MASPO_2$	-	-	-
	$MASPO_4$	-	-	-
	$MASPO_8$	-	-	-
k250	$MASPO_2$	-	+	+
	$MASPO_4$	+	-	-
	$MASPO_8$	+	-	-
k500	$MASPO_2$	-	+	+
	$MASPO_4$	+	-	-
	$MASPO_8$	+	-	-

From the Table 4.3, we can see that the parallel multi-swarm MAPSO provides better results than the APSO with a single population on the three treated instances. The improvement on the fitness value for the best obtained solutions ranges between 3.72 % and 14.35 %.
For the three instances, $MAPSO_8$ provides the best results in terms of minimizing the traveled distance of the vehicle fleet. Concerning the average distance, $MASPO_4$ gives better results on *k100* than $MAPSO_2$ and $MAPSO_8$. Moreover, $MASPO_8$ gives better results on *k25O* and *k500* than the other algorithms for this metric. We can explain the performance of the multi-population approach by the fact that it offers more ability for the algorithm to search in o different regions the same moment. Thereby, when new customer orders appear leading to the move of the optimum towards a new position in the search space, with several populations we have more chance to follow its movements and to reach it after some iterations.

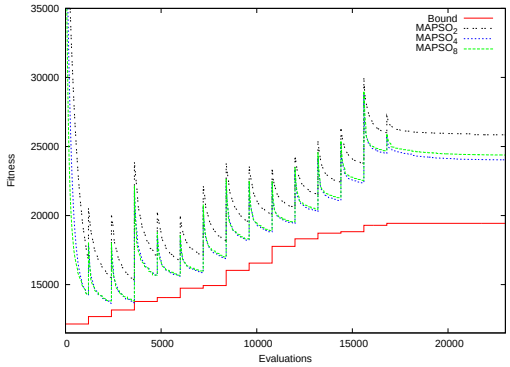
4.6.3 Dynamic Performance Assesment

Conventional Instances. In order to assess the dynamic performance of our MAPSO, we have computed the accuracy of the solutions obtained by our algorithm on each instance at the end of the optimization process. Table 4.5 shows the accuracy of MAPSO and the algorithms described in the above section. It reports the best obtained distances and the bounds Min_F^T (best known solutions) found by an off-line algorithm which had access to the entire instance (all customers are static), including dynamic requests beforehand. Most of them are not feasible solutions for the DVRP, but play a role of bounds. These solutions can be found in literature[5] over the 21 Kilby's instances. From Table 4.5, we see that MAPSO has the best accuracy on 18 instances and the best average on the set of treated instances. The accuracy

[5] `http://neo.lcc.uma.es/radi-aeb/WebVRP/`



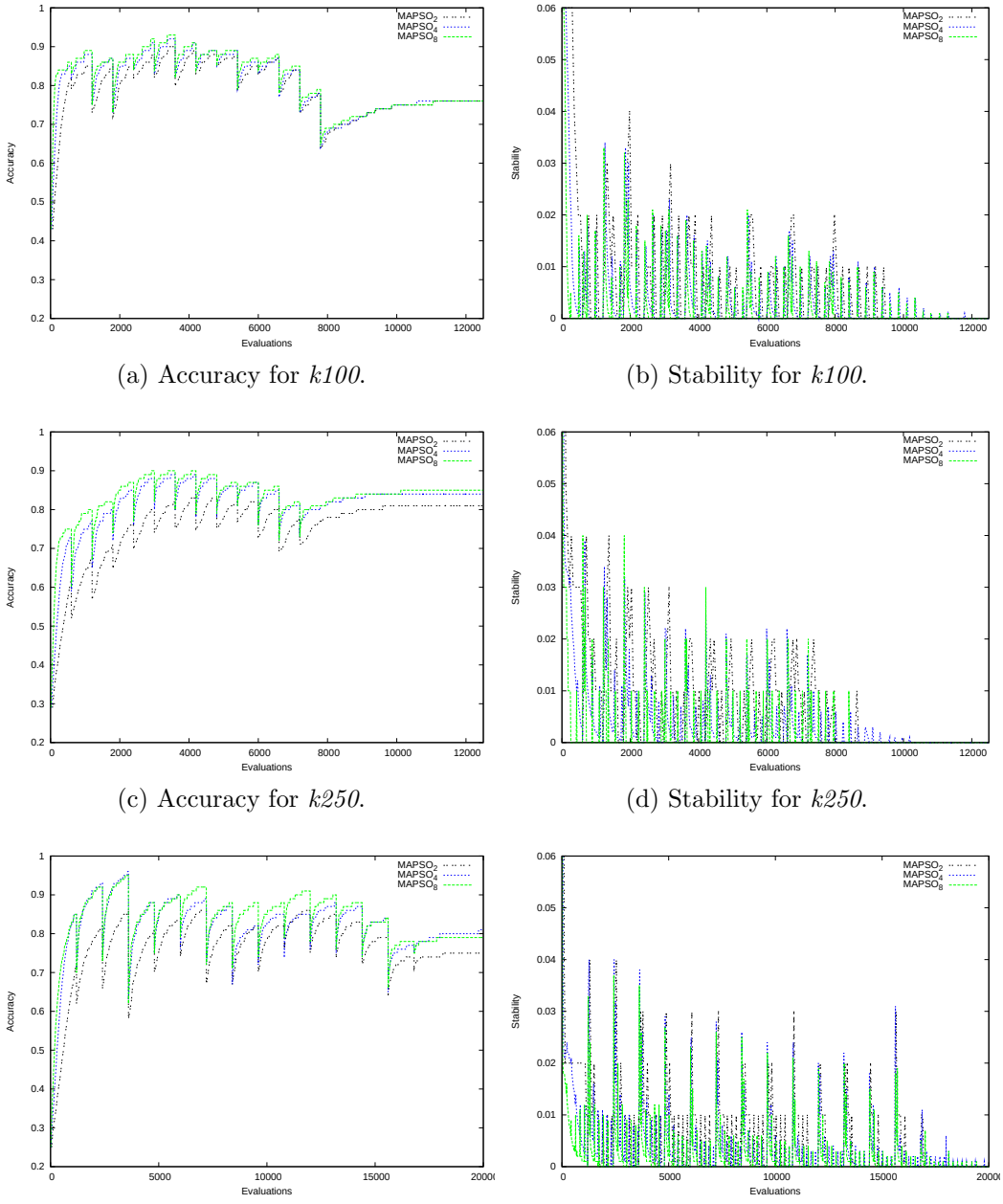
(a) *k100* instance. (b) *k250* instance.



(c) *k500* instance.

Figure 4.5: The evolution of each algorithm's mean trace for each instance; each of them shows also the optimum value for each time slice as obtained by running our algorithms over the static subproblems.

reaches for some instances as $tai75d$ a value of 0.98, which is a high achievement of our algorithm. The accuracy average is equal to 0.89 (being of 1.0 is the ideal) which demonstrates again that our multi-population algorithm outperforms the state-of-the-art metaheuristics and is able to produce high quality solutions on the conventional dynamic benchmarks.



(a) Accuracy for *k100*.

(b) Stability for *k100*.

(c) Accuracy for *k250*.

(d) Stability for *k250*.

(e) Accuracy for *k500*.

(f) Stability for *k500*.

Figure 4.6: Evolution of accuracy and stability across time slices for each instance.

Table 4.5: Accuracy of different metaheuristics on the Kilby's instances.

Instance	Min_F^T	Metaheuristics									
		APSO		MAPSO		AS [Montemanni 2005b]		GA [Hanshar 2007]		TS [Hanshar 2007]	
		Best	Accu.	Best	Accu.	Best	Accu.	Best	Accu.	Best	Accu.
c50	521	575.89	0.90	571.34	0.91	631.3	0.83	570.89	0.91	603.57	0.86
c75	832	970.45	0.86	931.59	0.89	1009.36	0.82	981.57	0.85	981.51	0.85
c100	817	988.27	0.83	953.79	0.86	973.26	0.84	961.1	0.85	997.15	0.82
c100b	820	924.32	0.89	866.42	0.95	944.23	0.87	881.92	0.93	891.42	0.92
c120	1042.11	1276.88	0.82	1223.49	0.85	1416.45	0.74	1303.59	0.8	1331.22	0.78
c150	1028.42	1371.08	0.75	1300.43	0.79	1345.73	0.76	1348.88	0.76	1318.22	0.78
c199	1291.45	1640.4	0.79	1595.97	0.81	1771.04	0.73	1654.51	0.78	1750.09	0.74
f71	237	279.52	0.85	287.51	0.82	311.18	0.76	301.79	0.79	280.23	0.85
f134	11620	15875	0.73	15150.5	0.77	15135.51	0.77	15528.81	0.75	15717.9	0.74
tai75a	1618.36	1816.07	0.89	1794.38	0.90	1843.08	0.88	1782.91	0.91	1778.52	0.91
tai75b	1344.64	1447.39	0.93	1396.42	0.96	1535.43	0.88	1464.56	0.92	1461.37	0.92
tai75c	1291.01	1481.35	0.87	1483.1	0.87	1574.98	0.82	1440.54	0.90	1406.27	0.92
tai75d	1365.42	1414.28	0.97	1391.99	0.98	1472.35	0.93	1399.83	0.98	1430.83	0.95
tai100a	2041.33	2249.84	0.91	2178.86	0.94	2375.92	0.86	2232.71	0.91	2208.85	0.92
tai100b	1940.61	2238.42	0.87	2140.57	0.91	2283.97	0.85	2147.7	0.90	2219.28	0.87
tai100c	1406.2	1532.56	0.92	1490.4	0.94	1562.3	0.9	1541.28	0.91	1515.1	0.93
tai100d	1581.25	1955.06	0.81	1838.75	0.86	2008.13	0.79	1834.6	0.86	1881.91	0.84
tai150a	3055.23	3400.33	0.90	3273.24	0.93	3644.78	0.84	3328.85	0.92	3488.02	0.88
tai150b	2656.47	3013.99	0.88	2861.91	0.93	3166.88	0.84	2933.4	0.91	3109.23	0.85
tai150c	2341.84	2714.34	0.86	2512.01	0.93	2811.48	0.83	2612.68	0.90	2666.28	0.88
tai150d	2645.39	3025.43	0.87	2861.46	0.92	3058.87	0.86	2950.61	0.90	2950.83	0.90
Average	1976.03	2390.04	0.86	2290.67	0.89	2422.68	0.83	2342.99	0.87	2380.37	0.86

Large Scale Instances. In the same way that the conventional instances, we assess the dynamic performances of MAPSO on *k-series* instances.

Tables 4.6 and 4.7 show the average of the accuracy and stability of the algorithms for the three *k-series* instances on the different time slices of the working day T. These results are plotted in Figure 4.6. The accuracy results confirm numerically what we already explained in Section 4.6, the size of the instance affects differently the performance of our algorithms. In instance *k100*, at the end of the working day, the accuracy of the algorithms is quite close to each other. However, when the size of the instance increases, a gap appears into the performances of the algorithms. Indeed, the MASPO_2 has the worst accuracy on the *k250* and *k500* instances, comparatively to the MAPSO_4 and MAPSO_8. The highest accuracy levels over the simulation were obtained by MAPSO_8 for the treated instances. With respect to stability, the algorithms vary between two phases; unstable at the beginning of the optimization process, and quite stable at the end. At each change in the environment (arrival of new orders), the algorithms are a bit destabilized. This is translated by the peaks at the beginning of each time slice as shown in the Figure 4.6. In terms of number of sub-populations, it may be noted that more the number of sub-populations increases, more the algorithm is stable. Table 4.7 shows that MAPSO_8 is the most stable algorithm. Given that MAPSO_8 is the most reactive to the changing comparatively to the other variants (see Section 4.6.2), it is also more robust and stable. The algorithms are relatively stable during the optimization process for a measure which ranges between 0 and 1.

4.6.4 Parallel Performance Assessment

In parallel algorithms, the main performance measures are speedup and efficiency. They have been introduced to evaluate the scalability of algorithms. The scalability of a parallel algorithm measures its ability to achieve performance proportional to the number of processors. The speedup S_N is defined as the time T_1 it takes to complete a program with one processor divided by the time T_N it takes to complete the same program with N processors [Cung 2002, Alba 2005, Talbi 2009].

$$S_N = \frac{T_1}{T_N} \tag{4.1}$$

The speedup is defined as the gain achieved by parallelizing a program. The larger the speedup, the greater is the gain. If $S_N > N$ (respectively $S_N < N$), a superlinear (respectively linear) speedup is obtained. In the case $S_N < N$ the speedup is said sublinear. The sublinear speedup is the most common.

Table 4.6: Accuracy of MAPSO on the dynamic *k-series* instances over different time steps.

Instance	T_{step}	Accuracy			
		APSO	$MASPO_2$	$MASPO_4$	$MASPO_8$
k100	0	0.87	0.82	0.85	0.86
	5	0.88	0.90	0.92	0.93
	10	0.86	0.87	0.87	0.88
	15	0.76	0.74	0.74	0.74
	20	0.76	0.76	0.76	0.76
	25	0.76	0.76	0.76	0.76
	Avg	0.81	0.81	0.82	0.82
k250	0	0.61	0.62	0.73	0.75
	5	0.75	0.82	0.89	0.9
	10	0.75	0.69	0.85	0.86
	15	0.72	0.81	0.84	0.85
	20	0.72	0.81	0.84	0.85
	25	0.72	0.81	0.84	0.85
	Avg	0.71	0.77	0.82	0.84
k500	0	0.77	0.72	0.85	0.90
	5	0.80	0.86	0.89	0.96
	10	0.79	0.85	0.88	0.91
	15	0.70	0.75	0.80	0.82
	20	0.70	0.75	0.81	0.82
	25	0.70	0.75	0.81	0.82
	Avg	0.73	0.76	0.82	0.85

This is due to the overhead of communication and synchronization costs. The efficiency E_N using N processors is defined as the speedup S_N divided by the number of processors N.

$$E_N = \frac{S_N}{N} \tag{4.2}$$

It defines how well are N processors used when the program is computed in parallel. An efficiency of one means that all of the processors are being fully used all the time.

To compute the speedup, we compare the algorithm both in sequential and in parallel architecture. The speedup and efficiency provide by the parallelization of each algorithm are shown in the Table 4.8. The trace of the speedup for the three instances is shown in the Figure 4.7. It seems that the speedup performance is quite similar on the three instances. The gain achieved by parallelizing these algorithms is high, and it slightly moves away from the linear speedup as the number of CPUs increases due to the overhead of communication between nodes.

In addition, the performance on the instance *k500* is less than the *k100* and *k250* instances. Since the number of evaluations allowed on the instance *k500* for each slice is 1200 evaluations which is two times more than the other instances (600 evaluations) (see Table 2.3), this leads to decrease the speedup performance in consequence of the cost of communication due to the migration

Table 4.7: Stability of MAPSO on the dynamic *k-series* instances over different time steps.

Instance	T_{step}	Stability			
		APSO	$MASPO_2$	$MASPO_4$	$MASPO_8$
k100	0	0.532	0.43	0.434	0.434
	5	0.003	0.02	0.017	0.017
	10	0.004	0.01	0.009	0.009
	15	0.001	0.01	0.000	0.000
	20	0.00	0.01	0.000	0.000
	25	0.00	0.000	0.000	0.000
	Avg	0.09	0.01	0.004	0.002
k250	0	0.33	0.29	0.292	0.29
	5	0.015	0.03	0.022	0.02
	10	0.019	0.02	0.022	0.000
	15	0.000	0.000	0.000	0.000
	20	0.000	0.000	0.000	0.000
	25	0.000	0.000	0.000	0.000
	Avg	0.061	0.02	0.004	0.002
k500	0	0.277	0.25	0.25	0.248
	5	0.021	0.03	0.024	0.023
	10	0.019	0.02	0.020	0.019
	15	0.021	0.001	0.006	0.003
	20	0.000	0.000	0.000	0.000
	25	0.00	0.000	0.000	0.000
	Avg	0.048	0.004	0.001	0.001

of particles between swarms.

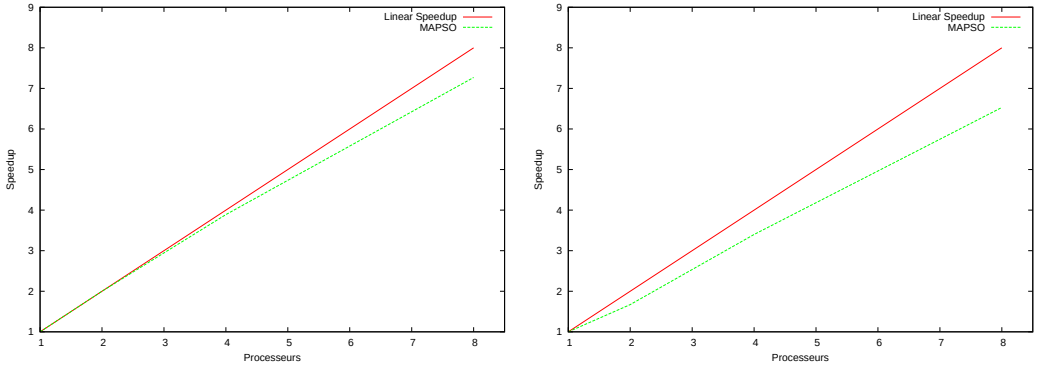
The same performances are replicated for the efficiency measure as it is shown in the Figure 4.8.

Table 4.8: Speedup and efficiency for MAPSO based algorithms.

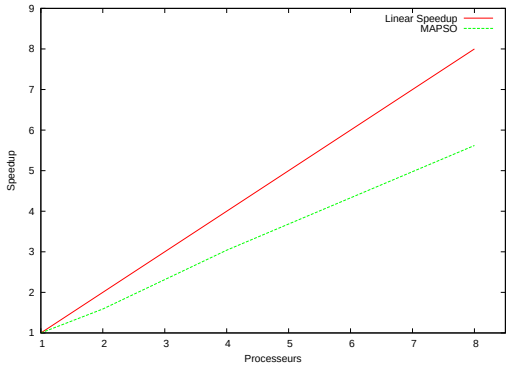
Instance	Algorithm	Speedup	Efficiency
k100	$MAPSO_2$	1.98	0.99
	$MAPSO_4$	3.89	0.97
	$MAPSO_8$	7.27	0.91
k250	$MAPSO_2$	1.74	0.87
	$MAPSO_4$	3.43	0.86
	$MAPSO_8$	6.53	0.82
k500	$MAPSO_2$	1.59	0.80
	$MAPSO_4$	3.04	0.76
	$MAPSO_8$	5.62	0.70

4.7 Conclusion

A parallel multi-swarm approach named Multi-Adaptive Particle Swarm (MAPSO) has been proposed in this chapter. Its principle is to divide the pop-



(a) *k100* instance. (b) *k250* instance.

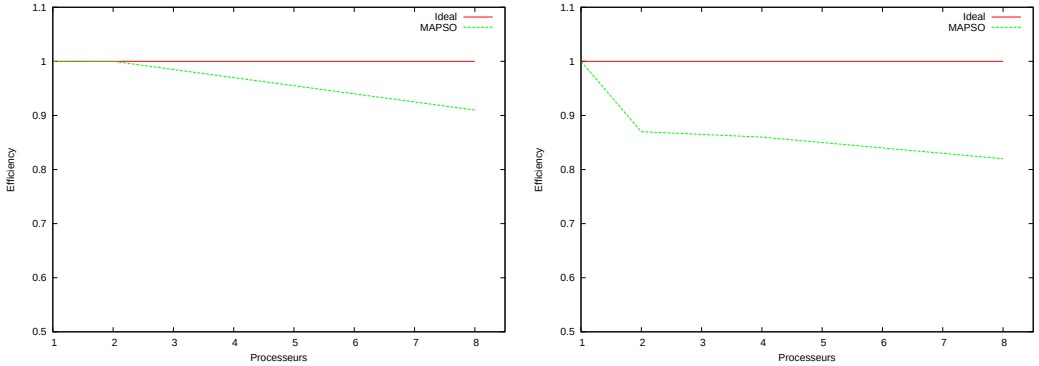


(c) *k500* instance.

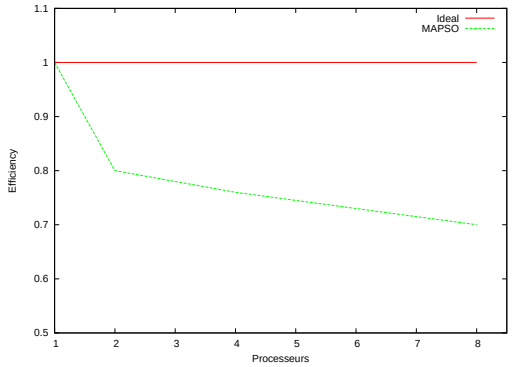
Figure 4.7: The speedup of the algorithms on each instance.

ulation into several sub-populations that evolve in parallel and which track the shifting optimum throughout the time. The aim is to position each of those subswarms on different promising local optima of the search space. However, simply breaking up the neighborhoods and dividing up the global swarm into a number of independent swarms is unlikely to be effective since the swarms would not interact (the dynamics governing the position and velocity updates of a particle in a particular swarm are specified by parameters belonging to that swarm only).

Therefore, we suggested to cooperate the swarms by exchanging information related to the best positions (local optima) found by their particles. Indeed, the dynamic change may cause the optima to be in the neighborhood of an old solution more often. The localization of several swarms on different optima allows to react quickly to the changing since the swarms are already located in the neighborhood of the new optima. Hence, we take the advantage of using the information gathered in the past and introduce more diversification in the search towards several cooperative swarms. MAPSO approach



(a) *k100* instance. (b) *k250* instance.



(c) *k500* instance.

Figure 4.8: The efficiency of the algorithms on each instance.

has been implemented on parallel architecture by using ParadisEO software framework.

For the experimental part, MAPSO gives better solution quality than metaheuristics taken from literature. A study on varying a number of sub-populations has been done, and showed that the performances increase with the number of sub-swarms which maintain population diversity through the search space. Dynamic and parallel performances were assessed over the different variants of the algorithm.

The next chapter will examine some modifications that can be brought to metaheuristics and which allow them not only find high quality solutions, but solutions that are robust and flexible as well.

Chapter 5

Flexibility and Robustness in Dynamic Vehicle Routing

Contents

5.1 Introduction

Recent approaches have mainly focused on maintaining the population diversity as a warrant for the ability of tracking the optimum. However, it could be also worthwhile to anticipate changes of the environment by explicitly searching for solutions which maintain their robustness and flexibility. Flexible solutions are those that can be easily adapted to account for changes in the environment.

As we have seen in previous chapters that adaptation to changes may be necessary, it should be worthwhile to anticipate these changes and to explicitly search for solutions that, not only have high quality, but that allow the adaptation of a high quality solutions after the environment has changed.

Although this is a valid approach to all dynamic optimization problems, it seems particularly important for optimization problems where the solution is gradually implemented over time, and thus, some part of the solution is permanently fixed between changes. In this case, it is not necessarily possible to switch from one optimal solution to the next optimal solution after the environment has changed. The set of solutions still available after a change depends on the previous selected solution.

For the example of Dynamic Vehicle Routing Problem, we argue that it is useful to anticipate the forthcoming arrival of new orders. Since we know that only the front part of the routing (routes) will actually be implemented, while the remainder will have to be re-routed after the arrival of a new requests in order to insert them in the existing routes.
Our aim here is to explicitly search for routes that are flexible enough to allow easy adaptation after a new order has arrived. For that we propose a measure of flexibility, and show that better solutions can be obtained when the "planning horizon" for all subproblems is modified to take the flexibility of solutions into account. The main idea of guiding the search towards solutions that are "well-prepared" for changes in the environment is the search for robust solutions, i.e. solutions that show a good average performance under all possible future scenarios. We suggest here to adjust the planning horizon in order to build solutions that might anticipate the forthcoming arrival of new orders. The underlying idea consists in gradually increasing the period covered by a vehicle routing plan over the day. This leads to allow a large number of committed vehicle at the beginning of the working day even if the orders do not require this size of vehicle fleet. The adjustment warrants flexible insertion of new incoming orders into the existing routes throughout the day. Therefore, we anticipate the future needs of our customers. The chapter is structured as follows: In Section 5.2 we report some developing works on robustness and flexibility. Section 5.3 presents our flexible strategy for solving DVRP, while Section 5.4 exposes the implementation of this strategy for Flexible Variable Neighborhood Search (FVNS). Section 5.5 reports experimental results of our algorithm on a classical set of benchmarks and assesses the dynamic performance indicator of our algorithm. Finally, we conclude this chapter with some highlights that can be the subject of future researches.

5.2 Background

Many works have approached the study of flexibility in optimization problems, although the point of view on what flexibility means differs greatly between them. Intuitively, it can be defined as the relative ease with which a solution can be adapted to the requirements of changing problem data [Sörensen 2003]. In general, flexible solutions are especially desirable in dynamic optimization problems since the environment is expected to change even in the problem definition itself. Although flexibility is sometimes used to refer to algorithms which are able to solve different types of problems [Jans 2007], we stick to the concept of flexibility as the ability to manage the changes that occur over the time on a dynamic problem. Most works which approached flexibility

in VRP addressed the flexibility of vehicle schedules [Hashimoto 2006] or by allowing soft time window and soft traveling time constraints [Afsar 2010], rather than the flexibility of solutions. Robustness and flexibility are often used as synonyms. A study on robustness and flexibility for the VRP with Stochastic Demands is provided in [Sörensen 2009]; in this case, a robustness/flexibility evaluation function is used to evaluate a set of scenarios for each solution. A recourse procedure is introduced when flexible solutions are needed and penalty functions is introduced to penalize violation of constraints. Scheffermann *et al.* [Scheffermann 2009] present and compare algorithms for creating robust solutions to the vehicle routing problem with time-windows (VRPTW) in which travel times are uncertain. They referee to feasibility-robustness of solutions which stay feasible in uncertain environments or become only slightly infeasible by proposing a model which penalize the delays (missing a time-window) for a given service plan for a set of customers. Flexibility has been thoroughly studied in the Dynamic Scheduling Problem domain [Branke 2005a, Snoek 2001]. Branke and Mattfeld [Branke 2005a] worked on anticipation in dynamic scheduling problems. They suggest the incorporation of a criterion into the fitness function which focuses on the early utilization of machine capacity. They showed that flexibility can be gained by avoiding early idle-times and thereby penalize them.

5.3 Flexible Solving Strategy

Given that the main feature in the DVRP is the dynamic arrival of orders, we consider that flexibility is largely determined by the maximum length of routes, which depends ultimately on the time window of the depot. To preserve flexibility, we propose to construct initial solutions being aware about the potential arrival of new orders; in order to do so, we propose to dynamically adjust the length of the working day, making it smaller at the beginning of the optimization and letting it increases until the value defined by the problem instance as the simulation takes place.

In this way, we expect to get solutions with a larger number of shorter routes at the beginning of the simulation time.

If there are more routes available and they are not built to use the whole working day length, it will be easier to place new customers in a good position. We define $T'(t)$ as the function which modifies the length of the working day:

$$T'(t) = \alpha \cdot T + (1 - \alpha) \cdot \frac{cur_{ts} \cdot T_{ts}}{dod} \tag{5.1}$$

where T is the length of the working day defined in the problem instance, cur_{ts} is the current time slot (period), T_{ts} is the length of a time slot, and dod

is the degree of dynamism of the problem. The parameter α ranges in $[0, 1]$, and it is used to determine the initial value of $T_{effective}$ at the beginning of the simulation time.

The parameter α controls the variation of the modified working day with respect to the initial working day. If α is equal to 1, T' is equal to the initial working day, when alpha is equal to 0, T' takes its smallest value.

The degree of dynamism is included in order to determine the incremental increasing in T': if the problem is very dynamic (*dod* is close to 1.0), T' increases slowly during the whole simulation; if *dod* is closer to 0.0 (static problem), T' increases faster in order to reach soon the original T. The effective working day length $T_{effective}(t)$ is a modification of the working day T which is used by the algorithm at each time step t. It takes into account that T is the maximum allowed length, which means that the routes designed by the algorithm can never exceed this length in order to be feasible.

$$T_{effective}(t) = \min(T, T'(t)) \tag{5.2}$$

This strategy is independent of the optimization algorithm since it consists in the relaxation of a problem constraint. This means that it can be adopted with little effort by any metaheuristics as long as they are able to manage constraints. An example of how $T_{effective}$ changes during the simulation is shown in Figure 5.1.

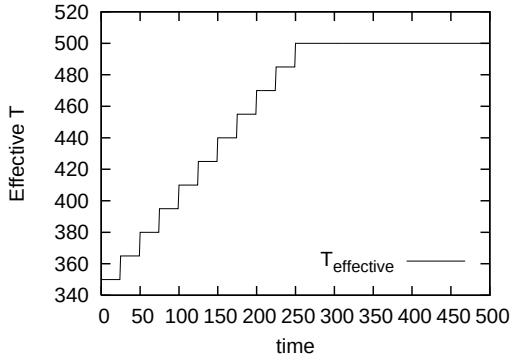


Figure 5.1: $T_{effective}$ changes with $\alpha = 0.7$, $T = 500$, $T_{ts} = 25$, and $dod = 0.5$.

5.4 Flexible VNS for DVRP

We propose Flexible Variable Neighborhood Search algorithm (FVNS) for DVRP which follows our flexibility strategy. As presented in the Chapter 2, VNS is a well-known trajectory-based metaheuristic proposed by Hansen and

Mladenović [Hansen 1999]. In order to adapt VNS for a particular problem, it is necessary to define the set of neighborhood structures and to establish the local search procedure that is applied to the solutions. Both our neighborhoods and the local search are related to specific operators of the VRP. We have proposed four different neighborhoods $\mathcal{N}_k(s)$:

- $\mathcal{N}_1(s)$ is the set of solutions which results of swapping any two customers in s,
- $\mathcal{N}_2(s)$ is the set which results of inserting a given customer into any position in s,
- $\mathcal{N}_3(s)$ results of applying 2-Opt [Lin 1965] to any subroute of s, and
- $\mathcal{N}_4(s)$ is the result of using 2-Opt* [Potvin 1995] in any two subroutes of s.

These neighborhoods allow the algorithm to escape from local optima, as constraints are not enforced at this stage. The local search consists in consecutively combining four local search operators: λ-exchange with $(1, 1)$ moves, λ-exchange with $(1, 0)$ moves, 2-Opt and 2-Opt*. For each local search heuristic, all possible moves are checked and the best one is performed, i.e. the one which reduces the solution cost the most. Our local search procedures avoid reevaluating the whole solution. A repair procedure makes any new solution feasible before its evaluation. This repair procedure is necessary since the neighborhood operators can generate unfeasible solutions. Initial solutions are generated using the Savings algorithm [Clarke 1964]. In order to avoid determinism in the construction of initial solutions, we use a parameter γ to calculate the savings as $s(i, j) = d(0, i) + d(0, j) - \gamma\, d(i, j)$, where $\gamma \sim U(0, 1)$ [Yellow 1970]. The same strategy is followed to insert dynamic customers in the solution: a partial solution including only new customers is built using the Savings algorithm and these new routes are added to the current solution.

In our Flexible Variable Neighborhood Search (*FVNS*), the flexibility of the constraints is used in the construction phase (Savings heuristic), while in the local search, we consider only the modifications that do not increase the number of routes according to the constraints.

5.5 Experimental Results and Discussion

This section presents the results obtained by our algorithm. For that purpose, we use a set of standard benchmarks introduced in the Section 1.6. This set

has been proposed by Kilby [Kilby 1998] and consists in 21 instances based on those of Christofides, Fisher, and Taillard, which have been adapted to the dynamic environment. These instances range in size $[50, 199]$ and have different topologies regarding the geographic distribution of customers (clustered, uniform, and a combination of these two). In the literature, the cutoff time T_{co} is set to $0.5 \times T$. For each instance, 30 independent runs are considered. Each static subproblem runs for 5000 evaluations and 25 time slices are considered. The *FVNS* algorithm runs on Intel Core 2 Quad 2.6 GHz machines with 4 GB memory.

5.5.1 Study on Sensitivity of the Flexibility Parameter

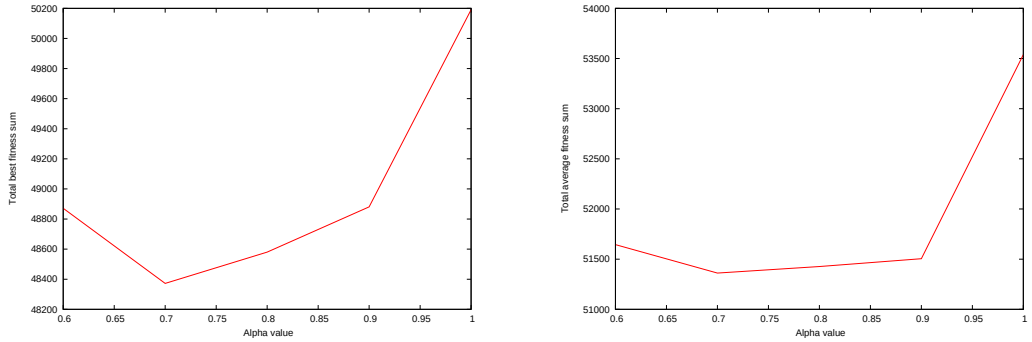
Here we analyze the influence of the α parameter for values in $[0.6, 1.0]$. If $\alpha = 1.0$, the results correspond to the canonical *VNS* with standard constraints (no modification is applied). We have not studied values smaller than 0.6 as the results for $\alpha = 0.6$ already point out that such a tight value is not a convenient strategy. Results obtained over the Kilby's instances are shown in Table 5.1 and represented graphically in Figure 5.2. The best obtained solutions are in highlighted into dark shaded cells, and the average results over 30 runs are given in light shaded cells.

The figure on the left shows the sum of the best fitness obtained for each one of the 21 instances in the benchmark, while the figure on the right represents the sum of the average fitness obtained on each instance.

From both Table 5.1 and Figure 5.2, we can see that the value $\alpha = 0.7$ is the more competitive one, followed closely by $\alpha = 0.8$ and 0.9. The results of $\alpha = 1.0$ are bad, while $\alpha = 0.6$ is too restrictive and not beneficial for the algorithm. With the variant $\alpha = 0.7$, the algorithm outperforms the other variants on 12 instances and got the best average on the whole set. As we can see in Figure 5.2, there is a correlation between the best and the average total fitness; in both cases, the best performance is achieved with $\alpha = 0.7$.

Table 5.1: Numerical results obtained by *FVNS* with different values of α.

Instances	α value											
	0.6		0.7		0.8		0.9		1.0	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
c50	596.97	647.33	**591.69**	629.61	598.10	631.01	610.54	637.80	599.53	653.84
c75	959.16	1010.37	969.45	1024.69	**949.01**	***1009.91***	983.20	1015.60	981.64	1040.00
c100	965.77	1015.56	**943.92**	1008.88	964.93	1005.16	964.50	***1000.99***	1022.92	1087.18
c100b	885.54	931.03	880.84	915.52	878.11	***914.59***	879.85	918.18	**866.71**	942.81
c120	1214.07	1405.08	**1207.51**	***1385.56***	1251.89	1386.19	1286.47	1411.92	1285.21	1469.24
c150	1287.23	1354.17	**1275.54**	***1349.71***	1304.04	1353.81	1287.89	1356.75	1334.73	1441.37
c199	1579.95	1635.91	**1556.43**	1639.59	1579.31	***1633.96***	1567.63	1636.09	1679.65	1769.95
f71	277.75	305.12	**272.65**	***292.68***	285.19	296.25	276.42	297.26	304.32	325.18
f134	15261.62	16063.06	**15104.51**	***16038.25***	15261.62	16080.62	15364.47	16081.81	15680.05	16522.18
tai75a	1796.63	1882.95	**1776.60**	***1879.12***	1791.69	1914.41	1784.09	1902.96	1806.81	1954.25
tai75b	1458.06	***1492.69***	**1455.13**	1500.15	**1455.13**	1524.34	1467.80	1524.96	1480.70	1560.71
tai75c	1539.64	1704.26	1520.93	1694.90	**1491.63**	1705.82	1547.95	***1676.03***	1621.03	1746.07
tai75d	1443.21	1517.69	1445.42	1517.40	**1441.97**	1522.52	1444.63	***1511.81***	1446.50	1541.98
tai100a	2235.92	2318.51	2196.27	2299.55	**2154.96**	***2272.26***	2181.70	2328.39	2250.50	2462.50
tai100b	2142.17	2290.79	2158.09	2239.06	2133.07	***2238.02***	**2125.52**	2256.58	2169.10	2319.72
tai100c	1484.27	1564.39	1498.06	1545.53	1491.39	1530.83	**1457.68**	***1530.81***	1490.58	1557.81
tai100d	**1819.41**	2042.58	1875.64	2050.78	1856.70	2015.21	1847.14	***2011.44***	1969.94	2100.38
tai150a	3470.69	3583.27	**3282.54**	3573.59	3310.14	3574.29	3361.69	***3560.64***	3479.44	3680.35
tai150b	2886.00	3023.82	**2870.77**	3004.32	2899.38	***3002.41***	2924.50	3028.77	2934.86	3089.57
tai150c	2617.86	2758.57	2582.93	***2701.60***	2549.11	2736.78	**2539.06**	2734.35	2674.29	2928.77
tai150d	2949.36	3098.01	**2907.27**	***3070.94***	2933.05	3077.92	2977.90	3081.63	2954.64	3147.38
Total	48871.28	51645.16	**48372.19**	***51361.43***	48580.42	51426.31	48880.63	51504.77	50190.87	53538.72



(a) Total best fitness (b) Total average fitness

Figure 5.2: Influence of the α parameter on quality solutions.

5.5.2 Comparison with State-of-the-Art Metaheurisitcs

Table 5.2 compares our *FVNS* ($\alpha = 0.7$) with other algorithms of the literature: Ant System (*AS*) [Montemanni 2005b], Genetic Algorithm (GA), and Tabu Search (TS), both proposed in [Hanshar 2007]. We highlight the best found solutions in shaded cells and best average results in light shaded cells. *FVNS* obtains 16 best solutions out of 21 instances, while *GA* obtained four and *TS* one best solution. *AS* obtains no best solution in this case. Concerning the average fitness, *GA* obtains the best average on 13 instances, while *FVNS* obtains best averages in 8 (*AS* and *TS* obtain none).
FVNS is also the algorithm which achieves the best total best fitness and *GA* obtains the best total average one (see last row in Table 5.2). This can be due to a higher standard deviation in the solutions provided by *FVNS*, which could be expected from a single-solution metaheuristic. The strength of the approach is found when solving the large scale instances, which points a good scalability of our algorithm.
Concerning the execution time, by normalizing this time according to the performance of machine testbed [Gee 2010] as shown in Table 5.3, FVNS spends less time than the AG, TS and it is comparable in time with AS.

5.5.3 Flexibility vs Multi-Populations

In this section, we compare our flexibility enhanced metaheuristic with the Multi-Swarm Adaptive PSO (*MAPSO*) approach that we have presented in the previous chapter, and which has shown to be a cutting-edge strategy for this problem (see Table 5.4). *MAPSO* improves the results of *AS*, *GA*, and *TS* on 15 out of 21 instances by using a parallel algorithm with 8 islands,

Table 5.2: Numerical results obtained by *FVNS* compared to *AS*, *GA*, and *TS*.

Instances	Metaheuristics							
	FVNS		*AS* [Montemanni 2005b]		*GA* [Hanshar 2007]		*TS* [Hanshar 2007]	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
c50	591.69	629.61	631.30	681.86	**570.89**	***593.42***	603.57	627.90
c75	**969.45**	1024.69	1009.36	1042.39	981.57	***1013.45***	981.51	1013.82
c100	**943.92**	1008.88	973.26	1066.16	961.10	***987.59***	997.15	1047.60
c100b	**880.84**	915.52	944.23	1023.60	881.92	***900.94***	891.42	932.14
c120	**1207.51**	***1385.56***	1416.45	1525.15	1303.59	1390.58	1331.22	1468.12
c150	**1275.54**	***1349.71***	1345.73	1455.50	1348.88	1386.93	1318.22	1401.06
c199	**1556.43**	***1639.59***	1771.04	1844.82	1654.51	1758.51	1750.09	1783.43
f71	**272.65**	***292.68***	311.18	358.69	301.79	309.94	280.23	306.33
f134	**15104.51**	16038.25	15135.51	16083.56	15528.81	***15986.84***	15717.90	16582.04
tai75a	**1776.60**	1879.12	1843.08	1945.20	1782.91	***1856.66***	1778.52	1883.47
tai75b	**1455.13**	***1500.15***	1535.43	1704.06	1464.56	1527.77	1461.37	1587.72
tai75c	1520.93	1694.90	1574.98	1653.58	1440.54	***1501.91***	**1406.27**	1527.72
tai75d	1445.42	1517.40	1472.35	1529.00	**1399.83**	***1422.27***	1430.83	1453.56
tai100a	**2196.27**	2299.55	2375.92	2428.38	2232.71	***2295.61***	2208.85	2310.37
tai100b	2158.09	2239.06	2283.97	2347.90	**2147.70**	***2215.93***	2219.28	2330.52
tai100c	**1498.06**	***1545.53***	1562.30	1655.91	1541.28	1622.66	1515.10	1604.18
tai100d	1875.64	2050.78	2008.13	2060.72	**1834.60**	***1912.43***	1881.91	2026.76
tai150a	**3282.54**	3573.59	3644.78	3840.18	3328.85	***3501.83***	3488.02	3598.69
tai150b	**2870.77**	***3004.32***	3166.88	3327.47	2933.40	3115.39	3109.23	3215.32
tai150c	**2582.93**	***2701.60***	2811.48	3016.14	2612.68	2743.55	2666.28	2913.67
tai150d	**2907.27**	3070.94	3058.87	3203.75	2950.61	***3045.16***	2950.83	3111.43
Total	**48372.19**	51361.43	50876.23	53794.02	49202.73	***51089.37***	49987.8	52725.85

which assigns 5000 evaluations to each sub-problem (see Table 4.2). *FVNS* uses the same criterion for each time slice. The best and average results are provided. In total, *FVNS* obtains 7 best solutions, while *MAPSO* got 14 out of 21 instances. *FVNS* is particularly suitable for Christofides' bigger instances (c100 to c199) as well as Fisher's instances, while *MAPSO* performs better in smaller Christofides' instances and most Taillard's instances. According to these results and the topology of customer localization in the different instances, *FVNS* is confirmed to be especially suitable for problem instances where customers are located following a uniform or clustered distribution, while *MAPSO* is better for those with clustered and semi-clustered customers. We can explain this by the fact that having a large number of routes early in the planning horizon with our flexible strategy promotes the insertion of future customer requests especially in instances where customers are distributed uniformly in the service region. Thus, it ensures that a vehicle will always be available to process a new order in a sub-region throughout the day. Less detours are made for servicing customers what reduces the related cost.

Table 5.3: Execution time in minutes of FVNS compared to AS, GA and TS.

	Metaheuristics			
Instance	FVNS	AS [Montemanni 2005b]	GA [Hanshar 2007]	TS [Hanshar 2007]
c50	0.71			
c75	1.26			
c100	2.91			
c100b	1.76			
c120	4.49			
c150	7.65			
c199	11.68			
f71	1.65			
f134	1.65			
tai75a	1.88			
tai75b	0.98	(25)	(12.5)	(12.5)
tai75c	1.3	1'per slot	30"per slot	30"per slot
tai75d	1.11		0.5 * 25	0.5 * 25
tai100a	2.55			
tai100b	2.29			
tai100c	1.91			
tai100d	2.8			
tai150a	7.71			
tai150b	6.48			
tai150c	5.33			
tai150d	5.09			
Total time	72.77	525	262.5	262.5
Normalized time	72.77	**62.08**	73.5	73.5

5.5.4 Dynamic Performance Assessment

In this section we assess our *FVNS* in terms of accuracy of the obtained solutions described in Section 5.5.3. The accuracy indicates the approximation between the solutions obtained by an algorithm and optimal solutions in cases where we consider a static environment (all customer orders are known before than the optimization takes place). The accuracy ranges in the interval [0-1]. An algorithm with accuracy being close to 1 is the ideal in terms of performance. Table 5.5 shows the accuracy of *FVNS* and our MAPSO with other metaheuristics proposed for DVRP in literature. We can see that the accuracy of FVNS ranges between $[0.77-0.94]$ and with an average of 0.86, and is very competitive comparing with the remain metaheuristics on 10 instances. The multi-population approach MAPSO has the best average with 0.89 and outperforms the other metaheuristics on 14 instances.

Table 5.4: Comparison between *FVNS* and *MAPSO*.

Instances	**Metaheuristics**			
	FVNS		MAPSO	
	Best	**Average**	**Best**	**Average**
c50	591.69	629.61	**571.34**	***610.67***
c75	969.45	1024.69	**931.59**	***965.53***
c100	**943.92**	1008.88	953.79	***973.01***
c100b	880.84	915.52	**866.42**	***882.39***
c120	**1207.51**	1385.56	1223.49	***1295.79***
c150	**1275.54**	***1349.71***	1300.43	1357.71
c199	**1556.43**	***1639.59***	1595.97	1646.37
f71	**272.65**	***292.68***	287.51	296.76
f134	**15104.51**	***16038.25***	15150.5	16193.00
tai75a	**1776.60**	1879.12	1794.38	***1849.37***
tai75b	1455.13	1500.15	**1396.42**	***1426.67***
tai75c	1520.93	1694.90	**1483.10**	***1518.65***
tai75d	1445.42	1517.40	**1391.99**	***1413.83***
tai100	2196.27	2299.55	**2178.86**	***2214.61***
tai100	2158.09	2239.06	**2140.57**	***2218.58***
tai100	1498.06	***1545.53***	**1490.40**	1550.63
tai100	1875.64	2050.78	**1838.75**	***1928.69***
tai150	3282.54	3573.59	**3273.24**	***3389.97***
tai150	2870.77	3004.32	**2861.91**	***2956.84***
tai150	2582.93	2701.60	**2512.01**	***2671.35***
tai150	2907.27	3070.94	**2861.46**	***2989.24***
Total	48372.19	51361.43	**48104.13**	***50349.66***

Table 5.5: Accuracy of FVNS compared with other metaheuristics on Kilby's instances.

Instance	Min_F^T	Metaheuristics									
		FVNS		MAPSO		AS [Montemanni 2005b]		GA [Hanshar 2007]		TS [Hanshar 2007]	
		Best	Accu.	Best	Accu.	Best	Accu.	Best	Accu.	Best	Accu.
c50	521	591.69	0.88	571.34	0.91	631.3	0.83	570.89	0.91	603.57	0.86
c75	832	969.45	0.86	931.59	0.89	1009.36	0.82	981.57	0.85	981.51	0.85
c100	817	943.92	0.87	953.79	0.86	973.26	0.84	961.1	0.85	997.15	0.82
c100b	820	880.84	0.93	866.42	0.95	944.23	0.87	881.92	0.93	891.42	0.92
c120	1042.11	1207.51	0.86	1223.49	0.85	1416.45	0.74	1303.59	0.80	1331.22	0.78
c150	1028.42	1275.54	0.81	1300.43	0.79	1345.73	0.76	1348.88	0.76	1318.22	0.78
c199	1291.45	1556.43	0.83	1595.97	0.81	1771.04	0.73	1654.51	0.78	1750.09	0.74
f71	237	272.65	0.87	287.51	0.82	311.18	0.76	301.79	0.79	280.23	0.85
f134	11620	15104.51	0.77	15150.5	0.77	15135.51	0.77	15528.81	0.75	15717.9	0.74
tai75a	1618.36	1776.6	0.91	1794.38	0.90	1843.08	0.88	1782.91	0.91	1778.52	0.91
tai75b	1344.64	1455.13	0.92	1396.42	0.96	1535.43	0.88	1464.56	0.92	1461.37	0.92
tai75c	1291.01	1520.93	0.85	1483.1	0.87	1574.98	0.82	1440.54	0.90	1406.27	0.92
tai75d	1365.42	1445.42	0.94	1391.99	0.98	1472.35	0.93	1399.83	0.98	1430.83	0.95
tai100a	2041.33	2196.27	0.93	2178.86	0.94	2375.92	0.86	2232.71	0.91	2208.85	0.92
tai100b	1940.61	2158.09	0.90	2140.57	0.91	2283.97	0.85	2147.7	0.90	2219.28	0.87
tai100c	1406.2	1498.06	0.94	1490.4	0.94	1562.3	0.9	1541.28	0.91	1515.1	0.93
tai100d	1581.25	1875.64	0.84	1838.75	0.86	2008.13	0.79	1834.6	0.86	1881.91	0.84
tai150a	3055.23	3282.54	0.93	3273.24	0.93	3644.78	0.84	3328.85	0.92	3488.02	0.88
tai150b	2656.47	2870.77	0.93	2861.91	0.93	3166.88	0.84	2933.4	0.91	3109.23	0.85
tai150c	2341.84	2582.93	0.91	2512.01	0.93	2811.48	0.83	2612.68	0.90	2666.28	0.88
tai150d	2645.39	2907.27	0.91	2861.46	0.92	3058.87	0.86	2950.61	0.90	2950.83	0.9
Average	1976.03	2303.44	0.86	2290.67	0.89	2422.68	0.83	2342.99	0.87	2380.37	0.86

5.6 Conclusion

In this chapter, we have stressed the role of anticipation for optimization in Dynamic Vehicle Routing Problem and have shown that it is important to search explicitly for solutions that are flexible enough to be easily adapted to changes in the environment.

We suggest a measure of flexibility and show that better solutions can be obtained when this measure is incorporated into the solving strategy of our Flexible Variable Neighborhood Search ($FVNS$).

Our approach consists in the relaxation of the standard constraints of the problem, in order to make early decisions that provide flexible solutions which could be easily adapted when changes happen in the environment (arrival of new requests).

Our empirical tests yielded excellent results and clearly demonstrated the effectiveness of our approach comparatively to the canonical version of the algorithm.

Note that the general idea of anticipation is not restricted to vehicle routing problems, although it is particularly useful for problems where a part of the solution is fixed because the decision cannot be revised later.

There remain numerous avenues for future research. The degree of anticipation could be extended, *e.g.* by incorporating prediction on the arrival time of customer orders. The problem becomes similar to a problem under uncertainty and the developed algorithms must be adapted to take into a count this issue.

Conclusions and Future Work

This thesis has focused on the application of metaheuristics to Dynamic Capacitated Vehicle Routing Problem (DCVRP). The vast amount of research on metaheuristics indicates that they have established themselves as an effective optimization tool to deal with this kind of dynamic problems.
Tackling dynamic problems entails addressing several issues related to algorithm design, performance measures, benchmarking. All this dynamism related issues have been addressed in this work.
In this thesis, different classes of metaheuristics are developed. Results of experimentation on DVRP demonstrate than the methods are effective and have a great potential for other dynamic COPs as well.
The contributions that stem from this PhD thesis are:

- **A state-of-the-art on dynamic vehicle routing problem**. It covers the description of the problem and its variants, the recent solving methods and the available benchmark data sets.

- **A framework for benchmark generation for dynamic VRPs**. It allows the generation of dynamic large scale instances according to different dynamic scenarios as broken down vehicles, variable travel times, etc. In terms of variants, it can generate instances with time windows, multi-depot, and pickup and delivery problems. These instances could be customized according to different spatial topologies of customers in the service area (cluster, uniform, mix), as well as time distributions (i.e. the arrival of customer demands can follow uniform, Poisson, or normal distributions).

- An analysis and a detailed description of the **performance measures** that should be used with DCOPs. These measures have been used to assess the different proposed metaheuristics in terms of accuracy, stability and reaction time when solving DVRP.

- **Efficient single-solution based metaheuristic** for DVRP. This class is represented by **Variable Neighborhood Search (VNS)**. The interest of this approach consists in the ability of shifting from a neighborhood to another one throughout the optimization process. This ability offers an adaptive mechanism for tracking the optimum during the environmental changes. For this proposal, dedicated neighborhoods have

been integrated to increase the efficiency of the approach. The experiments demonstrate that VNS is able to reach high quality solutions in few generations and is very competitive compared to other metaheuristics of literature.

- Enhance the performance of the standard **population based metaheuristics** such **Particle Swarm Optimization (PSO)** by adding an adaptive memory mechanism. This memory brings a new dimension to the traditional issue of balancing diversification and intensification. It consists in reusing the solutions gathered previously by the particles, and reuse them when the change occurs for repositioning the particles in the search space. For this purpose an Adaptive Particle Swarm Optimization (APSO) has been developed. The experimental results showed the benefits of the proposed adaptation in enhancing the overall performance of our algorithm.

- Counterbalance the effect of diversity loss of P-metaheuristics by **maintaining diversity** throughout the run. This was achieved by the multi-population approach called **Multi-Adaptive Particle Swarm (MAPSO)**. Its principle is to divide the population into several subpopulations that evolve in parallel and which track the shifting optimum throughout the time. The aim is to position each of those subswarms on different, promising local optima of the search space. In addition, we suggested to cooperate the swarms by exchanging information related to the best visited solutions (local optima) found by their respective particles. Since simply breaking up the neighborhoods and dividing up the global swarm into a number of independent swarms could lead to a situation where swarms would not interact and may find themselves isolated. Hence, we take the advantage of using the information gathered in the past and introduce more diversification in the search towards several cooperative swarms. For the experimental part, MAPSO introduces the best solutions so far on several classes of instances that belong to conventional set of benchmark for DVRP. The dynamic performance measures reinforce these results, while the accuracy of the obtained solutions demonstrates its competitiveness. Study on varying the number of sub-populations has been done, and showed that the performances increase with the number of sub-swarms which maintain population diversity through the search space.

- Demonstrate the role of **parallel approaches** in dynamic environments. Parallelizing metaheuristics in real-time context is an important issue due to the hard requirement on search time especially when we

deal with strong dynamic problems in which changes occur frequently and within short time slots. Furthermore, Parallel metaheuristics allow to solve large-scale instances of complex optimization problems. MAPSO approach has been implemented on parallel architecture by using ParadisEO software framework and experiments were carried out on GRID'5000. It showed its efficiency with new large-scale instances generated for DVRP.

- **Flexible strategy** for dynamic vehicle routing problem by searching solutions that account for possible future changes by allowing easy and successful adaptation after a change of the environment. We proposed to dynamically adjust the length of the planning horizon in order to make decisions which ensure that a vehicle will always be available to process new customer orders in a sub-region throughout the day. Therefore, it will be easier to place these customers in the existing routes and less detours are made for servicing customers what reduces the related cost. The flexible strategy was integrated into our **Flexible Variable Neighborhood Search (FVNS)**. The obtained robust solutions perform well over a wide range of customer topology distribution and empirical evaluation confirms that impressive improvement can be archived by this strategy.

Perspectives

While working on this Ph.D. thesis, some areas to improve further have arisen. They form the basis for future works:

  - The generalized framework of benchmarking can be extended to other COPs. It is hoped that in the future other interesting dynamic COPs (scheduling, planning, assignment, etc.) are constructed under this framework by taking into account their corresponding properties and different dynamic scenarios.
  - Some algorithms suggested in this work have some potential for improvement. MAPSO, for example, could be enhanced by considering the proximity between swarms and favoring swarms in deserted areas of the search space in order to provide a more even converge. The distance between swarms has to be defined. For instance, it could be done by measuring the phenotypic diversity

based on difference in solution fitness between the centroids of each swam (global best) .

Furthermore, VNS could be extended by a Cooperative Parallel Muti-Start paradigm in which different local search algorithms are launched using diverse initial solutions, and cooperate by exchanging information related to their searches at different steps of optimization progress.

- The application and the adaptation of the proposed approaches on dynamic multi-objective variants of the problem should be examined. The aim is to track a new Pareto-optimal front, as soon as there is a change in the problem. In handling such problems, not many algorithms exist, and certainly, there is a lack of test problems to adequately test a dynamic population based metaheuristics on multi-objective optimization problems.
- The effectiveness of the developed methods on the DVRP encourages their application to other dynamic problems, such as intelligent transportation systems, engine parameter control, scheduling of airline maintenance, and dynamic network routing. With these problems, however, several important applications dependent aspects may have to be investigated. Examples, the flexibility measure and neighborhood definition of a solution for each problem.
- Further benefits might arise from considering simultaneously the different aspects that cover this thesis. For instance, searching for robust solution will reduce the need to adapt solution, reducing the change cost. Thus, coupling the different metaheuristics such as MAPSO and VNS by their hybridization as well as the strategy of flexibility can lead to high competitive paradigms.
- There are other aspects related to the optimization in dynamic environment that have an increasing interests like for example the incorporation of learning. Indeed, real world models allow to add some prediction on forthcoming events. In our case, we could predict the future customer demands by following probabilistic models obtained by analyzing the customer demands (arrival time, location, quantity, etc.) over a certain period. Thus, our problem will converge towards a Stochastic Dynamic Vehicle Routing Problem (SDVRP), a wide class of combinatorial optimization problems under uncertainty, where part of the information about the problem data is unknown at the planning stage, but some knowledge about its probability distribution is assumed. Here, the aim is to extend and adapt our algorithms to deal with this class of problems.

– Dynamic optimization problems are more and more complex and their resource requirements increase. For instance, strongly dynamic systems are characterized by the fast pace of changes in the data and the urgency of almost all requests received. Emergency services, such as police, fire and ambulance departments exhibit strong dynamic behavior. Another example is taxi cab services in which only a negligible number of the customers have ordered their ride in advance.

  The importance of these problems motivates the analysis of their associated costs and quality of the obtained solutions. In particular, ways to decrease response times. Recently, the use of graphics processors has been extended to general application domains such as computational science. Indeed, GPUs are very efficient at manipulating computer graphics, and their parallel structure makes them more efficient than general-purpose CPUs for a range of complex algorithms. This is why it would be very interesting to exploit this huge capacity of computing to implement parallel metaheuristics for dynamic optimization problems.

The increasing dynamism in real-world problems and competition among various enterprises are likely to bring about newer optimization problems in which adaptation is absolutely essential. The proposed approaches are very promising, and the prospect of obtaining a powerful and widely metaheuristics for other dynamic problems is so tempting. We believe that the applications examined in this thesis show how difficult it is to build general and generic algorithms for solving DCOPs and particularly DVRPs and that conventional methods need to be adapted to the problems by adding different mechanisms in order to track the shifting optimum efficiently throughout the time.

Publications

The results of the research work presented in this PhD thesis have been published. In what follows these publications are listed, grouped by type of publication and sorted chronologically within each group:

Journals

1. M.R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E-G. Talbi. A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests. Applied Soft Computing Journal (to appear), Elsevier, 2011.

Chapters

2. M.R. Khouadjia, B. Sarasola, E. Alba, , E-G. Talbi, and L. Jourdan. Metaheuristics for dynamic vehicle routing problems. In Metaheuristics for Dynamic Optimization Problems (to appear), Springer, 2011.
3. M.R. Khouadjia, L. Jourdan, and E-G. Talbi. Livre communicant CISIT (in press), chapter Résolution du problème dynamique de tournées de véhicules par essaim de particules. Hermes Science, 2010.

International Conferences

4. M.R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E-G. Talbi. Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. In proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium(IPDPS'11), Workshop on Nature Inspired Distributed Computing (NIDISC'11), pages 395–403, 2011.
5. B. Sarasola, M.R. Khouadjia, E. Alba, L. Jourdan, and E-G. Talbi. Flexible variable neighborhood search in dynamic vehicle routing. In proceedings of the 8th European event on Evolutionary Algorithms in Stochastic and Dynamic Environments (EvoSTOC'11), pages 344–353, 2011.
6. M.R. Khouadjia, E. Alba, L. Jourdan, and E-G. Talbi. Parallel particle swarm optimization for solving the dynamic vehicle routing problem. In The 3rd International Conference on Metaheuristics and Nature Inspired Computing (META10), 2010.
7. M.R Khouadjia, E. Alba, L. Jourdan, and E-G. Talbi. Multi-swarm optimization for dynamic combinatorial problems: a case study on dynamic vehicle routing problem. In proceedings of the 7th International Conference on Swarm Intelligence, (ANTS'10), pages 227–238, 2010.

8. M.R. Khouadjia, L. Jourdan, and E. Talbi. Adaptive particle swarm for solving the dynamic vehicle routing problem. In IEEE/ACS International Conference on Computer Systems and Applications (AICCSA'2010), pages 1–8, 2010.
9. M.R. Khouadjia, L. Jourdan, and E-G. Talbi. A particle swarm for the resolution of the dynamic vehicle routing problem. In International Conference on Metaheuristics and Nature Inspired Computing, META'08, 2008.

Submitted Papers

10. M.R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E-G. Talbi. Parallel metaheuristics for solving dynamic vehicle routing problem. The Journal of Supercomputing, Springer, 2011.
11. M.R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E-G. Talbi. Dynamic Vehicle Routing Problems: Overview, Approaches, and Performance Measures. Computers and Operations Research (COR), Elsevier, 2011.

Bibliography

[Afsar 2010] H. M. Afsar. *A Branch-and-Price Algorithm for Capacitated Arc Routing Problem with Flexible Time Windows*. Electronic Notes in Discrete Mathematics, vol. 36, pages 319 – 326, 2010. ISCO 2010 - International Symposium on Combinatorial Optimization. 153

[Ai 2009] T.J. Ai and V. Kachitvichyanukul. *A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery*. Computers & Operations Research, vol. 36, pages 1693–1702, 2009. 100

[Alba 2005] E. Alba. Parallel metaheuristics: a New Class of Algorithms. Wiley-Interscience, 2005. 129, 145

[Alvarenga 2005] G.B. Alvarenga, R.M.A. Silva and G.R. Mateus. *A Hybrid Approach for the Dynamic Vehicle Routing Problem with Time Windows*. In Proceedings of the Fifth International Conference on Hybrid Intelligent Systems, pages 61–67. IEEE Computer Society Washington, DC, USA, 2005. 34, 37, 46, 49, 52

[Angelelli 2004] E. Angelelli, R.Mansini and M. G. Speranza. *A Real-time Vehicle Routing Model for a Courier Service Problem*. In Bernhard Fleischmann and Andreas Klose, editeurs, Distribution Logistics, volume 544 of *Lecture Notes in Economics and Mathematical Systems*, pages 87–103. Springer Berlin Heidelberg, 2004. 37

[Attanasio 2004] A. Attanasio, J-F. Cordeau, G. Ghiani and G. Laporte. *Parallel Tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem*. Parallel Computing, vol. 30, no. 3, pages 377 – 387, 2004. 35, 44, 53, 55, 63

[Bent 2003] R. Bent and P. Van Hentenryck. *Dynamic vehicle routing with stochastic requests*. In Proceedings of the 18th international joint conference on Artificial intelligence, pages 1362–1363, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. 30, 35, 37

[Bent 2004] R. Bent and P.Van Hentenryck. *Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers*. Operations Research, vol. 52, page 2004, 2004. 30, 35, 37

[Bent 2007] R. Bent and P. Van Hentenryck. *Waiting and relocation strategies in online stochastic vehicle routing*. In Proceedings of the 20th international joint conference on Artifical intelligence, pages 1816–1821, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 36, 37, 41, 76

[Bertsimas 1991] D.J. Bertsimas and G. van Ryzin. *A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane*. Operations Research, vol. 39, no. 4, pages 601–615, 1991. 33, 37, 40

[Bertsimas 1993a] D. Bertsimas and G.J. van Ryzin. *Stochastic and dynamic vehicle routing problem in the Euclidean plane with multiple capacitated vehicles*. Operations Research, vol. 41, pages 60–76, 1993. 40

[Bertsimas 1993b] D. Bertsimas and G.J. van Ryzin. *Stochastic and dynamic vehicle routing with general demand and interarrival time distributions*. Advanced Applied Probability, vol. 25, pages 947–978, 1993. 33, 37, 40

[Bianchi 2000] L. Bianchi. *Notes on dynamic vehicle routing -the state of the art-*. Rapport technique, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 2000. 29, 30, 32, 40

[Bianchi 2009] L. Bianchi, M. Dorigo, L. Gambardella and W. Gutjahr. *A survey on metaheuristics for stochastic combinatorial optimization*. Natural Computing, vol. 8, pages 239–287, 2009. 27

[Blackwell 2002] T. Blackwell and P.J. Bentley. *Dynamic search with charged swarms*. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 19–26, 2002. 126

[Blackwell 2004] T. Blackwell and J. Branke. *Multi-swarm Optimization in Dynamic Environments*. In Applications of Evolutionary Computing, volume 3005, pages 489–500. Springer, 2004. 126

[Blackwell 2006] T. Blackwell and J. Branke. *Multiswarms, exclusion, and anti-convergence in dynamic environments*. IEEE transactions on evolutionary computation, vol. 10, no. 4, pages 459–472, 2006. 126

[Blackwell 2007] T. Blackwell. *Particle swarm optimization in dynamic environments*. In Evolutionary Computation in Dynamic and Uncertain Environments, volume 51, pages 29–49. Springer, 2007. 90

Bibliography

[Blanton 1993] J.L. Blanton and R.L. Wainwright. *Multiple Vehicle Routing with Time and Capacity Constraints Using Genetic Algorithms*. In Proceedings of the 5th International Conference on Genetic Algorithms, pages 452–459, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. 47

[Bosman 2006] P. Bosman and H. La Poutré. *Computationally Intelligent Online Dynamic Vehicle Routing by Explicit Load Prediction in an Evolutionary Algorithm*. In Parallel Problem Solving from Nature - PPSN IX, volume 4193 of *Lecture Notes in Computer Science*, pages 312–321. Springer Berlin / Heidelberg, 2006. 47, 49

[Branchini 2009] R.M. Branchini, A.V. Armentano and A. Løkketangen. *Adaptive granular local search heuristic for a dynamic vehicle routing problem*. Computers & Operations Research, vol. 36, no. 11, pages 2955–2968, 2009. 32, 37, 41, 42

[Branke 1999a] J. Branke. *Memory enhanced evolutionary algorithms for changing optimization problems*. Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, vol. 3, pages –1882 Vol. 3, 1999. 105, 106, 125, 131

[Branke 1999b] J. Branke. *The moving peaks benchmark website*. [Online]. Available: http://www.aifb.uni-karlsruhe.de/ jbr/movpeaks, 1999. 127

[Branke 2000] J. Branke, T. Kaußler, C. Schmidt and H. Schmeck. *A multi-population approach to dynamic optimization problems*. Adaptive computing in design and manufacturing, vol. 2000, 2000. 125

[Branke 2002] J. Branke. Evolutionary optimization in dynamic environments. Kluwer Academic Publishers, 2002. 28

[Branke 2005a] J. Branke and D. C. Mattfeld. *Anticipation and flexibility in dynamic scheduling*. International Journal of Production Research, vol. 43, no. 15, pages 3103–3129, August 2005. 153

[Branke 2005b] J. Branke, M. Middendorf, G. Noeth and M. Dessouky. *Waiting Strategies for Dynamic Vehicle Routing*. Transportation Science, vol. 39, no. 3, pages 298–312, 2005. 40, 41, 46, 49, 55, 76, 99

[Breedam 1994] A. Van Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints*. PhD thesis, University of Antwerp, Belgium, 1994. 42

Bibliography

[Carlisle 2002] A. Carlisle and G. Dozler. *Tracking changing extrema with adaptive particle swarm optimizer*. Automation Congress, 2002 Proceedings of the 5th Biannual World, vol. 13, pages 265–270, 2002. 90

[Černỳ 1985] V. Černỳ. *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*. Journal of optimization theory and applications, vol. 45, no. 1, pages 41–51, 1985. 59

[Chen 2006a] A. Chen, G. Yang and Z. Wu. *Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem*. Journal of Zhejiang University-Science A, vol. 7, no. 4, pages 607–614, 2006. 100

[Chen 2006b] H-K. Chen, C-F. Hsueh and M-S. Chang. *The real-time time-dependent vehicle routing problem*. Transportation Research Part E: Logistics and Transportation Review, vol. 42, no. 5, pages 383 – 408, 2006. 55

[Chitty 2004] D. M. Chitty and M.L. Hernandez. *A Hybrid Ant Colony Optimisation Technique for Dynamic Vehicle Routing*. In Genetic and Evolutionary Computation – GECCO 2004, volume 3102 of *Lecture Notes in Computer Science*, pages 48–59. Springer Berlin / Heidelberg, 2004. 45, 49

[Christofides 1984] N. Christofides and J. Beasley. *The period routing problem*. Networks, vol. 14, page 237256, 1984. 52, 78, 111, 136

[Clarke 1964] G. Clarke and J. W. Wright. *Scheduling of Vehicles from a Central Depot to a Number of Delivery Points*. Operations Research, vol. 12, no. 4, pages 568–581, July 1964. 71, 155

[Cohen 1995] P.R. Cohen and R. Kohavi. Empirical methods for artificial intelligence. MIT press Cambridge, Mass, 1995. 114, 140

[Cordeau 2003] J-F. Cordeau and G. Laporte. *A tabu search heuristic for the static multi-vehicle dial-a-ride problem*. Transportation Research Part B: Methodological, vol. 37, no. 6, pages 579 – 594, 2003. 35, 55

[Cung 1997a] Van-Dat Cung, T. Mautor, P. Michelon and A. Tavares. *A scatter search based approach for the quadratic assignment problem*. In Evolutionary Computation, 1997., IEEE International Conference on, pages 165 –169, 1997. 93

[Cung 1997b] V.D. Cung, T. Mautor, P. Michelon and A. Tavares. *Improving the efficiency of Scatter Search.* In 2nd Metaheuristics International Conference (MIC'97), 1997. 93

[Cung 2002] V.D. Cung, S.L. Martins, C.C. Ribeiro and C. Roucairol. *Strategies for the parallel implementation of metaheuristics.* Essays and surveys in metaheuristics, vol. 15, 2002. 145

[Dantzig 1959] G.B. Dantzig and J.H. Ramser. *The truck dispatching problem.* Operations Research, Management Sciences, vol. 6(1), pages 80–91, 1959. 24

[Djadane 2006] M. Djadane, G. Goncalves, T. Hsu and R. Dupas. *Dynamic Vehicle Routing Problems under Flexible Time Windows and Fuzzy Travel Times.* Service Systems and Service Management, 2006 International Conference on, vol. 2, pages 1519–1524, 2006. 37

[Eberhart 2001] R.C. Eberhart and Y. Shi. *Tracking and optimizing dynamic systems with particle swarms.* In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, volume 1, pages 94–100. IEEE, 2001. 90

[Eggermont 2001] J. Eggermont, T. Lenaerts, S. Poyhonen and A. Termier. *Raising the Dead: Extending Evolutionary Algorithms with a Case-Based Memory.* In Genetic Programming, volume 2038 of *Lecture Notes in Computer Science*, pages 280–290. Springer Berlin / Heidelberg, 2001. 106

[Fabri 2006] A. Fabri and P. Recht. *On dynamic pickup and delivery vehicle routing with several time windows and waiting times.* Transportation Research Part B: Methodological, vol. 40, no. 4, pages 335–350, May 2006. 34, 55

[Fisher 1995] M. Fisher. *Vehicle routing.* In C.L. Monma M.O. Ball T.L. Magnanti and G.L. Nemhauser, editeurs, Network Routing, volume 8 of *Handbooks in Operations Research and Management Science*, pages 1 – 33. Elsevier, 1995. 52, 78, 111, 136

[Frieze 1982] A.M. Frieze, G. Galbiati and F. Maffioli. *On the worst-case performance of some algorithms for the asymmetric traveling salesman problem.* Networks, vol. 12, no. 1, pages 23–39, 1982. 70

[Funke 2005] B. Funke, T. Grünert and S. Irnich. *Local search for vehicle routing and scheduling problems: Review and conceptual integration.* Journal of heuristics, vol. 11, no. 4, pages 267–306, 2005. 68

[Gambardella 2003] LM. Gambardella, AE Rizzoli, F. Oliverio, N. Casagrande, AV Donati, R. Montemanni and E. Lucibello. *Ant Colony Optimization for vehicle routing in advanced logistics systems*. In MAS2003-The International Workshop on Modeling & Applied Simulation, pages 3–9, 2003. 44

[Garrido 2010] P. Garrido and M.C. Riff. *DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic*. Journal of Heuristics, vol. 16, pages 795–834, December 2010. 38, 46, 49

[Gee 2010] *Geekbench benchmark*. `http://www.primatelabs.ca/geekbench/`, 2010. 80, 112, 158

[Gendreau 1998] M. Gendreau and J-Y. Potvin. *Dynamic Vehicle Routing and Dispatching*. In Teodor G. Crainic and Gilbert Laporte, editeurs, Fleet management and logistics, chapitre 5, pages 115–126. Kluwer, Boston, 1998. 30

[Gendreau 1999] M. Gendreau, F. Guertin, J-Y. Potvin and E. Taillard. *Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching*. Transportation Science, vol. 33, no. 4, pages 381–390, 1999. 32, 37, 43, 46, 49, 62, 75

[Gendreau 2006] M. Gendreau, F. Guertin, J-Y. Potvin and R. Séguin. *Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries*. Transportation Research Part C: Emerging Technologies, vol. 14, no. 3, pages 157 – 174, 2006. 30, 37, 43, 52, 55

[Gendreau 2010] M. Gendreau and C.D. Tarantilis. *Solving large-scale vehicle routing problems with time windows: The state-of-the-art*. Rapport technique, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, 2010. 68

[Ghiani 2003] G. Ghiani, F. Guerriero, G. Laporte and R. Musmanno. *Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies*. European Journal of Operational Research, vol. 151, pages 1–11, 2003. 23, 30, 32, 40

[Glover 1990] F. Glover. *Tabu search, Part I*. ORSA Journal on computing, vol. 2, no. 1, pages 4–32, 1990. 59

Bibliography

[Goldberg 1987] D.E. Goldberg and R.E. Smith. *Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy.* In John J. Grefenstette, editeur, Proc. of the 2nd International Conference on Genetic Algorithms and Their Applications, pages 59–68. Lawrence Erlbaum Associates, 1987. 105

[Golden 1977] B. L. Golden, T. L. Magnanti and H. Q. Nguyen. *Implementing vehicle routing algorithms.* Networks, vol. 7, no. 2, pages 113–148, 1977. 71

[Haghani 2005] A. Haghani and S. Jung. *A dynamic vehicle routing problem with time-dependent travel times.* Computers & Operations Research, vol. 32, no. 11, pages 2959 – 2986, 2005. 27, 34, 37, 47

[Hansen 1999] P. Hansen and N. Mladenović. *An introduction to variable neighborhood search.* In S. Vo, S. Martello, I. Osman and C. Roucairol, editeurs, Metaheuristics: Advances and trends in local search paradigms for optimization, chapitre 30, pages 433–458. Kluwer Academic Publishers, 1999. 58, 59, 63, 155

[Hanshar 2007] F.T. Hanshar and B.M. Ombuki-Berman. *Dynamic vehicle routing using genetic algorithms.* Applied Intelligence, vol. 27, pages 89–99, 2007. 23, 28, 30, 37, 38, 39, 44, 45, 49, 63, 74, 79, 80, 81, 86, 99, 111, 112, 113, 119, 136, 139, 144, 158, 159, 160, 162

[Hashimoto 2006] H. Hashimoto, T. Ibaraki, S. Imahori and M. Yagiura. *The VRP with flexible time windows and traveling times.* Discrete Appl. Math., vol. 154, pages 2271–2290, 2006. 153

[Housroum 2006] H. Housroum, T. Hsu, R. Dupas and G. Goncalves. *A hybrid GA approach for solving the Dynamic Vehicle Routing Problem with Time Windows.* In 2nd International Conference on Information & Communication Technologies: Workshop ICT in Intelligent Transportation Systems, ICTTA'06, volume 1, pages 787–792, 2006. 23, 28, 34, 37, 38, 39, 46, 49, 99

[Hvattum 2006] L.M. Hvattum, A. Løkketangen and G. Laporte. *Solving a Dynamic and Stochastic Vehicle Routing Problem with a Sample Scenario Hedging Heuristic.* Transportation Science, vol. 40, pages 421–438, 2006. 35, 37, 41

[Ichoua 2000] S. Ichoua, M. Gendreau and J-Y. Potvin. *Diversion Issues in Real-Time Vehicle Dispatching.* Transportation Science, vol. 34, pages 426–438, 2000. 43, 49

[Ichoua 2003] S. Ichoua, M. Gendreau and J-Y. Potvin. *Vehicle dispatching with time-dependent travel times*. European Journal of Operational Research, vol. 144, pages 379–396, 2003. 49

[Jans 2007] R. Jans and Z. Degraeve. *Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches*. European Journal of Operational Research, vol. 177, no. 3, pages 1855 – 1875, 2007. 152

[Jih 1999] W-R. Jih and J. Yung-Jen Hsu. *Dynamic vehicle routing using hybrid genetic algorithms*. In IEEE International Conference on Robotics and Automation, volume 1, pages 453 –458 vol.1, 1999. 47, 49, 99

[Jones 1995] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, Univ. of New Mexico, Albuquerque, NM,, 1995. 58, 63

[Jun 2008] Q. Jun, J. Wang and B-J. Zheng. *A Hybrid Multi-objective Algorithm for Dynamic Vehicle Routing Problems*. In Proceedings of the 8th international conference on Computational Science, Part III, ICCS '08, pages 674–681, Berlin, Heidelberg, 2008. Springer-Verlag. 44, 49, 98

[Kennedy 1995] J. Kennedy and R. Eberhart. *Particle swarm optimization*. In IEEE International Conference on Neural Networks, 1995., volume 4, pages 1942–1948., 1995. 94

[Kennedy 2001] J. Kennedy, R. Eberhart and Y. Shi. Swarm intelligence. Morgan Kaufmann Publishers, 2001. 94

[Kilby 1998] P. Kilby, P. Prosser and P. Shaw. *Dynamic VRPs: A study of scenarios*. APES-06-1998, University of Strathclyde, U.K., 1998. 28, 32, 37, 41, 52, 75, 78, 80, 111, 135, 156

[Kindervater 1989] G.A.P. Kindervater, J.K. Lenstra and D.B. Shmoys. *The parallel complexity of TSP heuristics*. Journal of Algorithms, vol. 10, no. 2, pages 249–270, 1989. 70

[Kirkpatrick 1983] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi. *Optimization by simulated annealing*. science, vol. 220, no. 4598, page 671, 1983. 59

[Larsen 2000] A. Larsen. *The Dynamic Vehicle Routing Problem*. PhD thesis, Technical University of Denmark, 2000. 28, 29, 30

Bibliography

[Larsen 2002] A. Larsen, O.B.G. Madsen and M.M. Solomon. *Partially dynamic vehicle routing-models and algorithms.* The Journal of the Operational Research Society, vol. 53, no. 6, pages 637–646, 2002. 30, 33, 34

[Larsen 2004] A. Larsen, O.B.G. Madsen and M.M. Solomon. *The a priori dynamic traveling salesman problem with time windows.* Transportation Science, vol. 38, no. 4, pages 459–472, 2004. 34, 37, 41

[Larsen 2008] A. Larsen, O.B.G. Madsen and M.M. Solomon. *Recent Developments in Dynamic Vehicle Routing Systems.* In Bruce Golden, S. Raghavan and Edward Wasil, editeurs, The Vehicle Routing Problem: Latest Advances and New Challenges, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 199–218. Springer US, 2008. 25

[Lewis 1998] J. Lewis, E. Hart and G. Ritchie. *A Comparison of Dominance Mechanisms and Simple Mutation on Non-stationary Problems.* In Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, PPSN V, pages 139–148, London, UK, 1998. Springer-Verlag. 105

[Li 2006] X. Li, J. Branke and T. Blackwell. *Particle swarm with speciation and adaptation in a dynamic environment.* In GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 51–58, New York, NY, USA, 2006. ACM. 90

[Lin 1965] S. Lin. *Computer solutions of the traveling salesman problem.* Bell System Computer Journal, vol. 44, pages 2245–2269, 1965. 41, 68, 155

[Louis 1996] S.J. Louis and Z. Xu. *Genetic algorithms for Open Shop Scheduling and Re-Scheduling.* In M. E. Cohen and D. L. Hudson, editeurs, 11th Int. Conf. on Computers and their Applications (ISCA), pages 99–102, 1996. 105, 106

[Lund 1996] K. Lund, O.B.G. Madsen and J.M. Rygaard. *Vehicle Routing Problems with Varying Degrees of Dynamism.* Rapport technique, IMM, The Department of Mathematical Modelling, Technical University of Denmark, 1996. 28

[Magalhães 2006] J.M. De Magalhães and J. Pinho De Sousa. *Dynamic VRP in pharmaceutical distribution -a case study.* Central European Journal of Operations Research, vol. 14, no. 2, pages 177–192, 2006. 27

[Mitrović-Minić 2004a] S. Mitrović-Minić, R. Krishnamurti and G. Laporte. *Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows.* Transportation Research Part B: Methodological, vol. 38, no. 8, pages 669 – 685, 2004. 30, 34, 37, 41, 43, 49, 55, 62

[Mitrović-Minić 2004b] S. Mitrović-Minić and G. Laporte. *Waiting strategies for the dynamic pickup and delivery problem with time windows.* Transportation Research Part B: Methodological, vol. 38, no. 7, pages 635 – 655, 2004. 41

[Montemanni 2005a] R. Montemanni, L.M. Gambardella, A.E. Rizzoli and A.V. Donati. *Ant Colony System for a Dynamic Vehicle Routing Problem.* Journal of Combinatorial Optimization, vol. 10, pages 327–343, 2005. 52

[Montemanni 2005b] R. Montemanni, L.M. Gambardella, A.E. Rizzoli and A.V. Donati. *A new algorithm for a dynamic vehicle routing problem based on Ant colony system.* Journal of Combinatorial Optimization, vol. 10, pages 327–343, 2005. 23, 28, 30, 32, 37, 38, 39, 44, 45, 49, 55, 63, 74, 75, 79, 80, 81, 86, 98, 111, 112, 113, 119, 136, 139, 144, 158, 159, 160, 162

[Oliveira 2008] S. Oliveira, S.R. De Souza and M.A.L. Silva. *A Solution of Dynamic Vehicle Routing Problem with Time Window via Ant Colony System Metaheuristic.* In Neural Networks, 2008. SBRN '08. 10th Brazilian Symposium on, pages 21 –26, 2008. 34, 37, 45, 49

[Oppacher 1999] F. Oppacher and M. Wineberg. *The shifting balance genetic algorithm: Improving the GA in a dynamic environment.* In Proceedings of the Genetic and Evolutionary Computation Conference, volume 1, pages 504–510, 1999. 125

[Or 1996] I. Or. *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking.* PhD thesis, Northwestern University, Evanston, Illinois, 1996. 42

[Osman 1993] I.H. Osman. *Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem.* Annals of Operations Research, vol. 41, no. 4, pages 421–451, 1993. 44, 63, 67, 103

[Pankratz 2005] G. Pankratz. *Dynamic vehicle routing by means of a genetic algorithm.* International Journal of Physical Distribution & Logistics Management, vol. 35, no. 5, pages 362–383, 2005. 55

Bibliography

[Papastavrou 1996] J.D. Papastavrou. *A stochastic and dynamic routing policy using branching processes with state dependent immigration*. European Journal of Operational Research, vol. 95, no. 1, pages 167–177, 1996. 40

[Parrott 2004] D. Parrott and X. Li. *A particle swarm model for tracking multiple peaks in a dynamic environment using speciation*. In Evolutionary Computation, 2004. CEC2004. Congress on, volume 1, 2004. 126

[Pavone 2009] M. Pavone, N. Bisnik, E. Frazzoli and V. Isler. *A Stochastic and Dynamic Vehicle Routing Problem with Time Windows and Customer Impatience*. Mobile Networks and Applications, vol. 14, pages 350–364, 2009. 35, 37

[Potvin 1995] J-Y. Potvin and J-M. Rousseau. *An exchange heuristic for routing problems with time windows*. Journal of the Operational Research Society, vol. 46, pages 1433–1446, 1995. 68, 155

[Potvin 2006] J-Y. Potvin, Y. Xu and I. Benyahia. *Vehicle routing and scheduling with dynamic travel times*. Comput. Oper. Res., vol. 33, pages 1129–1137, April 2006. 36, 37

[Prins 2004] C. Prins. *A simple and effective evolutionary algorithm for the vehicle routing problem*. Computers & Operations Research, vol. 31, no. 12, pages 1985–2002, October 2004. 103

[Psaraftis 1988] H.N. Psaraftis. *Dynamic vehicle routing problems*. Vehicle routing: Methods and studies, vol. 16, pages 223–248, 1988. 24, 32, 37, 41

[Psaraftis 1995] H.N. Psaraftis. *Dynamic vehicle routing: status and prospects*. Annals of Opertations Reasearch, vol. 61, pages 143–164, 1995. 23, 24, 26

[Ramsey 1993] C.L. Ramsey and J.J. Grefenstette. *Case-based initialization of genetic algorithms*. In Proceedings of the Fifth International Conference on Genetic Algorithms, pages 84–91. Citeseer, 1993. 105, 106

[Rizzoli 2007] A. Rizzoli, R. Montemanni, E. Lucibello and L. Gambardella. *Ant colony optimization for real-world vehicle routing problems*. Swarm Intelligence, vol. 1, pages 135–151, 2007. 44, 45, 49

[Rochat 1995] Y. Rochat and É.D. Taillard. *Probabilistic diversification and intensification in local search for vehicle routing.* Journal of heuristics, vol. 1, no. 1, pages 147–167, 1995. 43, 62, 103, 106

[Salomon 1998] R. Salomon and P. Eggenberger. *Adaptation on the evolutionary time scale: A working hypothesis and basic experiments.* In Artificial Evolution, volume 1363 of *Lecture Notes in Computer Science*, pages 251–262. Springer Berlin / Heidelberg, 1998. 52

[Savelsbergh 1995] M. W. P. Savelsbergh and M. Sol. *The General Pickup and Delivery Problem.* Transportation Science, vol. 29, no. 1, pages 17–29, 1995. 34

[Scheffermann 2009] R. Scheffermann, M. Bender and A. Cardeneo. *Robust solutions for vehicle routing problems via evolutionary multiobjective optimization.* In Proceedings of the 11th Congress on Evolutionary Computation, pages 1605–1612, 2009. 153

[Shi 1998] Y. Shi and R. Eberhart. *A modified particle swarm optimizer.* In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, pages 69–73, 1998. 96

[Snoek 2001] M. Snoek. *Anticipation Optimization in Dynamic Job Shops.* In Proceedings of the 2001 Genetic and Evolutionary Computation Conference, 2001. 153

[Song 2005] J. Song, J. Hu, Y. Tian and Y. Xu. *Re-optimization in dynamic vehicle routing problem based on Wasp-like agent strategy.* In Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE, pages 231 – 236, 2005. 40

[Sörensen 2003] K. Sörensen. *A framework for robust and flexible optimization using meta-heuristics with applications in supply chain design.* PhD thesis, Antwerp, 2003. 152

[Sörensen 2009] K. Sörensen and M. Sevaux. *A Practical Approach for Robust and Flexible Vehicle Routing Using Metaheuristics and Monte Carlo Sampling.* Journal of Mathematical Modelling and Algorithms, vol. 8, pages 387–407, 2009. 153

[Sun 2007] L. Sun, X. Hu, Z. Wang and M. Huang. *A Knowledge-Based Model Representation and On-Line Solution Method for Dynamic Vehicle Routing Problem.* In ICCS '07: Proceedings of the 7th interna-

tional conference on Computational Science, Part IV, pages 218–226, Berlin, Heidelberg, 2007. Springer. 27

[Swihart 1999] M.R. Swihart and J.D. Papastavrou. *A stochastic and dynamic model for the single-vehicle pick-up and delivery problem*. European Journal of Operational Research, vol. 114, pages 447–464, 1999. 34, 37, 40

[Taillard 1993] É. Taillard. *Parallel iterative search methods for vehicle routing problems*. Networks, vol. 23, no. 8, pages 661–673, 1993. 52, 78, 111, 136

[Talbi 2009] E-G. Talbi. Metaheuristics: from design to implementation. Wiley-Blackwell, 2009. 42, 59, 72, 91, 93, 108, 111, 128, 134, 145

[Tian 2003] Y. Tian, J. Song, D. Yao and J. Hu. *Dynamic vehicle routing problem using hybrid ant system*. In IEEE Conference on Intelligent Transportation Systems, volume 2, pages 970 – 974 vol.2, 2003. 44, 49, 98

[Trojanowski 1997] K. Trojanowski, Z. Michalewicz and J. Xiao. *Adding memory to the Evolutionary Planner/Navigator*. In Evolutionary Computation, 1997, IEEE International Conference on, pages 483 –487, apr 1997. 106

[Ursem 2000] R.K. Ursem. *Multinational GAs: Multimodal Optimization Techniques in Dynamic Environments*. In Proceedings of the Second Genetic and Evolutionary Computation Conference. Morgan Kaufmann, 2000. 125

[Van Hemert 2004] J. Van Hemert and J. La Poutré. *Dynamic Routing Problems with Fruitful Regions: Models and Evolutionary Computation*. In Parallel Problem Solving from Nature - PPSN VIII, volume 3242 of *Lecture Notes in Computer Science*, pages 692–701. Springer Berlin / Heidelberg, 2004. 47, 49

[Wang 2006] W. Wang, B. Wu, Y. Zhao and D. Feng. *Particle Swarm Optimization for Open Vehicle Routing Problem*. In Computational Intelligence, volume 4114, pages 999–1007. Springer, 2006. 100

[Wang 2007] J-Q. Wang, X.-N. Tong and Z.-M. Li. *An Improved Evolutionary Algorithm for Dynamic Vehicle Routing Problem with Time Windows*. In ICCS'07: Proceedings of the 7th international conference on Computational Science, Part IV, pages 1147–1154, Berlin, Heidelberg, 2007. Springer. 27, 34, 47, 49, 99

[Weicker 1999] K. Weicker and N. Weicker. *On Evolution Strategy Optimization in Dynamic Environments*. In 1999 Congress on Evolutionary Computation, pages 2039–2046. IEEE Press, 1999. 50

[Weicker 2002] K. Weicker. *Performance measures for dynamic environments*. In Parallel Problem Solving from Nature PPSN VII, pages 64–76. Springer, 2002. 50, 58, 83, 117

[Xu 2008] J. Xu, G. Goncalves and H. Tinte. *Genetic algorithm for the vehicle routing problem with time windows and fuzzy demand*. In Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, pages 4125 –4129, 2008. 36, 37

[Yang 2004] J. Yang, P. Jaillet and H. Mahmassani. *Real-Time Multivehicle Truckload Pickup and Delivery Problems*. Transportation Science, vol. 38, pages 135–148, 2004. 35, 37

[Yellow 1970] P. C. Yellow. *A Computational Modification to the Savings Method of Vehicle Scheduling*. Operational Research Quarterly, vol. 21, no. 2, pages 281–283, 1970. 71, 155

[Zhao 2008] X. Zhao, G. Goncalves and R. Dupas. *A genetic approach to solving the vehicle routing problem with time-dependent travel times*. In 16th Mediterranean Conference on Control and Automation, pages 413 –418, 2008. 46, 49

[Zhu 2006] Q. Zhu, L. Qian, Y. Li and S. Zhu. *An Improved Particle Swarm Optimization Algorithm for Vehicle Routing Problem with Time Windows*. In Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pages 1386–1390, 2006. 100