Parking management system in a dynamic and multi-objective environment

Mustapha Ratli

№ d'ordre: 14/37

Thèse de doctorat

Pour obtenir le grade de Docteur de l'Université de

VALENCIENNES ET DU HAINAUT-CAMBRESIS

Discipline, spécialité selon la liste des spécialités pour lesquelles l'Ecole Doctorale est accréditée :

Sciences et Technologie, Mention : Informatique

Présentée et soutenue publiquement par: Mustapha RATLI

Le 12/12/2014 à 14h00

Ecole doctorale : Sciences Pour l'Ingénieur (SPI)

Equipe de recherche, Laboratoire : Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH), UMR CNRS 8201

Parking Management System in a Dynamic and Multi-Objective Environment

(Système de Gestion du Stationnement dans un Environnement Dynamique et Multi-Objectifs)

JURY

Président

- Abdelhakim ARTIBA. Professeur. Université de Valenciennes

Rapporteurs

- Feng CHU. Professeur. Université d'Evry Val d'Essone
- Aziz MOUKRIM. Professeur. Université de Technologie de Compiègne

Examinateurs

- Bruno DEFUDE. Professeur. Telecom Sud Paris

Directeur de thèse : Sylvain LECOMTE . Professeur. Université de Valenciennes
Co-directeur de thèse : Saïd HANAFI . Professeur. Université de Valenciennes

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

The parking problem is nowadays one of the major issues in urban transportation planning and traffic management research. In fact, the consequences of the lack of parking slots along with the inadequate management of these facilities are tremendous.The aim of this thesis is to provide efficient and robust algorithms in order to save time and money for drivers and to increase the income of parking managers. The problem is formulated as a multi-objective assignment problem in static and dynamic environments. First, for the static environment, we propose new two-phase heuristics to calculate an approximation of the set of efficient solutions for a bi-objective problem. In the first phase, we generate the supported efficient set with a standard dichotomic algorithm. In the second phase we use four metaheuristics to generate an approximation of the non-supported efficient solutions. The proposed approaches are tested on the bi-objective shortest path problem and the bi-objective assignment problem. For the dynamic environment, we propose a mixed integer linear programming formulation that is solved several times over a given horizon. The objective functions consist of a balance between the satisfaction of drivers and the interest of the parking managers. Two approaches are proposed for this dynamic assignment problem with or without learning phase. To reinforce the learning phase, an estimation of distribution algorithm is proposed to predict the future demand. in order to evaluate the effectiveness of the proposed algorithms, simulation tests have been carried out. A pilot implementation has also been conducted in the parking of the University of Valenciennes, using an existing platform called framework for context aware transportation services, which allows dynamic deployment of services. This platform can dynamically switch from one approach to another depending on the context. This thesis is part of the project SYstem For Smart Road Applications (SYFRA).

Keywords: Bi-objective shortest path problem, bi-objective assignment problem, smart parking, dynamic assignment problem, learning, metaheuristic.

Système de Gestion du Stationnement dans un Environnement Dynamique et Multi-Objectifs

Aujourd'hui, le problème de stationnement devient l'un des enjeux majeurs de la recherche dans la planification des transports urbains et la gestion du trafic. En fait, les conséquences de l'absence de places de stationnement ainsi que la gestion inadéquate de ces installations sont énormes. L'objectif de cette thèse est de fournir des algorithmes efficaces et robustes afin que les conducteurs gagnent du temps et de l'argent et aussi augmenter les revenus des gestionnaires de parking. Le problème est formulé comme un problème d'affectation multi-objectifs dans des environnements statique et dynamique. Tout d'abord, dans l'environnement statique, nous proposons de nouvelles heuristiques en deux phases pour calculer une approximation de l'ensemble des solutions efficaces pour un problème bi-objectif. Dans la première phase, nous générons l'ensemble des solutions supportées par un algorithme dichotomique standard. Dans la deuxième phase, nous proposons quatre métaheuristiques pour générer une approximation des solutions non supportées. Les approches proposées sont testées sur le problème du plus court chemin bi-objectif et le problème d'affectation bi-objectif. Dans le contexte de l'environnement dynamique, nous proposons une formulation du problème sous forme d'un programme linéaire en nombres entiers mixtes qui est résolue à plusieurs reprises sur un horizon de temps donné. Les fonctions objectives considérées, permettent un équilibre entre la satisfaction des conducteurs et l'intérêt du gestionnaire de parking. Deux approches sont proposées pour résoudre ce problème d'affectation dynamique avec ou sans phase d'apprentissage. Pour renforcer la phase d'apprentissage, un algorithme à estimation de distribution est proposé pour prévoir la demande future. Pour évaluer l'efficacité des algorithmes proposés, des essais de simulation ont été effectués. Aussi une mise en œuvre pilote a été menée dans le parking à l'Université de Valenciennes en utilisant une plate-forme existante, appelée Context Aware Transportation Services (CATS), qui permet le déploiement dynamique de services. Cette plate-forme peut dynamiquement passer d'une approche à l'autre en fonction du contexte. Enfin cette thèse s'inscrit dans le projet SYstem For Smart Road Applications (SYFRA).

Mots-clés: Plus court chemin bi-objectif, affectation bi-objectif, parking intelligent, affectation dynamique, apprentissage, métaheuristique.

Chapter 1

Smart parking management systems

1.1 Introduction

As the population keeps growing and the concentration of cars in cities increases, our society faces the significant challenge of global gridlock. To find a parking space becomes a common challenge, faced by millions of city-dwellers every day (Shoup et al., 2005), due to the significant and substantial increase in the demand for parking slots in cities and urban areas. Furthermore, the severe shortage of such spaces has created a challenge and a problem of management of these areas. Despite the creation of locations along roads and streets in the cities, it remains that these solutions do not absorb all of the demand, which is constantly increasing. The consequences are known, for example, an increase in traffic congestion and also economic, social and environmental losses. Moreover, with the continuous increase of the population, the problem becomes more and more critical. As such, the optimization of the parking slot allocation and control has become a real challenge for transport planners and traffic authorities. Most modern cities provide adequate support and guidance to drivers on the choice of parking slots and the efficient use of the parking slots in terms of variable message signs, directional arrows, the names of parks, the state, the number of parking slots, the actual entry, exit point of parking, etc. However, despite this, the circulation system and drivers are facing extreme difficulties especially during peak hours, or special events such as festivals, celebrations, new years and unpredictable situations of traffic congestion (Teodorović and Lučić, 2006). Ultimately in many urban or metropolitan areas, it is time consuming to find an available parking slot,

and when it is done, it is hard to know if it meets the aspirations of both drivers and the parking managers. Today, authorities are more concerned than ever with greenhouse gas emissions, and transport is one of the major contributors to this phenomenon. In the last decades several studies have been conducted to seek solutions to this problem and most of them are related to parking management. On a daily basis, it is estimated that 30% of cars on the road in the center of large cities are in search of a parking slot and it takes, on average, 7.8 minutes to find one (Arnott et al., 2005). This situation causes not only a waste of time and fuel for drivers looking for a parking slot, but it also contributes to further waste of time and fuel to other drivers due to traffic congestion. For example, it has been reported in (Shoup et al., 2005) that, during one year, in a small business district in Los Angeles, the distance travelled by cars looking for a parking was equivalent to 38 times around the world, burning 177 tons of gasoline and producing 730 tons of carbon dioxide. It has also been shown that over 40% of the total traffic volume in urban areas is composed of cars searching for parking (Shoup, 2006). By either decreasing the amount of cars searching for a parking slot or decreasing the waiting time to find one, it is possible to reduce pollution and to preserve resources (time and fuel) (Shoup et al., 2005).

The assignment problem of cars to parking slots has been discussed by many researchers and there have been numerous works studying this problem. (Caicedo, 2009) used two different ways of managing information availability of slot in parking facilities with PARC system to reduce the search time. (Caicedo, 2010) developed a demand assignment model in order to reduce the time and distance involved in the search for a parking slot. (Zhao and Collins Jr, 2005) developed a parallel algorithm for automatic parking in tight slots using a system based on a fuzzy logic controller. Spatial allocation of the parkings was analyzed by (Davis et al., 2010) to estimate the number of parking slots given a certain demand. (Leephakpreeda, 2007) presented a guide system for parkings. (Arnott and Rowse, 2009) developed an integrated parking on the sidewalk and control traffic congestion in inner-city model. (Shoup, 2006) presented a model to know whether drivers should search for a parking slot along the street or pay for off-street parking. (Teodorović and Lučić, 2006) proposed an inventory slot intelligent parking system. The system is based on a combination of fuzzy logic and techniques of integer programming.

It makes on line decisions to accept or reject the request of a new driver for a parking slot. (Ayala et al., 2011) presented a game-theoretic framework to analyze parking situations, and to introduce and analyze parking slot games in complete and incomplete information contexts. For both models they presented algorithms for individual players to choose parking slots ideally. (Benenson et al., 2008) presented an agent based system that simulates the behavior of each driver within a spatially explicit model. The system captures, within a non-homogeneous road space, the self-organizing and dynamics of a large collective parking agents. (Chou et al., 2008) presented an intelligent agent system with negotiable parking pricing for optimum car park for the driver. (Geng and Cassandras, 2012) proposed a system for an urban environment; the system assigns and reserves an optimal parking slot for each driver based on his requirements that combine proximity to destination and parking cost. In order to satisfy the users requirements and the parking managers at the same time, the authors used a Mixed Integer Linear Program (MILP) to solve this problem at each decision point. (Phillips, 1985) suggested a multi-objective optimization approach to assign university personnel cars to parking lots. (Geng and Cassandras, 2011) proposed a smart parking approach to help drivers in their search for an available parking lot. This approach is used by a centralized system that collects requests sent by drivers. The requests are collected over a certain time window after which, at the decision point, allocation of drivers to parking slots is made. In (Venkataramanan and Bornstein, 1991), a decision support system for the parking slot assignment is proposed where the parking lot assignment problem is modelled as a network problem. The proposed network combines the objectives of priority, cost and distance by weighting factors.

1.2 The issue of car parking demand

- **Parking demand:** refers to the amount of parking that would be used at a particular time, place and price. It is a critical factor in evaluating parking problems and solutions. Parking demand is affected by vehicle ownership, trip rates, mode split, duration (how long cars park), geographic location (*i.e*, downtown, regional town centre or suburban), the quality of travel alternatives, type of trip (work, shopping,

recreational), and factors such as fuel and road pricing. There are usually daily, weekly and annual demand cycles. For example, parking demand usually peaks on weekdays at office buildings and on weekend evenings at theaters and restaurants. Parking demand can change with transportation, land use and demographic patterns. For example, a particular building may change from industrial to residential or office use, neighborhood demographics and density may change, and the quality of transit service may change, all of which affect parking demand. Different types of trips have different types of parking demand, and different types of parking facilities tend to serve different types of trips. For example, commuters need long-term parking, and because they park all day they are relatively price sensitive. Many commuters are willing to walk several blocks for cheaper parking. Off-street parking leased by the month tends to serve commuters. Customers need shorter-term parking that is located as close as possible to their destination, and are often willing to pay a relatively high hourly price to increase convenience. On-street parking that is metered or regulated to maximize turnover tends to serve customers.

- **Parking Adequacy:** refers to whether there is sufficient parking at a particular time and location. What constitutes adequacy varies depending on conditions and user expectations. For example, even in dense areas parking is usually adequate, or at a sufficient price during off-peak periods. Similarly, parking may be considered inadequate at a particular location, but is available a few blocks away. Unregulated parking may be adequate for residents and employees, who park early in the day, but inadequate for delivery vehicles or clients, who arrive later. On the other hand, parking with a two-hour or less time limit, or more expensive, may be considered adequate for short-term users but inadequate for employees and residents who must park all day.

- **Parking problems:** most of the time, drivers consider parking as inadequate, inconvenient or expensive. This impression is based on the facts previously explained. As a solution, it is needed to increase parking supply without additional costs for the drivers. But there are other ways to tackle parking problems without making huge investments; for example developing new methods of management based on

technology of information and communication can be much more efficient than constructing new parking. Some situations that drivers may face when searching for a parking lot/slot are: inadequate information on parking availability and price; inconvenient parking pricing methods; economic, environmental and visual impacts of parking facilities, etc.

1.3 Cars and parking in smart cities

A real-time parking information system is composed of a variety of elements that generate raw data, that process the data into useable information, and that transmit the information to users. Each of these elements can be addressed with a number of different technology options. Parking guidance information systems, first proposed two decades ago, are used to minimize parking search traffic in large parking facilities and central cities by dynamically monitoring available parking, and directing drivers with variables message signs (Sakai et al., 1995). Parking guidance systems based on wireless sensor networks were also designed for automated and accurate monitoring of the parking slots and guidance to a vacant parking slot (Yoo et al., 2008).

Information and communication technologies are now integrated onboard systems for intelligent cars and offer new solutions to transportation problems. The main goal of these systems is to bring convenience, comfort and security for users. These systems rely, among others, on the current power of network technologies and the various protocols of communication (Biswas et al., 2006).

These systems help drivers to avoid accidents in the first place. In fact, several researches (see (Hafner et al., 2013)) have highlighted the dramatic impact of vehicular collisions in terms of costs, injuries and fatalities. Hence, intelligent cars will play a great role in reducing economical, social and environmental impacts. Some benefits of generalization of these cars are:

- Provide drivers with real-time information on traffic and allow them to avoid traffic jams (Baskar et al., 2011).
- Find the fastest or the shortest route between two places (Fu, 2001).

- Optimize the performance of their cars and thereby improve energy efficiency (Manzie et al., 2007).
- Monitor the status of the driver: vigilance, alcohol, drugs (Dong et al., 2011).
- Provide real-time information about available parking lots.

Due to the potential expected impact, car makers (such as Mercedes, BMW, or Renault) and independent laboratories intensify their research and development to come out with intelligent cars capable not only to assist the drivers all the time but also to become an interactive part of a more global and dynamic system as smart cities. As a consequence, during the last years, interest in applications for inter-vehicle communications increased in Europe and in the United States, giving rise to several communication projects such as: Vehicle Safety Communication, Car2Car Communication Consortium, Network on Wheels, Vehicular Event Sharing with a mobile P2P Architecture (VESPA).

In VESPA, the dissemination service is about to be stopped and replaced by a reservation service, which the trader can download and install. Whereas the use of V2V communications provides several advantages (cost, locality, dynamicity, etc.), it creates new technological challenges that the community faces in data management, to develop new techniques for data and query processing. (Delot et al., 2011) developed several prototypes for smart phones and evaluated the scalability of different approaches using simulation (see also (Delot and Ilarri, 2012), (Cenerario et al., 2011)). More precisely, they considered in the VESPA project (see Figure 1.1):

- The problem of assessing the relevance of the data exchanged in the network using both techniques based on distance calculations in the Euclidean space and technical operations of the road network information available in digital maps.
- The use of data relevance to design dissemination protocols.
- The use of data aggregation approaches that can help cars to extract and share environmental knowledge by summarizing the data elements.
- Information sharing about scarce resources such as parking slots; this is important because the communication of information for many cars without control could easily lead to sterile competition between cars to try to take the same resource.

- The problem of processing queries that need to access data remotely. They analyzed the challenge of conveying the results obtained remotely for a query back to the moving vehicle that issued this query. They are also studying the potential of mobile agent technology to route queries and results towards the appropriate nodes. The exchange of multimedia information is also discussed.

- The potential benefits of the application of the management of other technical and semantic data. They envision a future in which smart cars will be equipped with V2V communications and perform tasks for managing multiple data.

Figure 1.1: The Framework Architecture

It is clear that the outcomes of these projects will help the achievement of a safer automobile transportation. Nevertheless, many unresolved issues remain today including technical issues such as standardization, development issues such as application systems, and legal and institutional issues (Tsugawa, 2005).

1.4 Smart parking architectures

In order to address the problems associated with parking slots, smart parking has been developed for slot assignment or/and drivers guidance, and can be classified into centralized and distributed systems (see Figure 1.2 and Figure 1.3).

- **Centralized systems:** Centralized systems always have a server to store all parking slots information. They use sensor networks to gather parking slot information. Drivers must connect to the server through Internet or other communication means to access the parking slot information. These systems always have a database to store all parking slot information and a server to handle requests from users. The server gathers data of parking slot via vehicular ad hoc network VANET or wireless sensor networks with gateways, and updates its database with the latest information. The users must query the server to have knowledge of free parking slots. Although the centralized methods seem to work well, there are still some disadvantages. Constructing a reliable and scalable server costs a lot of money. And the region near the sinks or gateways to server will lead to heavy network loading. This will cause the congestion of network and decrease the data availability and accuracy. Thus, the service will be unstable in peak hours (Gibbons et al., 2003).

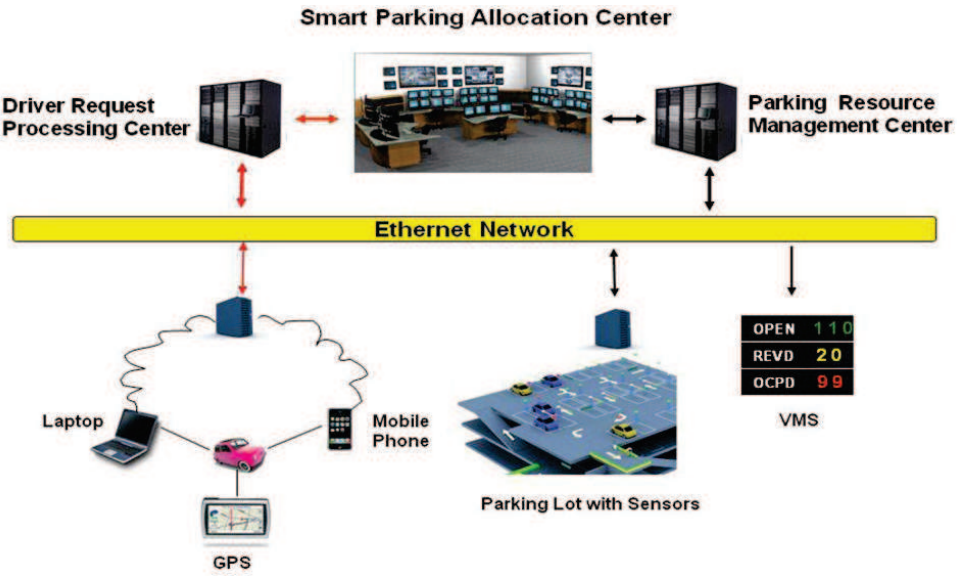

Figure 1.2: Centralized System Architecture

- **Distributed system:** An alternative to the previous architecture is the distributed

systems (Basu and Little, 2002), (Caliskan et al., 2006). They use wireless devices on parking meters and cars to construct a VANETs environment and then spread parking slot information on VANETs. The VANETs can provide many vehicular applications such as car safety, traffic analysis and information dissemination. The dissemination of parking slots information is one of the most popular applications in VANETs. It is able to gather and disseminate information in a dynamic and fast way, which is crucial as the availability of on-street parking slots is subject to frequent changes. This approach is characterized by the absence of a centralized infrastructure where each vehicle becomes at the same time a client and server through exchange or dissemination of information. Two modes can be distinguished:

- Cooperative mode, where cars share a common set of information so that everyone can locate one or more empty parking slots at the nearest destination and minimize the path relative to current their position.
- Non-cooperative mode (competitive) where cars do not share any information with each other. Each car must take into account, in its research strategy, the possibility that other cars choose the same slot as it.

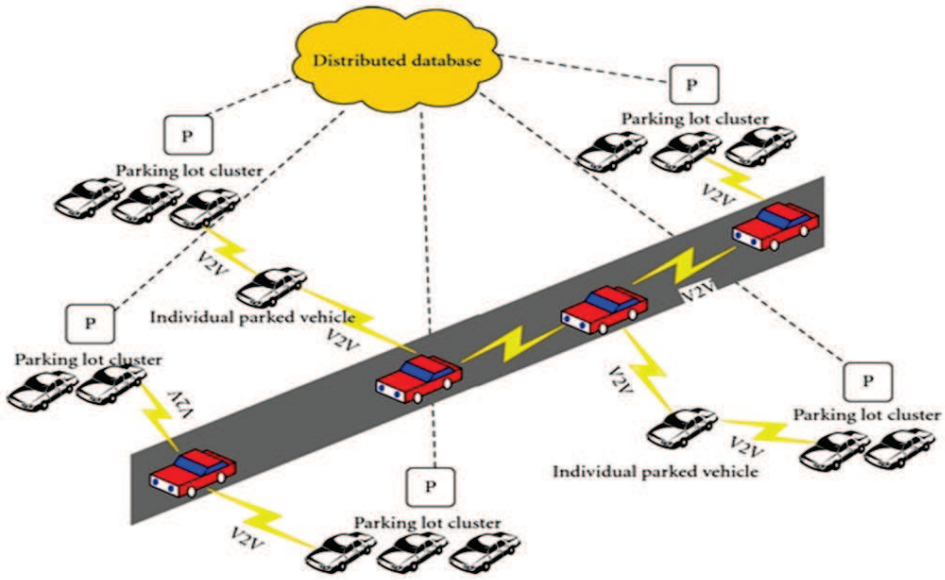

Figure 1.3: Distributed System Architecture

The cost of distributed systems is usually much lower than the cost of centralized ones, since there is no central server to dispatch the information. However, the main

disadvantage is that the wireless devices in distributed systems usually transmit a lot of redundant packets with the same parking information. Users with any mobile device or a car navigation system can communicate with the parking meters, book, spot and pay for parking even before arriving at the lot. But the same vacancies are displayed for all users, if the user just wants to check for vacant slots. This might cause multiple users to drive to the same slot and therfeore find it occupied when they get there.

Another reservation protocol was designed (Delot et al., 2009) using vehicular ad hoc networks where drivers can receive information about parking slots around them while driving. The protocol allocated efficiently parking slots to interested cars, thereby avoiding competition between them to get to the slot. Cars establish a VANET and receive, manipulate and relay parking slot information to cars in their vicinity. (Caliskan et al., 2007) proposed a mathematical model for parking lot occupancy prediction and an algorithm that uses parking lot data disseminated in a VANET to estimate the future occupancy of parking. This enables each vehicle to choose an appropriate parking lot.

Hybrid system architectures could use a combination of centralized and decentralized systems to form another one.

1.5 Context aware transportation services

A framework for Context Aware Transportation Services (CATS) has been developed at the University of Valenciennes. The goal of this framework is to provide an execution environment for service-based applications as well as management functionalities for the deployment and the adaptation to context changes. The aim of this framework is also to ensure continuous service applications, regardless of the conditions. That can be achieved by adjusting or replacing parts of the application with regard to the evolution of the context. The focus is on transportation applications such as routing, parking, traffic events in a context of low (pedestrians) or strong (VANETs) mobility. Some evaluations of CATS have been presented in (Popovici et al., 2011), showing that the framework is light enough for mobile devices such as smartphones. With CATS, using the ad-hoc

network we can detect services that offer a specific functionality to the area we are in, and we can benefit from them by installing them on our device. In (Popovici et al., 2011), the authors evaluated the necessary time for service download in different situations and found results coherent with our need: services can be downloaded fast enough from one-hop neighbors. In CATS, applications are composed of multiple functionalities, which can be divided in independent modules. The different reservation protocols are one of these functionalities.

With CATS, a simple dissemination service can be used to publish available parking slots on the network, or it can be replaced by reservation protocol, depending on the context in which the application is executed (centralized architecture or distributed architecture). The service switch modifies the architecture of the application and is the responsibility of the execution manager. However the decision of implementing this change is done according to the context and the available services that the trader can reach. These three components of CATS are illustrated on the right side of Figure 1.4.

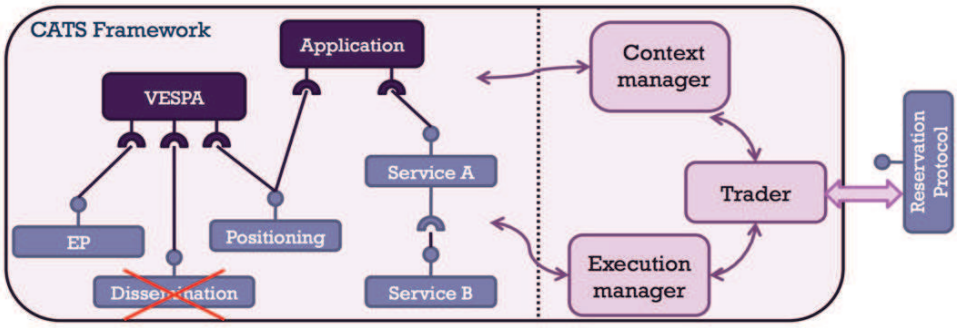

Figure 1.4: The Framework Architecture

In the configuration presented here, the dissemination service is about to be stopped and replaced by a reservation service, which the trader can download and install.

1.6 Parking search models

Parkings play an important role in the traffic system since all cars require a storage location when they are not being used to transport passengers. Due to the inherent uncertainty associated with many of the attributes of public car parks, including availability and location (Axhausen and Polak, 1991; Thompson and Richardson, 1998), a high proportion

of drivers travelling within central city areas must search for a parking slot. Generally, dedicated searching speed and searching time models represent a parking search phase. Such models are further split in two sub-models depending on the category of parking: on-street and off-street. In the on-street parking, searching speed is modeled by a fuzzy model as a function of the linear traffic density and the occupation rate of the parking facility. Searching time for on-street parking uses a probabilistic approach to calculate the searching time based on the occupation rate of the link connecting parking areas and searching speed. Searching time models estimate the time employed by a driver from the moment he decides to park up to the moment he finds the first available slot to park.

1.7 Contribution of this thesis

The current smart parking or parking guidance systems only obtain the availability information of parking slots from deployed sensor networks, and simply publish the parking information directly to drivers. However, since these systems cannot guide the drivers to their desired parking and destinations, and even sometimes make the situation worse, they are not smart enough. For instance, when the number of vacant slots in an area is limited, more drivers, who obtain the parking information, are heading for those slots. To alleviate such traffic problems and improve the convenience for drivers, the work presented in this thesis aims at proposing methods for assigning a parking lot/slot and computing driver paths across a metropolitan leading them, in a first moment, toward a parking slot and after to their final destination in a dynamic environment. Unlike classical assignment and shortest path problems, here the context is multi-objective due to existence of several criteria such as the distance/time between the current vehicle position and parking lot/slot and the distance/time between parking and the driver final destination, which must be considered in the path computation. In a multi-objective problem, there is no single solution, but a set of compromise solutions. Then, the difficulty is to compute assignments and shortest paths under a time constraint of a few seconds, in order to integrate the computation in the response time of lot/slot system assignment like centralized system architecture. In this thesis, we first propose new two-phase heuristics to approximate the set of the efficient solutions

to solve a bi-objective problem. In the first phase, we generate the supported efficient set by a standard dichotomic algorithm. In the second phase we propose four different metaheuristcs to generate an approximation of the non-supported efficient solutions. These metaheuristics are a cost perturbation method, a path-reliking, a genetic algorithm and finally a hybrid approach combining all of them. The hybrid approach combines genetic algorithm and mathematical programming techniques. This method is based on the dominance cost variant of the multi-objective genetic algorithm hybridized with an exact method. The initial population is generated by solving a series of mono-objective optimization problems obtained by a suitable choice of a set of weights. The crossover operator solves a reduced mono-objective problem where the weights are chosen to identify an unexplored region. The proposed approaches are tested on the bi-objective shortest path problem and the bi-objective assignment problem. In the second part, we consider a dynamic multi-objective assignment problem for a smart parking over a horizon. A mixed integer linear programming formulation is proposed. The objective functions consist of a balance between the satisfaction of the driver and the interest of the parking manager. The goal of a driver is to reduce the distance traveled between the assigned parking slot and his final destination. In addition, minimizing the waiting time to satisfy his request also interests him. Two approaches are proposed for this dynamic assignment problem with or without learning phase. To reinforce the learning phase, an estimation of distribution algorithm is proposed to adjust the demand during the horizon. To reduce the computational effort, a local search algorithm is introduced based on a decomposition scheme of the whole problem into a set of sub-problems with reduced number of cars parking slots.

Concepts of bi-objective optimization and methods are presented in Chapter II, and three sub-problems are considered in this thesis:

1. Approximation of the set of nondominated shortest paths, discussed in Chapter III.
2. Approximation of the set of non-dominated assignments, discussed in chapter IV.
3. Dynamic assignment with learning reinforcement, discussed in chapter V.

Chapter 2

Bi–objective combinatorial optimization problems

2.1 Introduction

Combinatorial optimization encompasses a wide class of problems with many real world applications in science, engineering, economics, social, medicine, etc. A combinatorial optimization problem consists in optimizing a given criterion with respect to various constraints. These constraints define the set of feasible solutions. However, the optimization problems in real applications are often multi-objective (production costs, quality, maintenance costs, times, distance, price, etc.), and the different criteria are usually conflicting. Thus, finding a single solution that optimizes all the criteria is a very difficult task. Therefore, using the multi-objective combinatorial optimization was the adequate solution for the interesting researchers in this field. Solving these multi-objective optimization problems exactly or approximatively requires methods that can generate the whole set of non–dominated points (called the Pareto-optimal front) or its approximation. For a general introduction to multi-objective optimization, we refer to (Ehrgott, 2005). In addition, Ehrgott and Gandibleux provided in (Ehrgott and Gandibleux, 2002) some details the multi-objective combinatorial optimization and presented their characteristics and the main findings of the related works.

In this chapter, the basics of multi-objective optimization are discussed and an introduction to some methods in multi-objective combinatorial optimization is given, including

the definitions of efficient solutions and non-dominated points. The two-phase method for bi-objective combinatorial optimization problems is also discussed.

2.2 Bi–objective combinatorial optimization

A combinatorial optimization problem consists of finding an optimal solution in a finite discrete set. Several problems of operational research are included in this framework, such as the knapsack problem, the assignment problem, the traveling salesman problem, and so on. The multi-objective combinatorial optimization belongs to the field of combinatorial optimization. Therefore, some definitions are inspired from combinatorial optimization, but specific concepts related to multi-objective are also introduced. Indeed, the main difference is the existence of several functions to optimize.

A bi-objective combinatorial optimization problem can be defined as follows:

$$(BP)\ "Minimize"\{c(x) = (c_1(x), c_2(x)) : x \in X\},$$

where c_1 and c_2 are two objective functions to minimize simultaneously, X is the set of feasible solutions, and x is n–dimensional vector of the decision variables.

2.2.1 Pareto optimality

First, we introduce some basic definitions for bi–objective optimization problems. The *objective space* is defined by $Y = \{(y_1, y_2) \in \mathbb{R}^2 : y_k = c_k(x), x \in X, k = 1, 2\}$. The set $Y \subset \mathbb{R}^2$ is the image of the set of feasible solutions X on the space of objectives, (see Figure 2.1).

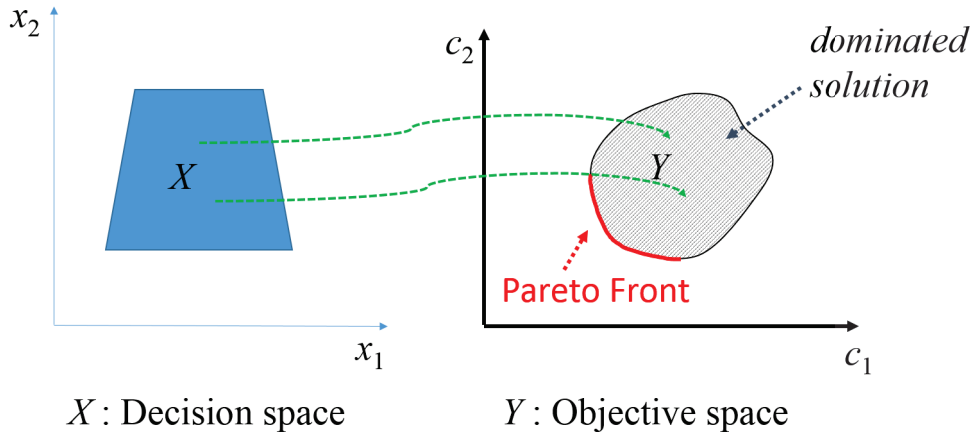

Figure 2.1: Definition of X and Y (case of two variables and two criteria)

Since in general there is no feasible solution that minimizes the two objectives simultaneously, we search for an acceptable trade-off between them. This compromise is defined by a dominance relation that corresponds to a partial order of the objective space (Giard and Roy, 1985). This dominance is such that no strictly better solution exists, and the equivalence between solutions is used to characterize Pareto efficiency, which replaces the notion of optimality in single objective optimization problems. The dominance relation is a binary relation $\mathcal{R}$ defined on a coherent of two criteria c_1 and c_2. Let x and x' be two solutions of the considered multi-objective problem: $x\mathcal{R}x'$ if and only if $c_k(x) \leq c_k(x')$ for $k = 1, 2$. It should be noted that $\mathcal{R}$ defines a partial preorder structure on X. The latter allows to define the concept of efficient solution. Therefore, a feasible solution $x^* \in X$ is called efficient if and only if there exists no solution $x \in X$ such that $x\mathcal{R}x^*$ and non $x^*\mathcal{R}x$. In other words, $x^* \in X$ is an efficient solution if there is no other feasible solution $x \in X$ that leads to an improvement on some criterion without simultaneous deteriorating at least another one.

Definition (Dominance relation). Let y and y' be two solutions of the objective space Y of a bi-objective problem. We say that y dominates y', denoted by $y' \succ y$, if and only if $y_k \geq y'_k$ for $k = 1, 2$, with at least one inequality being strict.

For multi-objective optimization problems, the order relation between solutions is partial (two solutions are not always comparable), so the concept of global optimality does not exist, and thus we talk about compromise solutions. Unlike the mono-objective case,

the goal here is to find a good compromise between the different optimized objective functions. The output of the optimization procedure consists of a set of solutions and the decision maker chooses the one among them that offers the most attractive compromise. The concept of Pareto optimality has appeared at the end of the 19^{th} century. It allows to find the compromise solutions. The optimal Pareto front is the set of all non-dominated solutions in the space of objectives (Allais, 1968).

Definition (Pareto efficiency). A solution $x \in X$ is Pareto efficient, if and only if there is no solution $x' \in X$ such that $c(x) \succ c(x')$. The efficient set is defined as $E^* = \{x \in X : x \; is \; Pareto \; efficient\}$, and the Pareto front as $F^* = \{c(x) : x \in E^*\}$.

An efficient solution $x^* \in E^*$ is also known as a *Pareto optimal* solution and its image $c(x^*)$ is called a *non–dominated* point. In other words, $x^* \in X$ is efficient if there is no other feasible solution $x \in X$ that leads to an improvement in some criterion without simultaneous deterioration in at least one other.

Definition (Weak Pareto optimum). A solution $x \in X$ is a weak Pareto optimum, also called weak efficient optimum, if and only if there is no solution $x' \in X$ such that $c_k(x') > c_k(x)$,$\forall k = 1, 2$. The Pareto set E^* is uniformly dominant if all points in F^* are weakly Pareto.

Definition (Strict Pareto optimum). A solution $x \in X$ is a strict Pareto optimum, also called efficient solution if and only if there is no solution $x' \in X$ such that $c_k(x') \geq c_k(x)$,$\forall k = 1, 2$ with at least one strict inequality.

The *ideal point* and *nadir point* are lower and upper bounds on non–dominated points. These points give an indication of the range of the values which non–dominated points can attain. They are often used as reference points in interactive methods in order to help the decision maker in his choice.

Definition (Ideal and Nadir points.) The point $y^I = (y_1^I, y_2^I)$, with $y_k^I = min\{c_k(x) : x \in X\}$ for $k = 1, 2$, is called the ideal point. The point $y^N = (y_1^N, y_2^N)$ with $y_k^N = max\{c_k(x) : x \in E^*\}$ for $k = 1, 2$, is called the nadir point.

Those points correspond to the optimal solution for each criterion if they are considered separately. They are reached if the objective functions are independents. If this condition is satisfied, the multi-objective problem can be considered as a set of mono-objective problems. In contrast, the nadir point corresponds to the worst values obtained for each objective function when the space of solutions is considered as the compromise area.

The ideal and nadir points for a nonconvex problem are shown in Figure 2.2. For a bi–objective optimization problem, the nadir point y^N can be determined as follows:

$$y_1^N = max\{c_1(x) : x \in E^*, c_2(x) \leq y_2^I\},$$

$$y_2^N = max\{c_2(x) : x \in E^*, c_1(x) \leq y_1^I\}.$$

Note that it is difficult to compute the nadir point y^N for multi-objective optimization problems with more than two objectives.

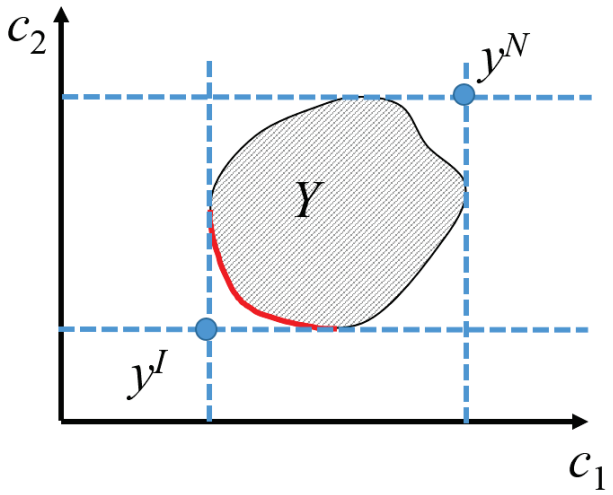

Figure 2.2: Ideal and Nadir points

Below we introduce Geoffrions definition of efficient solutions with bounded trade-offs, so called properly efficient solutions.

Definition (properly efficient solution, (Geoffrion, 1968)). A feasible solution $\widehat{x} \in X$ is called properly efficient, if it is efficient and if there is a real number $M > 0$ such that for all $k = 1, 2$ and $x \in X$ satisfying $c_k(x) < c_k(\widehat{x})$ there exists an index $h \neq k$ such that

$c_h(\widehat{x}) < c_h(x)$ and

$$\frac{c_k(\widehat{x}) - c_k(x)}{c_h(x) - c_h(\widehat{x})} \leq M.$$

The corresponding point $\widehat{y} = c(\widehat{x})$ is called properly non-dominated.

Note that the definition of efficient solutions and non-dominated points is not unique in the literature (see references (Chankong et al., 1981), (Miettinen, 1999) and table 2.4 in the book (Ehrgott, 2005) for more details).

In the following, we summarize the main results regarding (weakly) efficient solutions of bi–objective optimization problems.

Proposition 2.1 . *Let $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2$ be a weight vector, and suppose that $\widehat{x}$ is an optimal solution of the weighted sum optimization problem*

$$(BP(\lambda))\ min\{\lambda_1 c_1(x) + \lambda_2 c_2(x) : x \in X\}.$$

Then, the following statements hold

- *If $\lambda_1, \lambda_2 > 0$, then $\widehat{x}$ is an efficient solution.*
- *If $\lambda_1, \lambda_2 \geq 0$, then $\widehat{x}$ is a weak efficient solution.*
- *If $\lambda_1, \lambda_2 \geq 0$, and $\widehat{x}$ is a unique solution of $BP(\lambda)$ then $\widehat{x}$ is a strict efficient solution.*

(Geoffrion, 1968) establishes the relationships between properly non-dominated points and optimal points of weighted sum scalarizations with positive weights. He shows that these points coincide for convex sets.

Theorem 2.1 *(Geoffrion, 1968). Let $\lambda = (\alpha, 1 - \alpha)$ be a positive weight vector with $\alpha \in]0, 1[$. If $\widehat{x}$ is an optimal solution of $BP(\lambda)$ then $\widehat{x}$ is a properly efficient solution of BP. Moreover, if the feasible set X is convex and the objective functions c_1 and c_2 are convex, then $\widehat{x} \in X$ is properly efficient if and only if $\widehat{x}$ is an optimal solution of $BP(\lambda)$.*

2.2.2 Determining the set of efficient solutions

The most common methods for determining the set of efficient solutions are:

- ϵ–**constraints method**. This method, introduced by (Haimes et al., 1971), is the best known technique to solve multicriteria optimization problems. There is no aggregation of criteria. Instead, only one of the original objectives is minimized, while the others are transformed into constraints. This method is based on the minimization of one objective function (the most preferred or primary), and consider the other objectives as constraints bounded by some allowable level ϵ. Hence, a single objective minimization is carried out for the most relevant objective function, subject to additional constraint on the other objective function. More specifically, the bi–objective optimization problem BP is replaced by the following ϵ–constraint problem:

$$(BP^k(\epsilon))\ min\{c_k(x) : x \in X, c_h(x) \leq \epsilon_h, h \neq k\},$$

where $\epsilon \in \mathbb{R}$. The level ϵ is then altered to generate the entire Pareto optimal set. The ϵ–constraint method is easy to implement, but it requires a potentially high computational cost (many runs may be required).
The ϵ–constraint method is justified by the following properties.

Proposition 2.2 – *Let $\widehat{x}$ be an optimal solution of $BP^k(\epsilon)$ for some k. Then $\widehat{x}$ is weakly efficient.*

– *Let $\widehat{x}$ be a unique optimal solution of $BP^k(\epsilon)$ for some k. Then $\widehat{x}$ is a strict efficient (and therefore it is efficient).*

– *The feasible solution $\widehat{x} \in X$ is efficient if and only if there exists an $\epsilon \in \mathbb{R}$ such that $\widehat{x}$ is an optimal solution of $BP^k(\epsilon)$ for all $k = 1, 2$.*

(Vira and Haimes, 1983) provides a link between the weighted sum method and the ϵ–constraint method.

Theorem 2.2 *(Vira and Haimes, 1983).*

– *Suppose $\widehat{x}$ is an optimal solution of $BP(\lambda)$. If $\lambda_k > 0$, there exists $\widehat{\epsilon}$ such that $\widehat{x}$ is an optimal solution of $BP^k(\widehat{\epsilon})$, too.*

– *Suppose X is a convex set and c_1 and c_2 are convex functions. If $\widehat{x}$ is an optimal solution of $BP^k(\epsilon)$ for some k, there exists $\widehat{\lambda} \in \mathbb{R}^2$ with $\widehat{\lambda} \geq 0$ such that $\widehat{x}$ is optimal for $BP(\widehat{\lambda})$.*

- **Benson's Method (Benson, 1978)**: This method chooses an initial feasible solution $x' \in X$ and, if x' is not efficient, produces a dominated solution x'' as far from x' as possible. The distance from x' to x'' is computed as the sum of nonnegative deviations $\delta_k = c_k(x') - c_k(x'')$, $k = 1, 2$. More formally, the following problem is solved:

$$BM(x') \; max\{\delta_1 + \delta_2 : \delta_k = c_k(x') - c_k(x), x \in X, \delta_k \geq 0, k = 1, 2\}.$$

Proposition 2.3 *The feasible solution $x' \in X$ is efficient if and only if the optimal objective value of the problem $BM(x')$ is 0.*

- **Weighted Tchebycheff method** (Bowman Jr, 1976)
  A weighted Chebyshev norm of a point $y \in R^2$ with weight $\beta \in [0, 1]$ is defined as

$$||y||_\infty^\beta = max\{\beta|y_1|, (1 - \beta)|y_2|\}.$$

  Methods based on weighted Chebyshev norms select points with minimum weighted Chebyshev distance from the ideal point y^I. The following proposition is a well-known result for the weighted Chebyshev scalarization (Steuer and Choo, 1983).

Proposition 2.4 *If a point $y^* \in Y$ is a non–dominated point, then y^* is an optimal solution of the following problem*

$$max\{||y - y^I||_\infty^\beta : y \in Y\},$$

for some $\beta \in [0, 1]$.

The following result of (Bowman Jr, 1976) was originally stated for the efficient set but it is useful here to state the equivalent result for the Pareto set.

Proposition 2.5 . *If the Pareto set F^* is uniformly dominant, then any solution to the following problem*

$$\begin{cases} \max & z \\ s.t. & z \geq \beta(y_1^I - y_1), \\ & z \geq (1-\beta)(y_2^I - y_2), \\ & y \in Y. \end{cases}$$

where $\beta \in [0,1]$, corresponds to a Pareto front point.

- **Lexicographic Ordering** : In this method, the user is asked to rank the objectives in order of importance. The optimum solution is then obtained by minimizing the objective functions, starting with the most important one and proceeding according to the assigned order of importance of the objectives. Lexicographic optimization problems arise naturally when conflicting objectives exist in a decision problem but for reasons outside the control of the decision maker, the objectives have to be considered in a hierarchical manner. This method requires a definition of an order of the objectives, in decreasing importance, and thus, it defines a total order of the solutions. If some solutions have the same value for objectives c_k, objective c_h, with $k \neq h$, is used to break ties. For bi-objective problems, there are only two possible orderings of the objectives, and thus only two different scalarized problems can be defined, preventing the return of more than two solutions. The lexicographic or lexicographical order is a generalization of the alphabetical order of words, which is based on the alphabetical order of their component letters. Let $y, y' \in \mathbb{R}^2$, $y <_{lex} y'$ if $y_k < y'_k$ where $k = min\{h : y_h \neq y'_h\}$, the lexicographic order is total.

  Definition (lexicographically optimality): A solution $\widehat{x} \in X$ is lexicographically optimal if

$$c(\widehat{x}) <_{lex} c(x), \forall x \in X.$$

  The following proposition establishes the relationship between lexicographically optimal solution and efficient solution.

  Proposition 2.6 . *A lexicographically optimal solution is also an efficient solution.*

- **Max-Ordering Optimality** The max-ordering (MO) optimization problem is defined as follows:

$$(MO)\ \min\{max\{c_1(x), c_2(x)\} : x \in X\}.$$

Definition A feasible solution $\widehat{x} \in X$ is max-ordering optimal if there is no $x \in X$ such that

$$max\{c_1(x), c_2(x)\} < max\{c_1(\widehat{x}), c_2(\widehat{x})\}.$$

Proposition 2.7 *An optimal solution of the max-ordering problem is weakly efficient but not necessarily efficient.*

In the literature, there are other methods to characterize the set of non-dominated solutions as decision aid methods (ELECTRE methods and PROMETHEE methods (Figueira et al., 2005)). When the set of feasible solutions is discrete, fuzzy methods are used. These approaches will not be considered in this thesis. Interested readers can consult the book of (Collette and Siarry, 2003). For a broader survey of the field, we refer to recent bibliographies by (Ehrgott and Gandibleux, 2000), (Ehrgott and Gandibleux, 2002), (Ulungu and Teghem, 1994), and (T'kindt and Billaut, 2001).

2.2.3 The performance measures of optimal Pareto fronts

In a single objective optimization, the evaluation of the solution found by an algorithm is trivial. Indeed, it suffices to compare this solution with other ones of the same problem in terms of quality and time. However, in the multi-objective case, there is a set of non-dominated solutions and therefore they are not comparable. Hence, it is necessary to provide performance measures to assess the obtained results.

The goal of a multi-objective algorithm is to converge to the true Pareto front of a problem, which normally consists of a diverse set of points. In this comparative study, performance measures are taken into account for assessing the quality of solutions generated by the proposed algorithm such as:

1. Solution quality: the number or percentage of true Pareto-optimal solutions generated,

2. Solution efficiency: the computation time used to generate the Pareto-optimal set,

3. Generational distance,

4. Error ratio.

Three are normally the issues to take into consideration to design a good metric in this domain (Zitzler et al., 2000):

1. Minimize the distance of the Pareto front $\tilde{E}$ produced by an algorithm with respect to the true Pareto front E^*.

2. Maximize the spread of solutions found, so that we can have a distribution of solutions as smooth and uniform as possible.

3. Maximize the amount of elements of the Pareto optimal set found $\tilde{E}$.

Most of the performance metrics assume that the true Pareto front E^* of the multi-objective optimization under study is known. In this case, we can test the performance of a multi-objective algorithm by comparing the Pareto fronts produced by an algorithm $\tilde{E}$ with the true Pareto front E^* and then determine certain error measures that indicate how efficient is the algorithm analyzed.

The error ratio and generational distance metrics discussed next make this assumption. In order to compare two sets of non-dominated solutions, we need to identify the characteristics of these sets:

- The distance between the approximate set of non-dominated solutions $\tilde{E}$ and the true Pareto front E^*, which must be minimal.

- A good distribution of the solutions (uniform in most cases) is required.

- The magnitude of the obtained front must be maximized that is to say, for each objective, a wide range of values should be present.

In the literature, several metrics have been proposed (Knowles and Corne, 2002) and are divided into two sets. The first includes the metrics providing an absolute measurement with respect to Pareto optimal solutions E^* and the second contains the metrics that provide the measures associated between two approximate sets X' and X''.

The set E^* is known:

- **Error Ratio (ER):** This metric, proposed by (Van Veldhuizen, 1999a), reports the proportion of solutions in $\tilde{E}$ (approximation set of Pareto optimal solutions) that are not members of E^* (set of efficient solutions). Formally this metric is computed by the following equation:

$$ER = \frac{\sum_{x \in \tilde{E}} \mathcal{X}_E(x)}{|\tilde{E}|}$$

where $\mathcal{X}_E$ is the characteristic function relative to the set E, *i.e.*. $\mathcal{X}_E(x) = 0$ if $x \in E$ and $\mathcal{X}_E(x) = 1$ if $x \notin E$. Thus, when $ER = 0$ all the solutions in $\tilde{E}$ belong to E^*; but when $ER = 1$, this indicates that none of the points in $\tilde{E}$ are in E^*. The drawback of this metric is that it requires knowing the Pareto optimal set, which is often not easy to determine.

- **Generational Distance (GD):** This metric is proposed by (Van Veldhuizen and Lamont, 1998b). It estimates, for a given problem, how far is $\tilde{E}$ from the E^* of a problem, using the Euclidean distance (measured in objective space), between each solution and the nearest member of E^*. It is mathematically defined by the following equation:

$$GD = \frac{\sqrt{(\sum_{x \in \tilde{E}} d(x, x^*)^2)}}{|\tilde{E}|},$$

where x^* is the closest solution in E^* to that solution x, *i.e.*, $x^* = argmin\{d(x, x') : x' \in E^*\}$, being $d(x, x')$ the distance between the two solutions x and x'. It is easy to see that a value of $GD = 0$ indicates that $\tilde{E} = E^*$. Other similar metrics were proposed by (Rudolph, 1998; Schott, 1995; Zitzler et al., 2000; Van Veldhuizen and Lamont, 1998a). The problem with this metric is that only the distance to E^* is considered and not a uniform spread along the Pareto front.

- **Maximum Pareto front Error**(ME): The Maximum Pareto front Error (ME) compares two set of solutions (Van Veldhuizen, 1999b; Van Veldhuizen and Lamont, 1999). More precisely, it measures the largest minimum distance between each solution in $\tilde{E}$ and the corresponding closest solution in E^*, *i.e.*,

$$ME = max\{min\{(|c_1(y) - c_1(x)|^p + |c_2(y) - c_2(x)|^p)^{\frac{1}{p}} : y \in \tilde{E}\} : x \in E^*\}.$$

- **Spacing metric** (S): It describes the spread of the solution in $\tilde{E}$ (Coello et al., 2002; Schott, 1995). It measures the distance variance of neighboring solution in $\tilde{E}$ by computing the distance between two consecutive solutions of $\tilde{E}$:

$$S = \sqrt{\frac{1}{|\tilde{E}| - 1} \sum_{x \in \tilde{E}} (d(x) - \hat{d})^2},$$

where

$$d(x) = min\{|c_1(x) - c_1(y)| + |c_2(x) - c_2(y)| : y \in \tilde{E}\}$$

and

$$\hat{d} = \frac{\sum_{x \in \tilde{E}} d(x)}{|\tilde{E}|}.$$

In the case where $S = 0$, all members are spaced evenly apart. Note that this becomes important in the deception problems where all Pareto front solutions are equally spaced.

- **Maximum Spread** (MS): The metric measures the distribution of individuals in $\tilde{E}$ over E^*. It uses a statistical metric such as the chi-square distribution to measure spread along the Pareto front. This metric assumes that we know the true Pareto front of the problem:

$$MS = \sqrt{\frac{1}{|\tilde{E}|} \sum_{k=1}^{2} (\frac{\max_{x \in \tilde{E}} c_k(x) - \min_{x \in \tilde{E}} c_k(x)}{c_k^{max} - c_k^{min}})},$$

where c_k^{max} is the maximum value of the k^{th} objective function, *i.e.* $c_k^{max} = max\{c_k(x) : x \in X\}$, and c_k^{min} is the minimum value of the k^{th} objective function, *i.e.*, $c_k^{min} = min\{c_k(x) : x \in X\}$.

The set E^* is not known: In this case, the metrics allow to compare two approximate sets of non-dominated solutions. There are measures that evaluate the quality of one approximate set, that evaluate the convergence or the diversification or that evaluate both the convergence and the diversification simultaneously.

- **Attainment Surfaces**(AS): Draw a boundary in the objective space that separates the points that are dominated from those which are not (this boundary is called attainment surface). Perform several runs and apply standard non-parametric statistical procedures to evaluate the quality of the non-dominated vectors found.

It is unclear how we can really assess how much better is a certain approach with respect to others.

- **The coverage measures** (CM)**:** This metric measures the size of the objective value space area Y that is covered by a set of non–dominated solutions E^*. It combines the three issues previously mentioned (distance, spread and amount of elements of the Pareto optimal set found) into a single value. Therefore, sets differing in more than one criterion cannot be distinguished. (Zitzler et al., 2000) proposed another metric for comparing the performances of different algorithms. Suppose we want to compare the performance of two algorithms A' and A'', which generate the two sets X' and X'', respectively. More specifically, for each ordered pair (X', X'') subsets of solutions, *i.e.*, $X', X'' \subseteq X$, the metric CM corresponds to a value $CM(X', X'')$ in the interval $[0, 1]$ computed as follows

$$CM(X', X'') = \frac{|\{x \in X'' : \exists y \in X' \text{such that } y \prec x\}|}{|X''|}.$$

  If $CM(X', X'') = 1$, then all the solutions in X'' are dominated by or are equal to solutions in X'. If $CM(X', X'') = 0$, then none of the solutions in X'' are covered by the set X'.

There are other existing performance metrics given in (Sarker and Coello, 2002) and (Zitzler et al., 2003). However, we will only use these two previous metrics (AS and and CH).

2.3 Bi–objective methods

Solving a multi-objective combinatorial optimization problem consists of providing a set of Pareto solutions (as complete as possible) to the decision maker. Therefore, he can choose the most interesting solutions among them. Hence a question arises about the nature of these Pareto solutions and the techniques available to get them all. A study of the Pareto frontier should be performed.

2.3.1 Decision aid methods

Solving a multi-objective problem leads to the determination of a set of Pareto solutions. Therefore, it is necessary to integrate the decision maker, who is responsible for the final choice of the solution. Thus, before starting to solve of a multi-objective problem, we should select the kind of optimization method to be used. Indeed, one can divide the methods of solving multi-objective problems into three groups, depending on the time available to the decision maker. We can find the following families of methods(Knowles and Corne, 2004):

- ***A priori* methods**: In *a priori* optimization, the decision maker is consulted before the search process begins in order to build a mathematical model of his preferences that will be used in the search to evaluate all solutions. The best solution of this model is the result of process optimization without additional intervention of the decision maker. The main drawback of these methods is obvious: it is very hard for the decision maker to provide adequate models that determine which solutions he prefers without having an idea about the number of objectives to sacrifice for the benefit of others. This approach is fast, but we should take into account the time of modeling the compromise and the possibility that of the decision maker will be not satisfied with the obtained solution and repeat the search with another compromise.

- ***A posterior* methods**: In this second family of methods, we try to provide a set of good solutions to the decision maker. He can select among these solutions the one that seems most appropriate. These methods do not require the modeling of preferences of the decision maker and its presence throughout the search process. In this kind of method, two main phases are considered: the search phase of all Pareto optimal solutions (*i.e.* solving the optimization problem) and the choice phase of these solutions.

- **Interactive methods**: Here, the decision maker is involved in the process of finding solutions by answering different questions in order to guide the search. Interactive methods combine *a priori*i and *a posterior* methods in an iterative process, preferences and discover solutions. These methods are probably preferable to *a posterior* methods, since they limit the choices shown to the decision maker at any time, and

focus the search in a narrower space. This approach allows us to properly take into account the preferences of the decision maker, but requires its presence throughout the search process.

2.3.2 Two-phase methods

In this section, we present a two-phase method for the bi–objective problem. The two–phase methods have been widely used to solve bi–objective optimization problems. In the first phase, the set of supported efficient solutions is computed by solving a series of the single problems, while in the second phase other efficient solutions are computed. The two-phase method is a general method for solving bi–objective combinatorial problems ((Przybylski et al., 2008), (Przybylski et al., 2010), (Raith and Ehrgott, 2009b), (Ulungu and Teghem, 1995)). A description of the two-phase method for general multi-objective combinatorial optimization problems can be found in (Ulungu and Teghem, 1995).

The non–dominated points in F^* are partitioned into supported and non–supported points. The supported ones can be further subdivided into extreme and non-extreme points. Let us define the following notation:

$$Y^{\geq} = Y \oplus \{y \in \mathbb{R}^2 : y \geq 0\}),$$

where the operator $\oplus$ denotes the usual direct sum.

Definition (Supported efficient solution) A supported efficient solution x is an efficient solution of a BP problem, such that $x \in E^*$ and its image is situated on the boundary of the convex hull of the space $Y^{\geq}$, *i.e.*, $c(x)$ is on the boundary of $Conv(Y^{\geq})$. We denote E_S^* the set of supported efficient solutions.

The supported efficient solutions of the BP problem can be obtained by solving a series of mono–objective problems with a weighted sum of two objectives.

Proposition 2.8 *(Steuer, 1986). Given a bi–objective problem, a solution $\tilde{x} \in X$ is a supported efficient solution if it is an optimal solution of the following parametrized single objective problem:*

$$(BP(\lambda)) \quad min\{\tilde{c}(\lambda, x) = \lambda_1 c_1(x) + \lambda_2 c_2(x) : x \in X\},$$

where $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2_+ - \{0\}$.

The supported points are those found on the boundary of the convex hull of the objective space $Conv(Y^{\geq})$, while the non–supported efficient solutions are located in the interior of $Conv(Y^{\geq})$. The method to find supported efficient solutions is referred as the weighting method. The non–supported solutions can be found by the weighting method since their objectives are dominated by the convex combination of the supported efficient solution objectives. However, there is no known characterization of non–supported efficient solutions and a polynomial time algorithm for their computation is not known up to now. Figure 2.3 displays the set of efficient solutions where points A, B, C, and D are non–dominated supported extreme points; points E, F, and G are non–dominated non–supported points; and points H and I are dominated points.

In the bi–objective case, the supported extreme non–dominated points define a number of triangles in which non–supported non-dominated points may be found. These triangles are illustrated in the Figure 2.3.

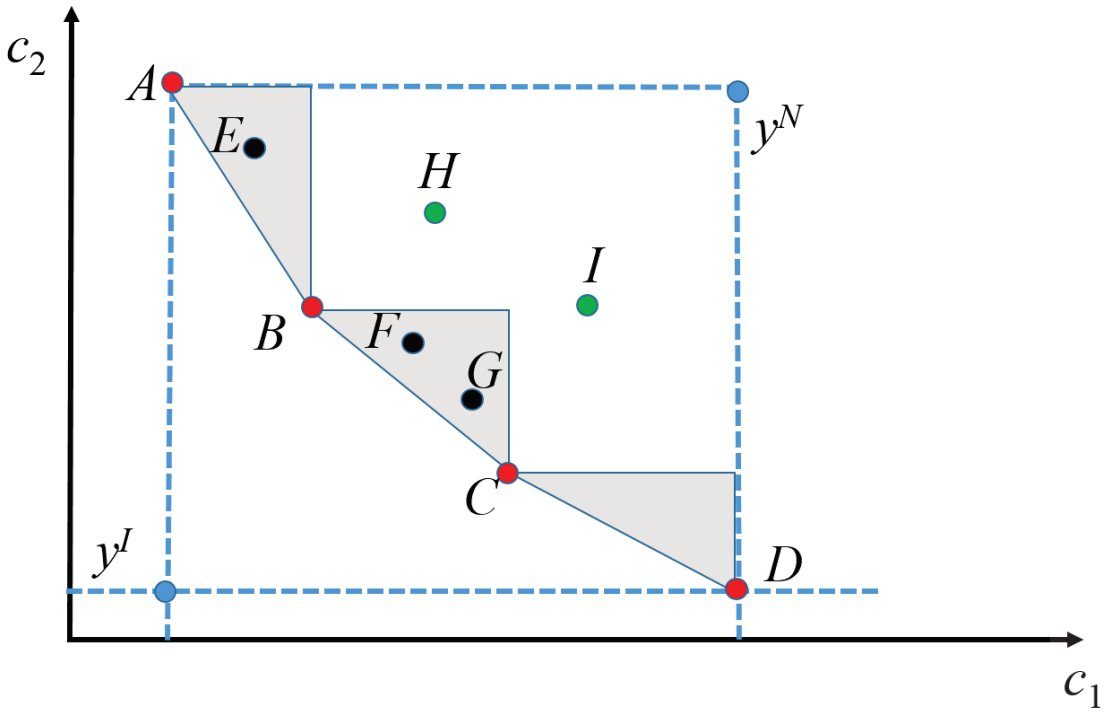

Figure 2.3: Supported and non supported efficient solutions

The Phase 1 is the non-inferior set estimation algorithm proposed by (Cohon, 1978),

and the idea was first applied to the bi-objective transportation problem (Aneja and Nair, 1979), where it was used to determine the set of supported extreme non-dominated points. The pseudo–code for Phase 1 is shown in Algorithm 1. The Phase 1 is initialized by finding the upper left and lower right points. The upper left point $x^{1,2}$ is an optimal solution of the lexicographic optimization problem

$$lexmin\{"c_1(x), c_2(x)" : x \in X\}.$$

The lower right point $x^{2,1}$ is computed by solving the lexicographic optimization problem

$$lexmin\{"c_2(x), c_1(x)" : x \in X\}.$$

In the case where the two points $c_1(x^{2,1}))$ and $c_2(x^{1,2}))$ coincide, the algorithm stops and returns only one supported solution. In Phase 1, the set of supported extreme non–dominated points is usually found using a parametric optimization problem $BP(\lambda)$. To define this problem $BP(\lambda)$, two supported extreme non-dominated points x^+ and x^- are used to define λ as the slope of the line connecting two non–dominated points $c(x^+)$ and $c(x^-)$. If the optimal solution x^* of $BP(\lambda)$ corresponds to a new supported extreme non-dominated point, then the parametric objective function value $\lambda_1 c_1(x^*) + \lambda_2 c_2(x^*)$ must be less than $\lambda_1 c_1(x^+) + \lambda_2 c_2(x^+)$ (see Figure 2.4). This process is finite since the weighted sums eventually find no solutions in interior of the convex hull $conv(Y)$.

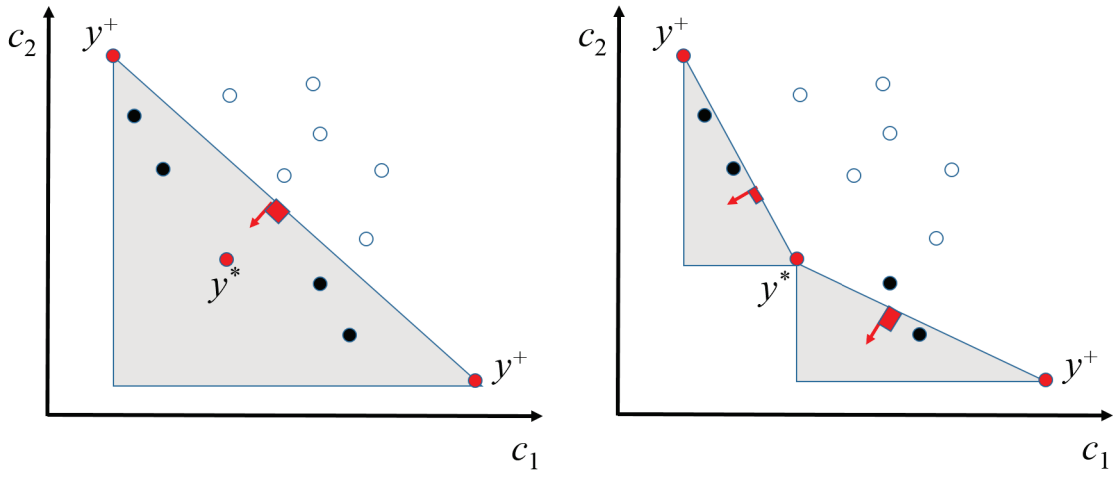

Figure 2.4: An iteration of dichotomic algorithm

Algorithm 1: Phase 1 – Finding Supported Extreme Non–dominated Points

Input: The two objectives c_k for $k = 1, 2$.

Output: E_S^* set of supported efficient solution set for the BP problem.

Let x^1 be an optimal solution of $BP(1, 0)$;

Let $x^{1,2}$ be an optimal solution of $BP(c_2(x^1), 1)$;

Let x^2 be an optimal solution of $BP(0, 1)$;

Let $x^{2,1}$ be an optimal solution of $BP(1, c_1(x^2))$;

Set $E_S^* = \{x^{1,2}, x^{2,1}\}$;

if $c(x^{1,2}) = c(x^{1,2})$ **then**
  Stop (only one non-dominated point);
Dichotomic($E_S^*, x^{1,2}, x^{2,1}$);

return E_S^*;

Algorithm 2: Dichotomic to Compute Supported Solutions

Input: E_S^* set of supported efficient solution set, $x^+, x^- \in E_S^*$ with

$$c_1(x^+) < c_1(x^-).$$

Output: E_S^* set of supported efficient solution set for the BP problem.

Let $\lambda_1 = c_2(x^+) - c_2(x^-)$ and $\lambda_2 = c_1(x^-) - c_1(x^+)$;

Solve $BP(\lambda_1, \lambda_2)$ with x^* optimal solution;

if $\lambda_1 c_1(x^*) + \lambda_2 c_2(x^*) < \lambda_1 c_1(x^+) + \lambda_2 c_2(x^+)$ **then**
  Set $E_S^* = E_S^* \cup \{x^*\}$;
  Dichotomic(E_S^*, x^+, x^*);
  Dichotomic(E_S^*, x^*, x^-);
return E_S^*;

Because supported and unsupported non–dominated points may exist, it is not possible, in general, to find all non–dominated points during the first phase. These points are found in Phase 2, which separately searches each triangle defined by the set of supported non–dominated points found in Phase 1. The set of supported non–dominated points F_S^* found in Phase 1 may be ordered according to increasing values of the first objective, and a given triangle is defined by two consecutive points in this list. The search for non–dominated points in this triangle may be carried out in several ways and it is often problem specific. There are many proposed methods in the literature (see, for instance,

(Tuyttens et al., 2000), (Visée et al., 1998), (Przybylski et al., 2008), and (Pedersen et al., 2008)). The best methods exploit some ranking algorithms to generate solutions to the weighted sum problem in the order of their objective value. This has been applied to many problems where efficient ranking algorithms exist, such as the multi-modal assignment problem (Pedersen et al., 2008), shortest path problem (Raith and Ehrgott, 2009) and finding network spanning trees (Steiner and Radzik, 2008).

2.3.3 Multi-objective evolutionary algorithms

Evolutionary algorithms (EAs) are inspired from the Darwinian theory of the natural evolution in order to provide solutions to optimization problems. The main difference between EAs and traditional techniques is that they are methods population based. The general framework of an EA can be presented as follows: initially a population of individuals that represent candidate solutions of the studied problem is generated. Then, the fitness of each individual is evaluated. Next, according to a fitness function, a subset of parents is selected from the initial population. At the next steps, some search operators such as recombination and mutation are applied to the selected parents for producing new generation. The whole process is repeated until satisfying a predetermined stopping criterion. Several algorithms have been developed in this area and the differences between them are defined by fitness evaluation, selection and search operators. The genetic algorithms (GA) where first proposed in (Holland, 1975) in 1975 and were later adapted by Goldberg for optimization problems (Goldberg, 1989). A GA is an evolutionary metaheuristic based on the process of natural evolution. Genetic Algorithms are the most popular techniques of evolutionary algorithms. They consist of a stochastic procedure based on mechanisms of natural selection, genetics and evolution.

Evolutionary algorithms are suitable to solve multi-objective optimization problems as they are able to find several Pareto-optimal solutions in a single evolutionary process, and may exploit similarities of solutions by recombination.

In each iteration, two parents are selected for reproduction, based on a fitness value, to create children. The created solutions can then mutate, in order to vary a bit from their parents and can replace some individuals of the current population. The first multi-objective GA was proposed by (Schaffer, 1985) in 1985 and it is called Vector

Evaluated Genetic Algorithm. Several multi-objective evolutionary algorithms were developed such as multi-objective Genetic Algorithm (C.M. Fonseca, 1993), Niched Pareto Genetic Algorithm (J. Horn, 1994), Random Weighted Genetic Algorithm (T. Murata, 1995), Non–dominated Sorting Genetic Algorithm (N. Srinivas, 1994), Strength Pareto Evolutionary Algorithm (E. Zitzler, 1999), Pareto-Archived Evolution Strategy (J.D. Knowles, 1985), Fast Non–dominated Sorting Genetic Algorithm (Deb et al., 2002), multi-objective Evolutionary Algorithm (R. Sarker, 2002), Rank-Density Based Genetic Algorithm (H. Lu, 2003). A survey on evolutionary multi-objective optimization can be found in (Coello, 2000).

Aggregating approach: In this approach, all objectives are combined into a single one via a weight vector. The weights define the relative importance of the objectives of the problem. The most used way is the weighted sum aggregation. In this case, it is usually assumed that the sum of weights is equal to one. However, in practice, it is difficult to predetermine precisely the weights. Moreover, this approach suffers the drawback of missing some members of the Pareto optimal set in the presence of concave Pareto front, regardless of the weights used (Coello Coello, 1996).

Vector Evaluated Genetic Algorithm (VEGA): It was proposed by Schaffer (1985). This method treats the objectives separately by presenting a particular selection operator. Given k objective functions and a population size P, k sub-populations of size P/k are generated, at each generation, by performing proportional selection according to each objective function in turn. These sub-populations are shuffled together to obtain a new population of size P and recombination and mutation are performed as usual.

Multi-Objective Genetic Algorithm (MOGA): This method was implemented by (C.M. Fonseca, 1993). Firstly, all non-dominated individuals are ranked to 1 and the rank of a dominated solution i, at generation g, is obtained according to the number of individuals by which it is dominated (q_i) as follows: $rank(i, g) = 1 + q_i$. Secondly, each individual should have fitness through one of two possible ways the Pareto ranking scheme of (Goldberg, 1989) or niche-formation methods.

Non-dominated Sorting Genetic Algorithm (NSGA): It was proposed by (N. Srinivas, 1994). The framework of this algorithm is the following. After generating the initial population, the latter is ranked using Pareto ranking. All non-dominated

solutions are classified in the first level, which represents the best front. Then, the next level consists of the non-dominated solutions after removing the first level and so on until all members of the population are classified. The next step consists to assign the highest fitness to the solutions belonging to the first front and then assign a progressively worse fitness to solutions of larger index front. Any solution i of the first non-dominated set is assigned to a fitness equal to the size of the population. In order to maintain diversity, the fitness is degraded according to the number of neighboring solutions. The sharing function is used for this aim. Therefore, the normalized Euclidean distance is computed for each pair of solutions (i, j) based on the decision variables of the problem. These distances are then used to compute the sharing function. Any solution j that has a distance greater than a constant sharing factor value σ_{share} from the i^{th} solution does not contribute to the sharing function value. Then, the niche count is obtained by the sum of the sharing function values. It gives the number of neighboring solutions of the i^{th} solution. The last step is to reduce the fitness of the i^{th} solution by its niche count and obtain the shared fitness values. For the solutions of the second front, initially, the assigned fitness is the minimum shared function value of the first front and then the shared function method is applied. The process continues in the same way for all the remaining fronts. Against MOGA, NSGA has presented lower performance and it seems to be more sensitive to the value of the sharing factor (Coello, 1996).

Niched Pareto Genetic Algorithm (NPGA): This algorithm was proposed by Horn et al. (1994). It differs from the previous methods in the selection phase. Two individuals are chosen at random and compared against a subset Q of the initial population. If one of them dominates all the solutions of Q and the other is dominated by at least one individual, then the non-dominated one is selected. If both individuals are either dominated or non-dominated, the selection is performed through the sharing function method based on the objective domain called equivalent class sharing.

Strength Pareto Evolutionary Algorithm (SPEA): This algorithm, proposed by (E. Zitzler, 1999) uses the concept of strength to locate and maintain a front of non-dominated solutions. The strength, computed as the ranking procedure in MOGA, indicates the degree to which a solution is dominated (strength). The fitness assignment of each individual is performed according to the strengths of all non-dominated solutions

that dominate it. The diversity of the algorithm is guaranteed by using the average linkage method.

Pareto Archived Evolution Strategy (PAES): It was developed by (J.D. Knowles, 1985) and uses the (1+1) evolutionary strategy. In fact, each parent produces one offspring via mutation and then they are compared to decide which one will be the parent of the next generation. If the offspring wins, it is added to an archive that contains the set of non-dominated solutions found so far.

Chapter 3

Metaheuristics for the bi–objective shortest path problem

3.1 Bi–objective shortest path problem

Shortest Path (SP) problems are among the most fundamental combinatorial optimization problems from an academic point of view. They play a major role in problems arising in many industrial applications such as transportation, logistic, routing, robot navigation, urban traffic planning, routing of telecommunications messages, approximating piecewise linear functions, optimal truck routing through given traffic congestion pattern, etc. Algorithms for solving these problems have been intensively studied since 1950's (see (Gallo and Pallottino, 1988; Ahuja et al., 1993; Cormen et al., 2001)) and still remain an active area of research. There are many contributions on a wide range of SP problems, including the combination of the following problems: point–to–point, single–source, single–target, all–pairs, directed or undirected graph, dense or space graph, arbitrary arc costs, k–shortest paths, dynamic problems, exact and approximate shortest paths, etc. Generally, shortest path problems are multi–objective in nature, and in many cases the choice of a route depends on various factors like time, cost and length. This choice may depend also on parking or maintenance facilities, accessibility, environmental impact, reliability and risk. For instance, in transportation of hazardous materials, a route is chosen considering the distance, the risk for the population, and the transportation costs (Erkut et al., 2007). Recently, (Wang et al., 2013) incorporates the travel time reliability and monetary cost

to design efficient routes in a road network.

In this chapter, we consider the Bi–objective Shortest Path (BSP) problem that aims to find efficient (non–dominated or Pareto–optimal) paths from a source vertex to a target vertex while optimizing two objectives simultaneously. Hereafter a standard mathematical formulation of the BSP problem is provided. Consider a directed graph $G = (N, A)$ where $N = \{1, 2, ..., n\}$ is the set of nodes and $A = \{(i, j) \in A : i, j \in N, i < j\}$ is the set of arcs joining nodes in N. For each arc $(i, j) \in A$, two costs (c^1_{ij} and c^2_{ij}) are associated. A path π in the graph G from node i_0 to node i_p is a sequence $\pi = (i_0, \ldots, i_p)$ such that $(i_h, i_{h+1}) \in A$ for $h = 0, \ldots, p-1$. Let $\Pi(i, j)$ be the set of all paths from i to j. The BSP problem from source $s \in N$ to target $t \in N$ can be formulated as:

$$min\{c(\pi) = (c_1(\pi), c_2(\pi)) : \pi \in \Pi(s, t)\},$$

where $c_k(\pi =< i_0, \ldots, i_p >) = \sum_{h=0}^{p-1} c_k(i_h, i_{h+1})$ for $k = 1, 2$. The standard mixed integer program formulation of the BSP problem may be stated as follows:

$$\begin{aligned}
&min\ c(x) = (c_1(x), c_2(x)) \\
&s.t. \\
&\textstyle\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = \begin{cases} 1 & if\ i = s, \\ 0 & if\ i \neq s, t, \\ -1 & if\ i = t, \end{cases} \\
&x_{ij} \in \{0, 1\}, \forall (i, j) \in A.
\end{aligned} \tag{3.1}$$

where x_{ij} is a binary variable that equals to 1 if edge (i, j) is selected in an efficient path, and 0 otherwise, and $c_k(x) = \sum_{(i,j)\in A} c^k_{ij} x_{ij}$ for $k = 1, 2$.

A simple graph is given in Figure 3.1, where three efficient paths from node s to node t are shown in color, namely (s, a, d, t), (s, b, c, d, t) and (s, b, c, t) with total costs $(7, 5)$, $(5, 7)$ and $(6, 6)$ respectively.

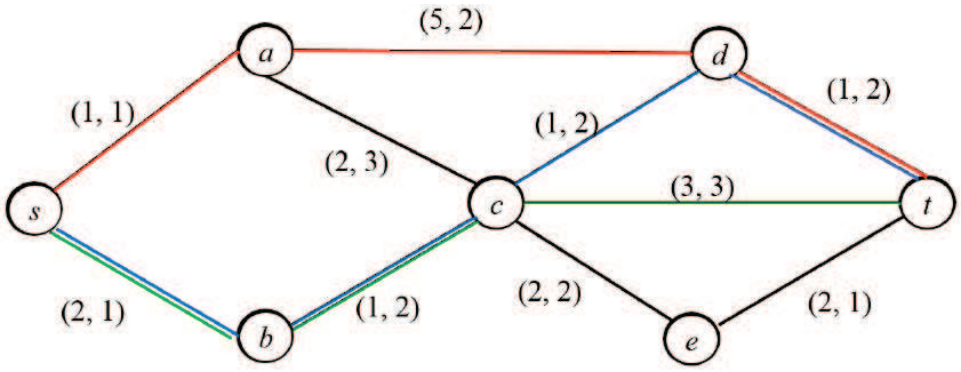

Figure 3.1: A small example with 3 efficient solutions.

The BSP problem is well known to be NP–hard (Serafini, 1987). (Hansen, 1980) shows that the number of non-dominated paths may increase exponentially with the number of nodes n. The BSP problem is more difficult to solve than the corresponding single criterion problem. BSP problems are often treated independently of multi–objective SP problems with more than three objectives because of their special structure, the difficulty grows strongly with the number of objectives ("Three is more than two plus one." (Ehrgott and Gandibleux, 2002)).

Several exact and approximative methods for solving the bi–objective shortest path problem are proposed in the literature. (Skriver, 2000), (Ehrgott and Gandibleux, 2002) and (Raith and Ehrgott, 2009a) give surveys on existing exact methods proposed for BSP problem. They classify these approaches into two categories: enumerative approaches such as label correcting ((Daellenbach and De Kluyver, 1980), (Corley and Moon, 1985), (Brumbaugh-Smith and Shier, 1989), (Skriver and Andersen, 2000), (Guerriero and Musmanno, 2001), (Sastry et al., 2003)), and label setting ((Hansen, 1980), (Martins, 1984), (Tung and Chew, 1988), (Tung Tung and Lin Chew, 1992), (Martins and Santos, 1999)) or ranking methods ((Namorado Climaco and Queiros Vieira Martins, 1982)). The second category includes the two phase method or parametric approach ((Mote et al., 1991), (Raith and Ehrgott, 2009a)). The interested reader is referred to (Raith and Ehrgott, 2009a) for a more detailed discussion on the related literature. The most recent work, see for example (Xie and Travis Waller, 2012), proposes an approximate polynomial–time parametric procedure for BSP problems, to find the full or near full set of Pareto–optimal paths.

Despite their efficiency to solve the considered problem for a small to medium size

instances, these methods cannot be used to solve real problems where instances are very large. The reason is that none of these instances can be solved in a reasonable time. Thus, during the last few years, multi-objective evolutionary algorithms have attracted the interest of many researchers from the field of combinatorial optimization. For instance, (Gen and Lin, 2005) use a multi-objective hybrid genetic algorithm to solve the network design problem in order to minimize the cost and maximize the flow simultaneously.

(Mooney and Winstanley, 2006) developed an Evolutionary Algorithm to solve the BSP problem on networks with multiple independent criteria. (Pangilinan and Janssens, 2007) presented a review of recent issues regarding the resolution methods of the BSP problem. (Lin and Gen, 2009) proposed new multi–objective hybrid genetic algorithms for three kinds of bicriteria network design models. (Ghoseiri and Nadjari, 2010) studied and solved the BSP problem using an algorithm based on multi-objective Ant Colony Optimization. More recently (Mohamed et al., 2010) presented a bi-objective genetic algorithm approach based on vector evaluation to solve the bi-objective shortest path problem, and (Liu et al., 2012) investigated an oriented spanning tree based genetic algorithm for the multi–criteria shortest path problem.

In this chapter, we present a two phase method for the BSP problem. The two–phase methods have been widely used to solve bi–objective optimization problems ((Przybylski et al., 2008), (Przybylski et al., 2010), (Raith and Ehrgott, 2009b), (Ulungu and Teghem, 1995)). In the first phase, the set of supported efficient solutions is generated by solving a series of the single SP problems, while in the second phase an approximation of the set of unsupported solutions is investigated. We adopt the dichotomic approach for the first phase of the two phase algorithm proposed by (Raith and Ehrgott, 2009b) (see also (Aneja and Nair, 1979), (Cohon, 1978)). The next sections describe the metaheuristics used to approximate the non-supported efficient solutions set: cost perturbation method, path–relinking, genetic algorithm and finally an hybrid approach combining these three metaheuristics.

3.2 Cost perturbation method

The first proposed heuristic is a new method based on the cost perturbation. In this method, small perturbations are performed on the costs of the considered bi–objective problem. The new weighted cost $\tilde{c}_{ij}$ of variable x_{ij} is as follows:

$$\tilde{c}_{ij}(\lambda) = \lambda_1(c^1_{ij} + \epsilon^1_{ij}) + \lambda_2(c^2_{ij} + \epsilon^2_{ij}),$$

where λ_1 and λ_2 are respectively the weights of objective c_1 and c_2. The ϵ^k_{ij}'s are random values in $[-\epsilon^k_{max}, \epsilon^k_{max}]$.

For each disturbed cost, the single SP problem with this weighted sum problem is solved using the Dijkstra algorithm. These variations will allow us to slightly move to the Pareto front and therefore find a non-supported efficient solution. The cost perturbations heuristic starts with the set of supported efficient solutions generated at the first phase and the process is repeated for a fixed number of iterations. At each iteration, the optimal solution of the current disturbed single SP problem is added to the current list of non-dominated solutions if it is non dominated by those solutions. This approach is connected with the noising method proposed by (Charon and Hudry, 1993). The noising method applied to single objective optimization tries to generate local optima, while if it is applied to multi–objective optimization problem it generates non–dominated ones.

Algorithm 3 summarizes the Cost Perturbations Heuristic (CPH). Let S be a subset of efficient solutions and x a feasible solution, and let $Dom(x, S) = \{y \in S : x \textit{ dominates } y\}$.

Algorithm 3: Cost perturbation heuristic

Input: The two objectives c_k and parameters ϵ^k_{max}, for $k = 1, 2$; and $Iter_{max}$ the maximum number of iterations.

Output: $\tilde{E}$ Approximation of efficient solution set for the BSP problem associated with G.

Compute the set of supported efficient solutions E_S;

Set $\tilde{E} = E_S = \{(\pi, \lambda) : \pi \text{ is an optimal solution of } SP(\lambda)\}$;

Set $iter = 1$;

while $iter \leq Iter_{max}$ **do**

  Select randomly $(\pi, \lambda) \in \tilde{E}$;

  For $k = 1, 2$, choose randomly $\epsilon^k_{ij} \in [-\epsilon^k_{max}, \epsilon^k_{max}]$;

  For all $(i, j) \in A$, set $\tilde{c}(\lambda)_{ij} = \lambda_1(c^1_{ij} + \epsilon^1_{ij}) + \lambda_2(c^2_{ij} + \epsilon^2_{ij})$;

  Solve the $SP(\lambda)$ problem related to the graph G with the cost function $\tilde{c}(\lambda)$, obtaining solution π;

  if $Dom(\pi, \tilde{E}) = \emptyset$ **then**

    Add π to the set $\tilde{E}$, *i.e.*. $\tilde{E} = \tilde{E} + \{\pi\}$;

  Set $iter = iter + 1$;

return $\tilde{E}$;

3.3 Bi–objective path relinking

Path-Relinking (PR) was originally proposed by (Glover, 1977) as an evolutionary metaheuristic to explore trajectories connecting good solutions obtained by heuristic methods such as tabu search or scatter search. Path-relinking generalizes the combination method of scatter search. It generates paths between two selected solutions in the neighborhood space. In single objective optimization, path relinking is used as an intensification strategy, while here we use it to generate non–supported efficient solutions. In order to deal with the BSP problem, we present in this section a new Bi–objective Path Relinking (BPR) heuristic to generate an approximation of the non–supported efficient solution set $E_{\bar{S}}$. This BPR heuristic takes as an argument the set of supported solutions E_S generated in the first phase. At each iteration, two solutions are generated, as usually done by path relinking. One is called the *initial* solution, and the other is called the

guiding solution. The initial solution π is selected randomly from the set of the support efficient solutions, $\pi \in E_S$ with its associated weight $\lambda = (\lambda_1, \lambda_2)$, *i.e.*. π is an optimal solution of the parametrized single objective problem $SP(\lambda)$. The guiding solution π' is also selected randomly from $\pi' \in E_S$ with its associated weighted $\lambda' = (\lambda'_1, \lambda'_2)$, *i.e.*. π' is an optimal solution of $SP(\lambda')$.

Once the two solutions are selected, a Local Search (LS) procedure is applied. The LS procedure takes as arguments the initial solution π and the weights λ' of the guiding solution π'. At each iteration, two nodes i and j belonging to the path π are selected randomly, then the sub–graph H_{ij} of G is constructed, containing all the paths between nodes i and j in G. After, the single $SP(\lambda')$ problem is solved related to H_{ij} to obtain a shortest path $\pi'(i,j)$. Next, the sub–path $\pi(i,j)$ is replaced by $\pi'(i,j)$ in the shortest path π and if this new path is added to the efficient set $\tilde{E}$, it is non dominated by one of the already efficient solutions generated, (*i.e.*, if yes, a new non–supported efficient solution is generated).

Algorithm 4: Bi–objective path relinking

Input: The two objectives c_k and E_S the set of supported efficient solutions for the BSP problem associated with G.

Output: $\tilde{E}$ Approximation of efficient solution set for the BSP problem associated with G.

Set $\tilde{E} = \{(\pi, \lambda) : \pi \in E_S$ and λ the associated weight $\}$;

while *stopping criterion is not met* **do**

- select randomly $(\pi, \lambda) \in \tilde{E}$;
- select randomly $(\pi', \lambda') \in \tilde{E}$;
- $E' =$ Local Search$(\pi, \lambda', \tilde{E})$;
- $\tilde{E} = Dom(E', \tilde{E})$;

return $\tilde{E}$;

Algorithm 5: Local Search

Input: A supported efficient solution π, a weight λ, and an approximation $\tilde{E}$ of efficient solutions set.

Output: E' a subset of efficient solutions set for the BSP problem associated with G.

Set $E' = \emptyset$;

while *stopping criterion is not met* **do**

Select randomly two nodes i and j on the path π;

Construct the sub–graph H_{ij} containing all paths between i and j;

Let $\pi'(i,j)$ be a shortest path between i and j in H_{ij} with the cost $\tilde{c}(\lambda')$;

Replace the sub–path $\pi(i,j)$ by $\pi'(i,j)$ in π;

if $Dom(\pi, \tilde{E}) = \emptyset$ **then**

Add π to the set E' and $\tilde{E}$, *i.e.*. $E' = E' + \{\pi\}$, $\tilde{E} = \tilde{E} + \{\pi\}$;

return E';

This process is repeated 20 times at each iteration, meaning, 20 new generated solutions by iteration.

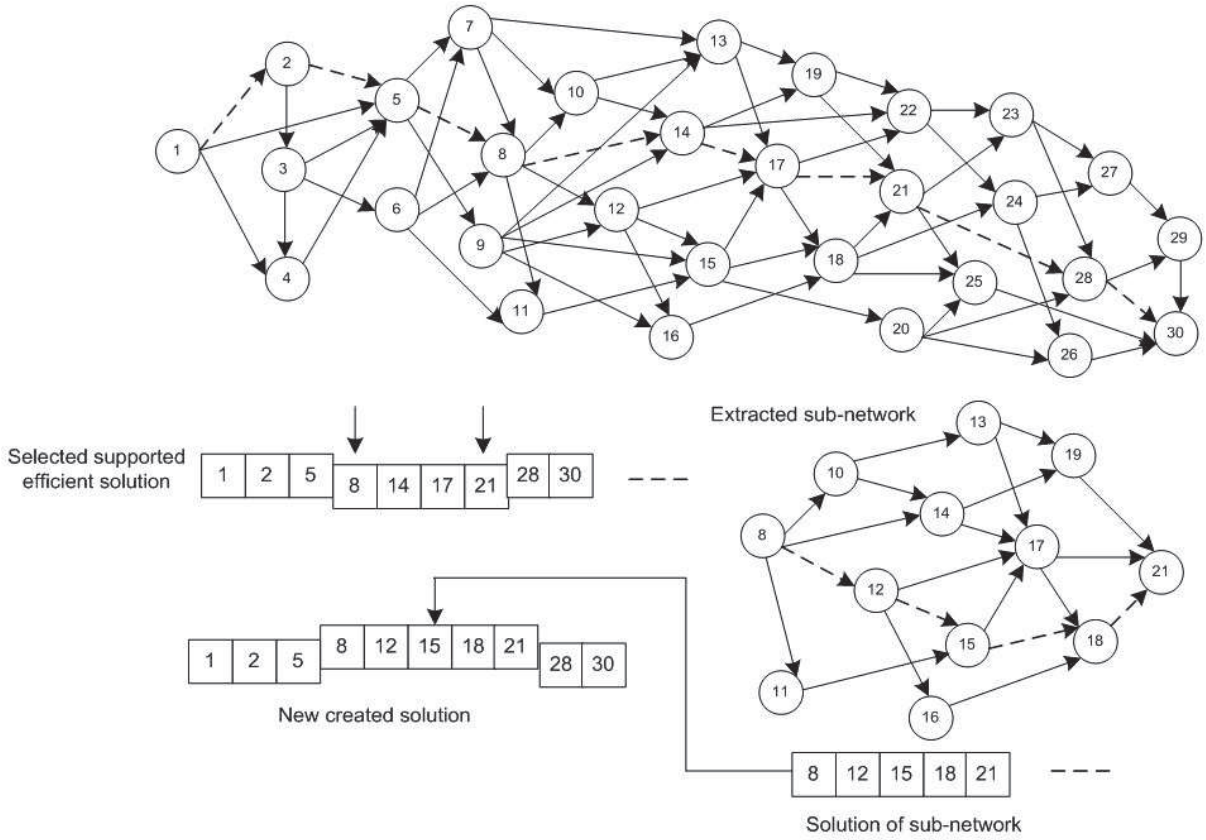

Figure 3.2: A non–supported efficient solution generated by local search

Figure 3.2 illustrates one iteration of the LS procedure to generate a non–supported efficient solution. Let $\pi = (1-2-5-8-14-17-21-28-30)$ be the supported efficient solution chosen randomly from the set E_S. Let $i = 8$ and $j = 21$ the two nodes of π selected randomly. The sub-graph H_{ij} of G containing all paths between i and j is drawn bellow G. Now, the single $SP(\lambda)$ is solved on the sub–graph H_{ij} to obtain the shortest path $\pi'(i,j) = (8-12-15-18-21)$. Then, this subpath replaces $\pi(i,j)$ in π. Hence, the generated path by local search is $\pi = (1-2-5-8-12-15-18-21-28-30)$.

3.4 Genetic algorithm

In this section, we propose a Genetic Algorithm (GA) for the bi-objective shortest path problem to generate an approximation of the set of non-supported efficient solutions. GA operates with a collection of solutions, called a population. The population is initialized as the set of supported efficient solutions. GA uses two operators to generate new solutions from the existing ones: crossover and mutation operators. The crossover operator is the

most important operator of GA. In crossover, generally two solutions, called parents, are combined together to form new solutions, called offspring. The crossover operator may generate a new non–supported solution by solving a single SP problem combining the two SP problems associated to parents. The parents are selected from the current population. The mutation operator introduces random changes into characteristics of solutions. In our implementation the mutation is not applied.

Algorithm 6: Genetic algorithm

Input: The two objectives c_k and E_S the set of supported efficient solutions for the BSP problem associated with G.

Output: $\tilde{E}$ Approximation of efficient solution set for the BSP problem associated with G.

Set $\tilde{E} = \{(\pi, \lambda) : \pi \in E_S$ and λ the associated weight $\}$;

while *stopping criterion is not met* **do**

  Select randomly two supported solutions $(\pi', \lambda'), (\pi'', \lambda'') \in \tilde{E}$;

  Set $E' = \text{Crossover}((\pi', \lambda'), (\pi'', \lambda''), \tilde{E})$;

  $\tilde{E} = Dom(E', \tilde{E})$;

return $\tilde{E}$;

- **Solution representation:** For routing problems various encoding methods have been proposed in the literature (see for example, (Gen et al., 1997), (Inagaki et al., 1999), and (Ahn and Ramakrishna, 2002)). In the proposed GA, a solution is represented by a chromosome as proposed by (Ahn and Ramakrishna, 2002). Since a solution of the BSP problem is a path, a variable-length representation is adopted for each chromosome, which consists of sequences of nodes belonging to this path.

  Each locus of the chromosome represents an order of nodes in the path, and the length of the chromosome is variable but should not exceed the number of nodes. The first and last loci are reserved respectively to the source and the target nodes. Figure 3.3 shows a solution $\pi = (1 - 3 - 5 - 8 - 10)$ where $s = 1$ and $t = 10$.

- **Initial population:** The initial population is defined as the set of supported efficient solutions E_S^* determined in the first phase of the two phase method. This set E_S^* is generated by the dichotomic approach described in Chapter 2. For each

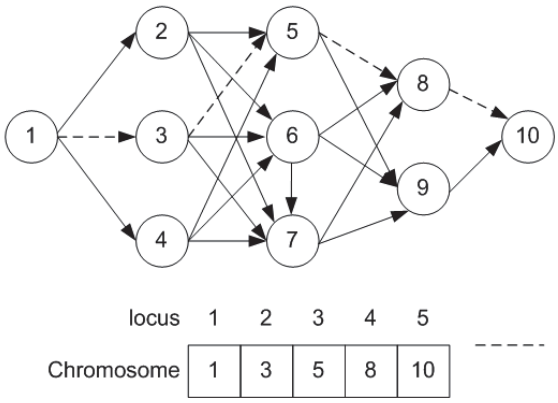

Figure 3.3: Chromosome representation.

solution (path) π in the initial or the current population. We keep its associated weight vector $\lambda(\pi)$.

- **Crossover operator:** The crossover operator in genetic algorithms allows to generate new solutions to add to the current population. The crossover operator implemented for the BSP problem is described in Algorithm 7. The crossover operates on two parents π' and π'', chosen randomly from the current population. Those parents correspond to two efficient solutions π' and π'' with their associated weights λ' and λ'' respectively. We define a set of blocks relating to π' and π''. A *block* of π' and π'' is a subpath π of π' and π'' and if we extend π from left or right relatively to π' or π'', π will not be a subpath of π' and π''. For each block π with the extremities i and j, we construct a sub–graph H_{ij} containing all possible paths between i and j. Then the $SP(\lambda)$ problem is solved relatively to H_{ij} to obtain a shortest path $\pi'(i,j)$, where the weight λ is a linear combination of the two weights λ' and λ''. Next, the sub–path $\pi'(i,j)$ is replaced by $\pi(i,j)$ in π' and the same happens for π''. If those paths are not dominated, we add them to the approximate set of efficient solutions.

Algorithm 7: Crossover Operator

Input: The two efficient solutions π' and π'' with their associated weights λ' and λ'' respectively, and an approximation $\tilde{E}$ of efficient solutions set.

Output: E' a subset of efficient solutions set for the BSP problem associated with G.

Set $E' = \emptyset$;

Set $B = \{\pi : \pi \text{ is a block of } \pi' \text{ and } \pi''\}$;

for $\pi \in B$ **do**

Select randomly two nodes i and j on the subpath π;

Construct the sub–graph H_{ij} containing all paths between i and j;

Choose randomly $\delta \in [0, 1]$ and set $\lambda = \delta \times \lambda' + (1 - \delta) \times \lambda''$;

Let $\pi'(i, j)$ be a shortest path between i and j in H_{ij} with the cost $\tilde{c}(\lambda)$;

Replace the sub–path $\pi(i, j)$ by $\pi'(i, j)$ in π';

if $Dom(\pi', \tilde{E}) = \emptyset$ **then**

Add π' to the set E' and $\tilde{E}$, *i.e.*. $E' = E' + \{\pi'\}$, $\tilde{E} = \tilde{E} + \{\pi'\}$;

Replace the sub–path $\pi''(i, j)$ by $\pi'(i, j)$ in π'' ;

if $Dom(\pi'', \tilde{E}) = \emptyset$ **then**

Add π'' to the set E' and $\tilde{E}$, *i.e.*. $E' = E' + \{\pi''\}$, $\tilde{E} = \tilde{E} + \{\pi''\}$;

return E';

3.5 Hybrid genetic algorithms

In this hybrid GA we apply a local search to each offspring generated by the crossover operator. This local search is the same as the crossover operator. The crossover operator in genetic algorithms allows to generate new solutions to add to the current population. The crossover operator implemented for the BSP problem is described in Algorithm 7. The crossover operates on two parents π' and π'', chosen randomly from the current population. Those parents correspond to two efficient solutions π' and π'' with their associated weights λ' and λ'' respectively. We define a set of blocks relative π' and π'',

A *block* of π' and π'' is a subpath π of π' and π'' and if we extend π from left or right relatively to π' or π'', π will be not a subpath of π' and π''. For each block π with the extremities i and j, we construct a sub–graph H_{ij} containing all possible paths between i and j. Then the $SP(\lambda)$ problem is solved relatively to H_{ij} to obtain a shortest path $\pi'(i,j)$, where the weight λ is a linear combination of the two weights λ' and λ''. Next, we replace the sub–path $\pi(i,j)$ by $\pi'(i,j)$ in π' and the same for π''. If those paths are not dominated we add them to the approximate set of efficient solutions.

Due to the potential and characteristics of each of the previously proposed heuristics, we propose a combination of them to take advantage from the best properties of each one. This originates the heuristic HGA.

Algorithm 8: Hybrid genetic algorithm

Input: **The two objectives** c_k and E_S the set of supported efficient solutions for the BSP problem associated with G.

Output: $\tilde{E}$ Approximation of efficient solution set for the BSP problem associated with G.

Set $\tilde{E} = \{(\pi, \lambda) : \pi \in E_S \text{ and } \lambda \text{ the associated weight } \}$;

while *stopping criterion is not met* **do**

  Select randomly two supported solutions $(\pi', \lambda'), (\pi'', \lambda'') \in \tilde{E}$;

  Set $E' = \text{Crossover}((\pi', \lambda'), (\pi'', \lambda''), \tilde{E})$;

  for $(\pi, \lambda) \in E'$ **do**

    $E'' = \text{Local Search}(\pi, \lambda, \tilde{E})$;

    $\tilde{E} = Dom(E'', \tilde{E})$;

return $\tilde{E}$;

3.6 Computational results

In order to evaluate the efficiency of the proposed heuristics (CPH, BPR, GA and HGA), several numerical experiments were conducted. The purpose of this set of experiments is to investigate the behavior of these heuristics, and evaluate their efficiency for networks with large sizes and different densities. The results of the proposed heuristics on a set of instances are compared with those of the *Brumbaugh-Smith* (BRUM) algorithm. This last

one returns the best route for every visited node. To our knowledge, this algorithm is one of the most effectient algorithms among all the bi–objective labeling methods available in the literature. The algorithms were implemented with language C++. We used a personal computer with a processor Intel Corei7 quadri-core, CPU 3 GHz, 16 Go RAM Windows 7.

Instances of $n \in \{100, 200, 300, 500\}$ nodes are considered. For each network size, 5 random networks are generated. In addition, 60 large sized problems are randomly generated with $n \in \{1000, 3000, 4000, 5000\}$ nodes.

Table 3.1 gives the results of the proposed heuristics (CPH, BPR and HGA) for the set of small size instance and different density networks. The density of a graph $G = (N, A)$ is defined as $d = \frac{2m}{n(n-1)}$ where $m = |A|$. It gives respectively, in the first and second column, the size and the number of edges of networks. The third and fourth columns illustrate the computational time (in seconds) and the number of Pareto solutions of the labeling algorithm (BRUM) and the remainder of the table gives the average consumed time (in seconds), the average number of non-dominated solutions and the average percentage of the true Pareto solutions found by each heuristic.

n	m	**BRUM**		**CPH**			**BPR**			**HGA**		
		Times	$\lvert\tilde{E}\rvert$	**Times**	$\lvert\tilde{E}\rvert$	**% Best sol**	**Times**	$\lvert\tilde{E}\rvert$	**%Best sol**	**Times**	$\lvert\tilde{E}\rvert$	**% Best sol**
$d = 5$												
100	501.6	0.038	16.8	0.069	16.6	**96.89**	0.068	16	93.03	0.069	15.6	88.12
200	995.8	0.101	15.8	0.141	15.6	**98.67**	0.115	14.8	91.89	0.132	15.4	89.28
300	1541.4	0.355	20.4	0.189	20.2	**99.2**	0.223	19.4	93.84	0.205	18.4	86.69
500	2549.6	0.909	32.6	0.468	31.6	**97.21**	0.436	29.8	90.67	0.395	28.8	86.81
$d = 10$												
100	985.4	0.105	17.8	0.090	17.8	**97.65**	0.086	17.2	94.92	0.093	16.6	86.98
200	2029.2	0.200	13.4	0.184	13.4	**100**	0.158	12.2	91.33	0.145	11.2	82.07
300	3009.8	0.343	13.8	0.248	13.8	**100**	0.287	13	94.76	0.267	13.6	97.71
500	5087.4	1.006	16.4	0.795	16.4	**100**	0.829	16.2	96.4	0.656	15.4	90.22
$d = 20$												
100	2031.2	0.121	8	0.103	7.8	**97.5**	0.107	7.6	95.56	0.102	7.4	91.56
200	4068.6	0.603	16	0.255	16	**100**	0.282	15.4	96.54	0.261	14.6	91.96
300	5994.6	0.967	15.4	0.338	15	**97.78**	0.410	14	90.06	0.387	13.4	84.91
500	10459.2	1.158	10.2	1.099	10	98.82	1.132	10.2	**100**	0.891	10	98.82

Table 3.1: Average number of non-dominated solution in large size networks.

From Table 3.1, we can observe that the three heuristics provide good results. We

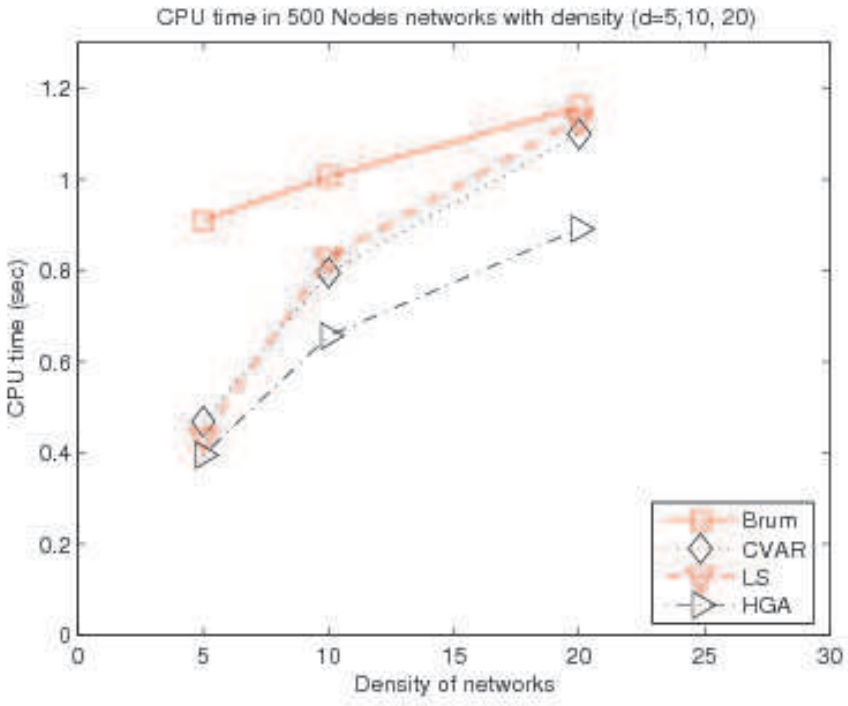

Figure 3.4: Computation time in 500 nodes network over different densities.

observed that the CPH heuristic far exceeds other heuristics having a percentage of the true Pareto solutions that varies between 96 and 100%. In a second position, there is the BPR heuristic wich has a percentage of true Pareto solution between 90 and 100%, and in the last position, the HGA gives a percentage between 82 and 98% of true Pareto solutions. Concerning the computation time used to generate the Pareto-optimal set, it is the same as the one consumed by the labeling algorithm and it does not exceed 0.13 seconds for the instance of 100 and 200 nodes; 1 second for the instances of 300 nodes and 1.16 seconds for the instances of 500 nodes. Figure 3.4 illustrates the variation of time consumed over the variations of networks density (example for networks of 500 nodes).

Table 3.2 summarizes the results for experimental analysis on the large–scale instances.

The information given in this table is related to specific details about instances (number of nodes (n), number of edges (m) and the density (d) of networks), the consumed computation CPU time and the number of Pareto optimal solutions generated by the labeling algorithm (BRUM). In the next columns we present the number of supported non-dominated solutions of each instance. The remaining columns of the table

are devoted to the results found by the proposed heuristics (CPH, BPR, GA and HGA). We list the number of solutions and the consumed computation time for each proposed heuristic. To analyze the efficiency of these heuristics, some performance measures are used and illustrated. These performance measures are the generational distance (GD), the error ratio (ER) and the percentages of the true Pareto solutions found.

A more comprehensive comparison is made by putting review the various results found by the labeling and the proposed heuristics. From the results illustrated in Table 3.2, and considering the generational distance and error ratio, we can see that the results of the proposed algorithms are of a very good quality. Concerning the error ratio, it does not exceed the average value 0.058 in the three heuristics and 0.022 in the combination of the three heuristics (HGA). The generational distance does not exceed the average value 1.17 in the three heuristics and is at worst 0.474 in GA. This indicates that the generated Pareto front has a very small distance to the true Pareto front.

Table 3.2: Average number of non-dominated solution in large size networks.

		E_S	**BRUM**		**HGA**					**CPH**					**BPR**					**GA**				
n	m	$\lvert\tilde{E}\rvert$	$\lvert\tilde{E}\rvert$	**CPU**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**
$d = 5$																								
1000-1	4701	10	70	11.91	69	4.64	0.092	0.015	97.14	69	3.26	0.164	0.029	95.71	57	4.87	2.880	0.140	70	33	3.79	2.727	0.091	42.89
1000-2	4685	15	66	11.57	64	4.65	0.000	0.000	96.97	65	3.29	0.000	0.000	98.48	59	4.83	0.000	0.000	89.39	52	3.79	0.000	0.000	78.79
1000-3	4659	17	60	6.60	60	4.76	0.000	0.000	100	60	3.31	0.000	0.000	100	55	4.67	0.509	0.036	88.33	52	3.52	0.000	0.000	86.68
1000-4	4685	9	58	6.95	58	4.67	0.000	0.000	100	57	3.32	0.000	0.000	98.28	48	4.77	0.495	0.063	77.59	38	3.04	0.000	0.000	65.52
1000-5	4714	10	51	8.36	51	4.73	0.000	0.000	100	50	3.30	0.278	0.020	96.08	43	4.84	0.323	0.023	82.35	32	3.20	0.828	0.063	58.82
Average	4688.8	12.2	61	9.07	60.4	4.69	0.018	0.003	98.822	60.2	3.29	0.088	0.010	97.71	52.4	4.80	0.841	0.053	81.532	41.4	3.47	0.711	0.031	66.54
3000-1	15066	22	115	138.39	113	17.99	0.000	0.000	98.26	107	11.88	0.218	0.019	91.3	104	16.66	0.714	0.039	86.96	98	18.48	1.731	0.153	72.17
3000-2	15181	12	84	149.31	81	16.27	0.000	0.000	96.43	83	12.04	0.000	0.000	98.81	75	15.93	0.195	0.027	86.9	45	12.78	1.960	0.067	50
3000-3	14858	18	97	188.56	95	17.65	0.024	0.011	96.91	92	11.98	0.023	0.109	93.81	89	16.15	0.355	0.034	88.66	80	17.26	0.725	0.013	81.44
3000-4	14977	17	85	158.70	83	16.79	0.000	0.000	97.65	84	11.96	0.036	0.024	96.47	83	15.83	0.000	0.000	97.65	70	14.88	0.477	0.043	78.82
3000-5	14965	15	77	164.95	77	16.54	0.000	0.000	100	77	11.83	0.557	0.039	96.1	66	15.82	0.835	0.061	80.52	62	14.64	0.175	0.016	79.22
Average	15009.4	16.8	91.6	159.98	89.8	17.05	0.005	0.002	97.85	88.6	11.94	0.167	0.038	95.298	83.4	16.08	0.420	0.032	88.138	71	15.61	1.014	0.058	72.33
4000-1	20045	22	116	350.10	114	25.49	0.000	0.000	98.28	116	20.46	0.260	0.017	98.28	102	22.60	0.000	0.000	87.93	96	28.41	0.000	0.000	82.76
4000-2	19844	15	108	367.45	99	24.17	0.276	0.030	88.89	101	20.52	1.560	0.059	87.96	94	22.36	1.310	0.117	76.85	83	24.41	0.879	0.072	71.3
4000-3	20000	21	122	396.46	121	25.07	0.033	0.008	98.36	122	20.76	0.000	0.000	100	114	22.18	0.256	0.035	90.16	95	28.49	0.312	0.032	75.41
4000-4	19910	19	116	380.25	110	24.58	0.069	0.009	93.97	113	20.53	0.163	0.018	95.69	105	22.46	0.568	0.076	83.62	76	24.73	1.933	0.079	60.34
4000-5	19873	22	135	466.99	133	24.01	0.405	0.053	93.33	131	23.03	0.297	0.031	94.07	124	22.10	0.682	0.088	83.7	117	23.16	0.864	0.077	80
Average	19934.4	19.8	119.4	392.25	115.4	24.66	0.157	0.020	94.566	116.6	21.06	0.456	0.025	95.2	107.8	22.34	0.563	0.063	84.452	93.4	25.84	0.798	0.052	73.962
5000-1	25163	19	145	710.48	145	31.00	1.027	0.062	93.79	136	33.02	0.380	0.059	88.28	124	29.83	1.715	0.145	73.1	114	30.40	1.074	0.079	72.41
5000-2	25205	30	162	880.81	159	31.81	0.028	0.006	97.53	159	32.75	0.154	0.019	96.3	145	29.41	0.314	0.035	86.42	142	33.76	0.342	0.035	84.57
5000-3	25276	23	134	769.05	130	31.11	0.011	0.008	96.27	131	29.73	0.108	0.007	97.01	115	30.40	0.124	0.017	84.33	107	32.28	0.545	0.028	77.61
5000-4	25124	27	159	879.11	153	33.95	0.178	0.033	93.08	146	33.45	0.029	0.014	90.57	140	29.48	1.035	0.136	76.1	125	32.11	2.227	0.096	71.07
5000-5	24304	21	128	593.84	126	31.67	0.000	0.000	98.44	128	38.81	0.000	0.000	100	122	29.66	0.187	0.016	93.75	111	37.90	1.670	0.054	82.03
Average	25014.4	24	145.6	766.66	142.6	31.91	0.249	0.022	95.822	140	33.55	0.134	0.020	94.432	129.2	29.75	0.675	0.070	82.74	119.8	33.29	1.172	0.058	77.538

Table 3.2: Average number of non-dominated solution in large size networks (continued).

		E_S	**BRUM**		**HGA**					**CPH**					**BPR**					**GA**				
n	m	$\lvert\tilde{E}\rvert$	$\lvert\tilde{E}\rvert$	**CPU**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**
$d = 10$																								
1000-1	10280	8	39	10.62	39	6.52	0.000	0.000	100	39	4.79	0.000	0.000	100	31	6.93	0.000	0.000	79.49	30	6.34	0.000	0.000	76.92
1000-2	10330	7	37	14.29	37	6.44	0.000	0.000	100	37	4.69	0.000	0.000	100	29	7.32	1.147	0.138	67.57	29	5.13	0.358	0.103	70.27
1000-3	10419	12	38	9.43	38	6.80	0.000	0.000	100	38	4.81	0.000	0.000	100	35	6.82	0.000	0.000	92.11	34	5.50	0.000	0.000	89.47
1000-4	10053	11	30	11.00	30	6.38	0.000	0.000	100	30	4.70	0.500	0.033	96.67	29	7.08	0.000	0.000	96.67	28	4.94	0.927	0.036	90
1000-5	10350	12	44	8.73	43	6.69	0.000	0.000	100	43	4.72	0.000	0.000	97.73	42	7.08	0.075	0.024	93.18	43	5.32	0.000	0.000	97.73
Average	10286.4	10	37.6	10.81	37.4	6.57	0.000	0.000	100	37.4	4.74	0.100	0.007	98.88	33.2	7.05	0.244	0.032	85.804	32.8	5.44	0.257	0.028	84.878
3000-1	31722	17	71	286.96	69	22.64	0.000	0.000	97.18	68	18.87	0.000	0.000	95.77	66	21.62	0.242	0.015	91.55	68	19.42	0.282	0.029	92.96
3000-2	31507	18	69	400.44	66	22.88	0.000	0.000	95.65	67	18.96	0.000	0.000	97.1	64	22.17	0.377	0.031	89.86	57	19.09	0.055	0.018	81.16
3000-3	31210	16	96	320.80	94	22.28	0.000	0.000	97.92	96	18.89	0.000	0.000	100	90	21.27	0.069	0.022	91.67	90	18.97	0.069	0.022	91.67
3000-4	31273	14	54	135.00	54	22.13	0.000	0.000	100	54	18.95	0.176	0.019	98.15	50	21.69	0.000	0.000	92.59	47	17.96	0.060	0.021	85.19
3000-5	31262	20	84	316.82	83	22.76	0.000	0.000	98.81	83	19.24	0.555	0.012	97.62	83	21.85	0.000	0.000	98.81	81	19.87	0.027	0.012	95.24
Average	31394.8	17	74.8	292.00	73.2	22.54	0.000	0.000	97.912	73.6	18.98	0.146	0.006	97.728	70.6	21.72	0.138	0.014	92.896	68.6	19.06	0.099	0.020	89.244
4000-1	53639	13	77	629.95	73	35.47	1.197	0.055	89.61	73	30.23	0.000	0.000	94.81	71	33.19	0.803	0.127	80.52	58	32.53	0.422	0.086	68.83
4000-2	53623	16	86	630.23	84	34.22	0.000	0.000	97.67	81	30.20	0.000	0.000	94.19	78	33.10	0.628	0.013	89.53	77	31.57	0.000	0.000	89.53
4000-3	53527	17	59	902.12	58	35.65	0.000	0.000	98.31	58	30.27	0.000	0.000	98.31	58	33.98	0.017	0.017	96.61	57	34.93	0.092	0.035	93.22
4000-4	53946	18	98	599.52	98	34.24	0.000	0.000	100	97	33.24	0.170	0.010	97.96	93	33.06	0.050	0.022	92.86	89	34.04	0.091	0.023	88.78
4000-5	53634	22	85	664.36	83	34.28	0.000	0.000	97.65	84	33.50	0.000	0.000	98.82	82	32.89	0.343	0.049	91.76	77	33.58	0.209	0.026	0.8824
Average	53673.8	17.2	81	685.23	79.2	34.77	0.239	0.011	96.648	78.6	31.49	0.034	0.002	96.818	76.4	33.25	0.368	0.045	90.256	71.6	33.33	0.163	0.034	68.24848
5000-1	70382	16	114	1297.15	110	44.74	0.218	0.009	95.61	109	50.54	0.592	0.028	92.98	106	48.31	0.000	0.000	92.98	101	45.89	0.014	0.010	87.72
5000-2	70479	15	85	1246.53	85	44.56	0.000	0.000	100	84	51.32	0.000	0.000	98.82	83	44.76	0.349	0.048	92.94	79	44.63	0.229	0.025	90.59
5000-3	70054	18	93	1152.10	89	44.08	0.000	0.000	95.7	92	42.50	0.202	0.033	95.7	84	46.16	0.089	0.036	87.1	82	39.99	0.061	0.024	86.02
5000-4	70354	24	97	1776.08	92	44.88	0.199	0.022	92.78	93	43.93	0.000	0.000	95.88	94	44.02	0.021	0.011	95.88	88	44.29	0.046	0.011	89.69
5000-5	70431	22	111	1384.17	107	45.75	0.068	0.009	95.5	101	40.90	0.193	0.030	88.29	106	46.84	0.177	0.057	90.09	109	47.98	0.000	0.000	98.2
Average	70340	19	100	1371.21	96.6	44.80	0.097	0.008	95.918	95.8	45.84	0.197	0.018	94.334	94.6	46.02	0.127	0.030	91.798	91.8	44.56	0.070	0.014	90.444

Table 3.2: Average number of non-dominated solution in large size networks (continued).

		E_S	**BRUM**		**HGA**					**CPH**					**BPR**					**GA**				
n	m	$\lvert\tilde{E}\rvert$	$\lvert\tilde{E}\rvert$	**CPU**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**	$\lvert\tilde{E}\rvert$	**CPU**	**GD**	**ER**	**%Best**
$d = 20$																								
1000-1	20519	9	30	12.40	30	9.39	0.000	0.000	100	30	6.99	0.000	0.000	100	30	9.27	0.000	0.000	100	29	7.02	0.000	0.000	96.67
1000-2	20446	7	23	10.84	23	10.14	0.000	0.000	100	22	6.74	0.000	0.000	95.65	21	9.51	0.662	0.048	86.96	21	7.23	0.000	0.000	91.3
1000-3	20512	7	25	12.25	25	9.98	0.000	0.000	100	25	6.71	0.000	0.000	100	24	9.42	0.000	0.000	96	22	6.69	0.000	0.000	88
1000-4	20858	9	29	12.65	28	6.79	0.000	0.000	96.55	29	6.71	0.000	0.000	100	27	9.67	0.000	0.000	93.1	25	8.14	0.000	0.000	86.21
1000-5	20452	7	20	15.31	20	6.84	0.000	0.000	100	20	6.78	0.000	0.000	100	20	9.76	0.335	0.050	95	17	6.94	0.395	0.059	80
Average	20557.4	7.8	25.4	12.69	25.2	8.63	0.000	0.000	99.31	25.2	6.79	0.000	0.000	99.13	24.4	9.53	0.199	0.020	94.212	22.8	7.20	0.079	0.012	88.436
3000-1	59720	14	50	325.65	50	30.40	0.000	0.000	100	50	26.81	0.000	0.000	100	50	29.27	0.160	0.040	96	47	23.48	0.000	0.000	94
3000-2	59470	14	44	161.90	44	30.12	0.000	0.000	100	43	26.06	0.000	0.000	97.73	42	29.31	0.119	0.238	93.18	38	24.29	0.132	0.026	84.09
3000-3	59813	10	48	364.72	47	30.97	0.447	0.021	95.83	48	26.42	0.813	0.063	93.75	43	29.99	0.000	0.000	89.58	44	26.32	0.364	0.046	89.58
3000-4	59139	16	49	401.56	47	30.79	0.000	0.000	95.92	49	26.30	0.641	0.041	95.92	43	29.40	0.000	0.000	87.76	43	25.73	0.000	0.000	87.76
3000-5	59559	9	38	342.57	38	30.07	1.333	0.026	97.37	34	26.42	0.912	0.029	86.84	35	28.93	0.000	0.000	92.11	34	22.97	0.278	0.029	86.84
Average	59540.2	12.6	45.8	319.28	45.2	30.47	0.356	0.009	97.824	44.8	26.40	0.473	0.027	94.848	42.6	29.38	0.056	0.056	91.726	41.2	24.56	0.155	0.020	88.454
4000-1	80298	17	56	348.82	56	41.43	0.161	0.018	98.21	55	44.53	0.000	0.000	98.21	55	40.79	0.000	0.000	98.21	54	38.53	0.000	0.000	96.43
4000-2	80222	13	70	924.50	66	44.43	1.390	0.061	88.57	65	46.76	1.180	0.046	88.57	66	41.13	0.097	0.030	91.43	61	36.47	0.000	0.000	87.14
4000-3	79766	16	69	869.08	66	43.89	0.015	0.015	94.2	66	46.27	0.518	0.046	91.3	58	40.73	0.000	0.000	84.06	60	39.38	0.000	0.000	86.96
4000-4	80178	16	70	804.08	67	43.89	0.612	0.015	94.29	66	46.69	0.015	0.015	92.86	65	40.19	0.203	0.046	88.57	63	40.87	0.813	0.079	82.86
4000-5	80272	83	16	756.08	83	42.57	0.000	0.000	100	83	50.48	0.098	0.024	97.59	81	41.48	0.089	0.037	93.98	81	38.47	0.260	0.025	95.18
Average	80147.2	15.6	69.6	740.51	67.6	43.24	0.436	0.022	95.054	67	46.95	0.362	0.026	93.706	65	40.87	0.078	0.023	91.25	63.8	38.75	0.215	0.021	89.714
5000-1	102023	19	92	1826.81	87	53.32	1.024	0.046	90.22	86	52.97	1.051	0.047	89.13	85	61.58	0.106	0.035	89.13	81	50.29	0.149	0.037	84.78
5000-2	103297	16	59	1358.60	59	53.09	0.000	0.000	100	59	60.67	0.000	0.000	100	58	56.77	0.069	0.017	96.61	58	53.71	0.103	0.035	94.92
5000-3	103536	19	71	1307.62	70	53.45	0.101	0.014	97.18	69	56.91	0.013	0.015	95.77	67	57.04	1.535	0.105	84.51	65	48.66	0.314	0.031	88.73
5000-4	102485	17	82	1625.23	78	54.78	1.046	0.039	91.46	80	57.89	1.374	0.063	91.46	78	59.33	0.000	0.000	95.12	80	51.15	0.000	0.000	97.56
5000-5	102121	16	76	1593.45	76	53.68	0.201	0.013	98.68	76	56.69	0.000	0.000	100	74	61.20	0.000	0.000	97.37	74	55.63	0.027	0.014	96.05
Average	102692.4	17.4	76	1542.34	74	53.66	0.474	0.022	95.508	74	57.03	0.488	0.025	95.272	72.4	59.19	0.342	0.031	92.548	71.6	51.89	0.119	0.023	92.408

Concerning the percentage of the true found solutions, both heuristics (GA and BPR) found on average 80-90% of Pareto solutions. This represents a worse result than the one provided by the CPH heuristic, which generates results varying from 93% to 99% of the true Pareto solutions. While the heuristic combining the three heuristics HGA can generate solutions that are slightly better than the CPH heuristic, the generated results vary on average between 94.5% and 100% of the true Pareto solutions.

Figure 3.5, presents the percentage of true Pareto solutions generated by each one of the different heuristics in large networks with different densities. From Tables 1 and 2, considering the evaluation performance measure, we can conclude that the HGA is the best one from the proposed heuristics and can find very good results that can reach 100% of the true Pareto solutions and can cover the entire Pareto front. But we should not ignore the performance of the three heuristics separately and in particular heuristic CPH, which represents an excellent method that provides very good results in a reasonable time.

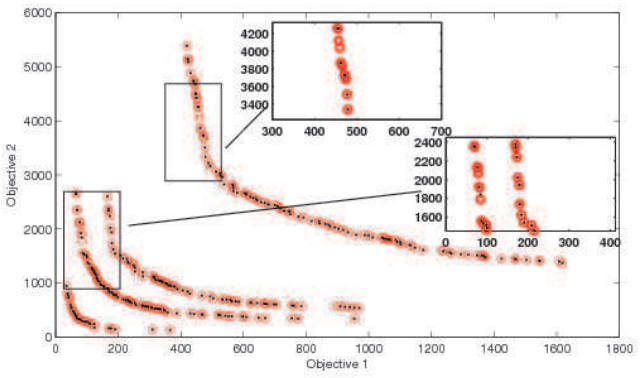

Figure 3.6: Pareto-optimal solution profiles of different large size networks (worst results).

Figure 3.6, presents a set of selected Pareto front found by the HGA heuristic. These sets represent the worst found results. These are compared to the true Pareto front generated by the labeling algorithm BRUM. In this figure, we can notice that the generated Pareto fronts are superimposed to the true Pareto fronts and that they cover almost the whole of solutions in these real Pareto fronts.

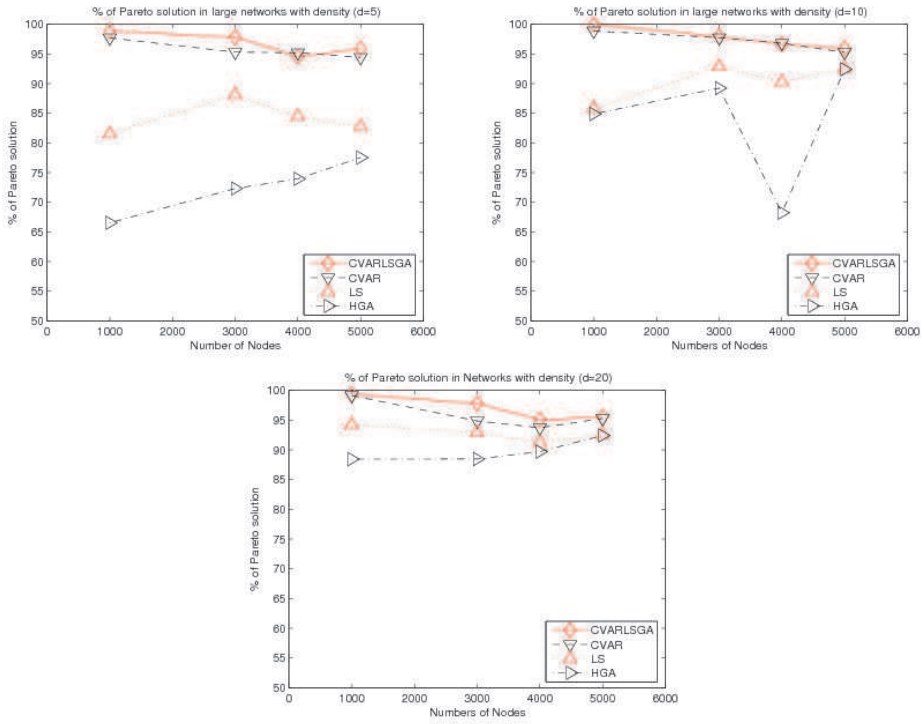

Figure 3.5: % of Pareto solution generated by different heuristics in large networks with different density.

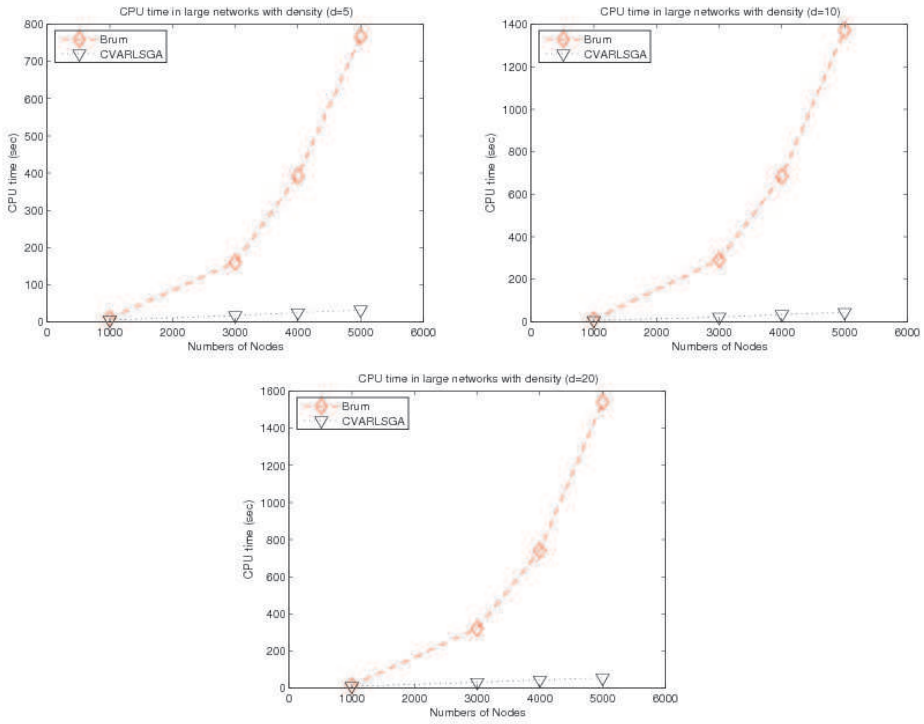

Figure 3.7: Computation time in large networks.

In Figure 3.7, to show the solution efficiency, we compare the average CPU time of

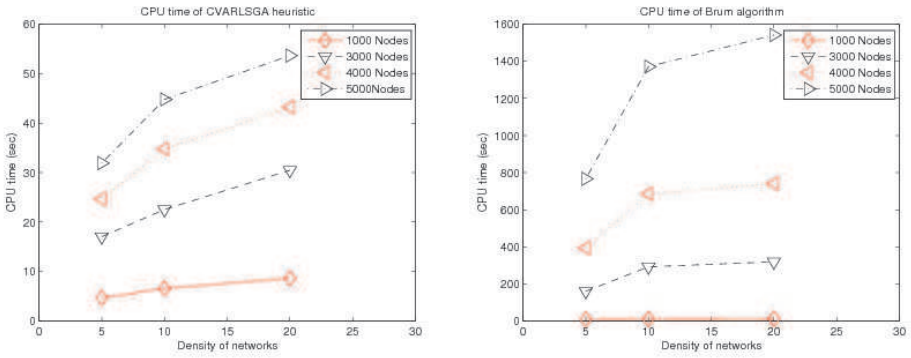

Figure 3.8: Computation time of (HGA and BRUM) over network density.

the labeling algorithm (BRUM) and the HGA heuristic over the largest size networks and different densities. Indeed, the exact algorithm can find all the Pareto optimal solutions, but we noticed that the time required to find these solutions explodes while increasing the size of the networks. In fact, the consumed time by the labeling algorithm passes from 9 seconds for the networks with 1000 nodes to 1540 seconds for networks with 5000 nodes. Sometimes, the proposed algorithm does not permit to enumerate all the Pareto optimal solutions. However, the percentage of true Pareto-optimal solutions generated approximately ranges between 87% and 100%, and the time consumed to obtain these solutions passes from 4.6 seconds, for the networks with 1000 nodes, to 53.66 seconds for the networks with 5000 nodes, which represents a rate of 5% of the time consumed by the labeling algorithm BRUM.

Figure 3.8 illustrates the CPU time variations over the density of the networks of the proposed algorithm and the labeling algorithm. From these two figures, one can notice a slight sensitivity of the HGA heuristic regarding the labeling algorithm (BRUM) for this variation in density.

3.7 Conclusion

A bi–objective shortest path problem is a natural extension of the mono–objective shortest path problem. To solve this NP-hard problem, we propose new two phase heuristics to approximate the set of the efficient solutions. In the first phase, we generate the supported efficient set by a standard dichotomic algorithm. In the second phase we use four meta-heuristics to generate an approximation of the non–supported efficient solutions. These

metaheuristics are the cost perturbation method, the path–reliking, the genetic algorithm and, finally, a hybrid approach combining all of them. We propose a hybrid approach for bi–objective optimization problems, which combines genetic algorithm and mathematical programming techniques. This method is based on the dominance cost variant of the multi-objective genetic algorithm hybridized with an exact method. The initial population is generated by solving a series of mono–objective optimization problems obtained by a suitable choice of a set of weights. The crossover operator solves a reduced mono-objective problem where the weights are chosen to identify an unexplored region. The proposed approaches are tested on instances of the bi–objective shortest path problem.

Chapter 4

Metaheuristics for the bi-objective assignment problem

4.1 Bi–objective assignment problem

The assignment problem arises in many real life situations including production scheduling (Dessouky and Kijowski, 1997), student schools (McKeown and Workman, 1976), aircraft routing (Soumis et al., 1980), snow removal and disposal (Campbell and Langevin, 1995), parking place assignment (Venkataramanan and Bornstein, 1991), etc. The classical Assignment Problem (AP) is a combinatorial optimization problem involving one-to-one matchings from two finite sets. In the canonical variation of the problem the two sets have the same cardinality. Typically, its objective, is to obtain the minimum cost or the maximum profit (Burkard et al., 2009). The dimension of the problem is assessed by the number of items to be matched. In the AP, tasks must be assigned to agents in such a way that the sum of the assignment costs is minimized. Mathematically, the AP is equivalent to the weighted bipartite matching problem from graph theory. The single objective AP can be formulated as a linear 0–1 Programming:

$$min \quad c(x) = \sum_{i,j \in N} c_{ij} x_{ij} \tag{4.1}$$

$$s.t. \quad \sum_{j \in N} x_{ij} = 1, \quad \forall i \in N, \tag{4.2}$$

$$\sum_{j \in N} x_{ji} = 1, \quad \forall i \in N, \tag{4.3}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \in N, \tag{4.4}$$

where x_{ij} is a binary variable that equals to 1 if task i is assigned to agent j, and 0 otherwise. In the objective function (4.1), c_{ij} is the cost resulting from assigning task i to agent j and $c(x)$ designates the total cost of the assignments. Constraints (4.2) state that every task is assigned to one and only one agent. Constraints (4.3) ensure that every agent is allocated to one and only one task. Variables x_{ij} are restricted to be binary, according to constraint (4.4). By considering only one objective, the Hungarian method of (Kuhn, 1955) provides an adequate solution of the problem (Papadimitriou and Steiglitz, 1998).

The assignment problem becomes multi–objective whenever the decision maker must take into account several criteria simultaneously, in order to provide more realistic solutions that optimize, for example, cost, time, distance, and quality. Therefore, the AP can be defined as an optimization problem with multiple potentially conflicting objectives. We consider the bi–objective framework of the assignment problem that can be formulated as follows:

$$min \quad c(x) = (c_1(x), c_2(x)) \tag{4.5}$$

$$s.t. \quad \sum_{j \in N} x_{ij} = 1, \quad \forall i \in N, \tag{4.6}$$

$$\sum_{j \in N} x_{ji} = 1, \quad \forall i \in N, \tag{4.7}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \in N. \tag{4.8}$$

where $c_k(x) = \sum_{i,j \in N} c_k(i, j) x_{ij}$. We denote by X the set of feasible solutions.
The bi–objective assignment problem is NP-complete, #P-complete, and intractable. The intractability means that the number of efficient solutions can be exponential. For a proof see (Serafini, 1987) for NP-completeness. (Malhotra et al., 1982) exploited the duality of the problem while assuming that the efficient solutions are connected by simplex pivots in order to generate the set of efficient supported solutions.

4.2 Genetic algorithm

In this section, we discuss the framework of our proposed genetic algorithm for solving the Bi-objective Assignment Problem (BAP). We propose a genetic algorithm for the BAP to generate an approximation of the set of non-supported efficient solutions. The proposed GA is similar to the one for the bi–objective shortest path problem. The population is

initialized as the set of supported efficient solutions generated by the first phase of the two phases method. The main steps of our GA consist of encoding solutions, defining an initial population and a crossover operator (see Algorithm 9). In our implementation the mutation is not applied.

Algorithm 9: Genetic algorithm

Input: E_S^* the set of supported efficient solutions for the BAP problem.

Output: $\tilde{E}$ approximation of efficient solution set for the BAP problem.

Set $\tilde{E} = \{(x, \lambda(x)) : x \in E_S^* \text{ and } \lambda(x) \text{ the associated weight }\}$;

while *stopping criterion is not meet* **do**

  Select randomly two supported solutions $(x', \lambda(x')), (x'', \lambda(x'')) \in \tilde{E}$;

  Set $\tilde{x} = \text{Crossover}((x', \lambda(x')), (x'', \lambda(x'')))$;

  $\tilde{E} = Dom(\tilde{x}, \tilde{E})$;

return $\tilde{E}$;

- **Encoding solutions:** An assignment is a bi–ojective mapping of the set N into itself, *i.e.* a permutation. An assignment can be modeled and visualized in different ways: every permutation φ of the set N corresponds in a unique way to a permutation matrix $x_\varphi = (x_{ij})$, with $x_{ij} = 1$ for $j = \varphi(i)$ and $x_{ij} = 0$ for $j \neq \varphi(i)$. The matrix x_φ can be viewed as an adjacency matrix of a bipartite graph $G_\varphi = (N \cup N, E)$, where an edge $(i, j) \in E$ if and only if $j = \varphi(i)$. In our implementation of the genetic algorithm, an assignment is represented as a permutation φ of n agents ($\{1, \ldots, n\}$). Table 4.1 illustrates the assignment of 7 tasks to 7 agents.

Tasks	1	2	3	4	5	6	7
Agents	5	2	7	3	6	4	1

Table 4.1: Encoding scheme

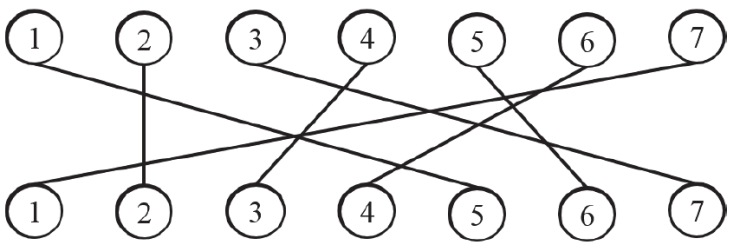

Figure 4.1: Bipartite graph representation.

In this coded solution, $\varphi(1) = 5$ signifies that task 1 is assigned to agent 5, $\varphi(2) = 2$ *i.e.* task 2 is assigned to agent 2 and so on. In the equivalent binary solution x, *i.e.* $x_{15} = x_{22} = x_{37} = x_{43} = x_{56} = x_{64} = x_{71} = 1$ and $x_{ij} = 0$ for the remaining cases. The associated matrix x is as follows:

$$x = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- **Initial population:** The initial population is defined as the set of supported efficient solutions E_S^* determined in the first phase of the two phase method. This set E_S^* is generated by the dichotomic approach described in Chapter 3. For each solution $x' \in E_S^*$ in the initial or the current population, we keep its associated weight vector $\lambda' = \lambda(x')$, *i.e.* the solution x' is an optimal solution of the following aggregated problem:

$$(AP(\lambda')) \begin{cases} \min \lambda'_1 c_1(x) + \lambda'_2 c_2(x) \\ \sum_{j \in N} x_{ij} = 1, \forall i \in N \\ \sum_{j \in N} x_{ji} = 1, \forall i \in N \\ x_{ij} \in \{0, 1\}, \forall i, j \in N \end{cases}$$

Example Consider the following numerical example for assigning 7 tasks to 7 agents, where the matrix costs associated to the first and the second objectives are c_1 and c_2, respectively:

$$c_1 = \begin{pmatrix} 42 & 68 & 35 & 1 & 70 & 25 & 79 \\ 59 & 63 & 65 & 6 & 46 & 82 & 28 \\ 62 & 92 & 96 & 43 & 28 & 37 & 92 \\ 5 & 3 & 54 & 93 & 83 & 22 & 17 \\ 19 & 96 & 48 & 27 & 72 & 39 & 70 \\ 13 & 68 & 100 & 36 & 95 & 4 & 12 \\ 23 & 34 & 74 & 65 & 42 & 12 & 54 \end{pmatrix}$$

$$c_2 = \begin{pmatrix} 69 & 48 & 45 & 63 & 58 & 38 & 60 \\ 24 & 42 & 30 & 79 & 17 & 36 & 91 \\ 43 & 89 & 7 & 41 & 43 & 65 & 49 \\ 47 & 6 & 91 & 30 & 71 & 51 & 7 \\ 2 & 94 & 49 & 30 & 24 & 85 & 55 \\ 57 & 41 & 67 & 77 & 32 & 9 & 45 \\ 40 & 27 & 24 & 38 & 39 & 19 & 83 \end{pmatrix}$$

The initial population defined by the optimal set of supported efficient solutions obtained by the above procedure is given in the Table 4.2.

λ_1	λ_2	c_1	c_2
0.995851	0.004149	115	239
0.537815	0.462185	140	208
0.511962	0.488038	170	175
0.477778	0.522222	198	148
0.356913	0.643087	217	132
0.003155	0.996845	315	128

Table 4.2: The initial population

Columns 1 and 2 present the weight vector (λ_1, λ_2) associated to each efficient supported solution. Columns 3 and 4 display the value of each objective function c_1 and c_2, respectively.

- **Crossover operator:** It is well known that the crossover (recombination) operator is the main operator in genetic algorithms. It produces new individuals, called offspring, from the selected parents (in our case, efficient non supported assignments). In our genetic algorithm, the crossover operator works as follows. Let x' and x'' be two parents from the current efficient set with their associated weights $\lambda(x')$ and $\lambda(x'')$ respectively. We define the set $I(x', x'') = \{(i, j) \in N \times N : x'_{ij} = x''_{ij} = 1\}$, which corresponds to the set of tasks i assigned to the agent j in both solutions x' and x''. Now, we define the reduced assignment problem $AP(x', x'')$ associated to the two parents x' and x'' as follows:

$$(AP(x', x'')) \begin{cases} \min \lambda_1 c_1(x) + \lambda_2 c_2(x) \\ \sum_{j \in N} x_{ij} = 1, \forall i \in N \\ \sum_{j \in N} x_{ji} = 1, \forall i \in N \\ x_{ij} = 1, \forall (i, j) \in I(x', x'') \\ x_{ij} \in \{0, 1\}, \forall i, j \in N \end{cases}$$

where $\lambda_1 = \alpha\lambda_1(x') + (1 - \alpha)\lambda_1(x'')$, $\lambda_2 = \beta\lambda_2(x') + (1 - \beta)\lambda_2(x'')$, and $\alpha, \beta \in [0, 1]$. In our experimentation, we set $\alpha = \beta = 0.5$. An optimal solution $\tilde{x}$ of the problem $AP(x', x'')$ is a candidate solution to be added to the current approximation $\tilde{E}$ of efficient solutions set.

The crossover operator implemented for the BAP problem is described in Algorithm 10. The crossover operates on two parents chosen randomly from the current population. Those parents correspond to two efficient solutions x' and x'' with their associated weights are λ' and λ'', respectively. We solve the reduced problem $AP(x', x'')$. Let $\tilde{x}$ be its optimal solution. If it is not dominated, we add it to the approximate set $\tilde{E}$ of efficient solutions.

Algorithm 10: Crossover Operator

Input: Two efficient solutions x' and x'' with their associated weights $\lambda(x')$ and $\lambda(x'')$ respectively.

Output: $\tilde{x}$ a possible efficient solution for the BAP problem.

Compute the set $I(x', x'') = \{(i, j) \in N \times N : x'_{ij} = x''_{ij} = 1\}$;

Solve the reduced problem $AP(x', x'')$ with $\tilde{x}$ an optimal solution;

return $\tilde{x}$;

Example Consider two parents x' and x'', selected at random from the current approximate set $\tilde{E}$ of efficient solutions, and $\tilde{x}$ a new offspring (Table 4.3).

i	1	2	3	4	5	6	7
x'	4	3	5	2	1	7	6
x''	4	5	3	7	1	6	2
$\tilde{x}$	4	*	*	*	1	*	*

Table 4.3: crossover operator

From Table 4.3 we have $I(x', x'') = \{(1, 4), (5, 1)\}$.

The obtained feasible offspring of the example after solving the reduced problem $AP(x', x'')$ is presented in Table 4.4.

i	1	2	3	4	5	6	7
$\tilde{x}$	4	5	3	2	1	7	6

Table 4.4: The complete permutation of the obtained offspring

The value of the first objective of this solution is equal to 189 whereas the second objective is equal to 159.

4.3 Bi–objective path relinking

In this section, we present the details of the proposed Bi–objective Path Relinking procedure (BPR) for BAP to generated an approximation of the non–supported efficient solution set $E^*_{\tilde{S}}$. The procedure is similar to the one proposed for the bi–objective shortest path problem. Based on the same idea of the crossover operator, a new efficient solution will be generated by solving a reduced problem in a dynamic way. However, in the crossover operator a new solution is generated based on two other solutions called parents.

The procedure starts from an initial solution selected randomly from the set of supported efficient solutions E^*_S and a weight vector chosen arbitrarily. The latter corresponds necessarily to another efficient supported solution as a target to the search direction. Next, we select at random some decision variables to be fixed and optimize the remaining ones according to the weight vector, by solving the weighted sum sub-problem. The process is repeated for several search directions to approximate the Pareto Front until reaching a given stopping criterion.

This BPR heuristic has as argument the set of supported solutions E^*_S generated in the first phase of the two phases method. At each iteration, two solutions are generated, as usually done by path relinking. One of them is called the *initial* solution, and the other is called the *guiding* solution. The initial solution x is selected randomly from the set of the support efficient solutions ($x \in E^*_S$) with its associated weight $\lambda(x)$, *i.e.* x is an optimal solution of the parametrized single objective problem $AP(\lambda(x))$. The guiding solution x' is also selected randomly from E^*_S with its associated weighted $\lambda(x')$, *i.e.* x' is an optimal solution of $AP(\lambda(x'))$.

Once the two solutions are selected, the crossover procedure is applied. The solution $\tilde{x}$ generated by the crossover operator is added to the efficient set $\tilde{E}$ if it is non dominated by one of the already efficient solutions generated, (*i.e.*, if yes a new non–supported efficient solution is generated).

Algorithm 11: Bi–objective path relinking

Input: E_S^* the set of supported efficient solutions for the BAP.

Output: $\tilde{E}$ approximation of efficient solution set for the BAP.

Set $\tilde{E} = \{(x, \lambda(x)) : x \in E_S^*$ and λ the associated weight $\}$;

while *stopping criterion is not meet* **do**

select randomly $(x, \lambda(x)) \in \tilde{E}$;

select randomly $(x', \lambda(x')) \in \tilde{E}$;

Set $\tilde{x} =$ Crossover$((x', \lambda(x')), (x'', \lambda(x'')))$;

$\tilde{E} = Dom(\tilde{x}, \tilde{E})$;

return $\tilde{E}$;

4.4 Hybrid genetic algorithm

In the literature, various hybridization methods have been integrated to genetic algorithms aiming at preventing them from being trapped into a local optimum. Hybrid approaches were extensively exploited in combinatorial optimization. Therefore, the proposed bi–objective path relinking presented in the above section is combined with the genetic algorithm after the crossover operator. Thus, the new offspring is considered as an initial solution. At each iteration, the choice of the weight vector is modified in an altering way between the weight vector of the first parent $\lambda(x')$ and the weight vector of the second parent $\lambda(x'')$. In other words, the weight sum sub-problem is solved by considering $\lambda(x')$ and $\lambda(x'')$ iteration-by-iteration. At each iteration of the local search procedure, we must check if the obtained solution is an efficient one or not. If there is a new efficient solution, then it will be introduced into the initial population.

Algorithm 12: Hybrid genetic algorithm

Input: E_S^* the set of supported efficient solutions for the BAP.

Output: $\tilde{E}$ approximation of efficient solution set for the BAP.

Set $\tilde{E} = \{(x, \lambda(x)) : x \in E_S^*$ and λ the associated weight $\}$;

while *stopping criterion is not meet* **do**

Select randomly two supported solutions $(x', \lambda(x')), (x'', \lambda(x'')) \in \tilde{E}$;

Set $\tilde{x} = \text{Crossover}((x', \lambda(x')), (x'', \lambda(x'')))$;

$E' = \text{BPR}(\tilde{x}, \lambda(\tilde{x}), \tilde{E})$;

$\tilde{E} = Dom(E', \tilde{E})$;

return $\tilde{E}$;

4.5 Computational results

The proposed algorithms were coded with language C++. All experiments for the bi-objective assignment problem were run in Windows 7 on a workstation with Intel Pentium, quad core i7, 3.00 GHz processor and 16 MB memory. The data sets used in our experiments are those of (Przybylski et al. (2008)). Moreover, we generated large scale instances for $n \in \{150, 200, 250, 300, 350, 400, 450, 500\}$. For each size, the range of the coefficients c_{ij} varies between 20 and 100 with a step of 20. In total, 180 instance problems were tested. The parameters of the proposed Genetic Algorithm (GA), Biobjective Path Relinking (BPR), Hybrid Genetic Algorithm (HGA) and Cost Perturbation Heuristic (CPH), described in Chapter 3, were set experimentally as follows: the stopping criterion for the GA is fixed as a maximum CPU time according to the instances type. For the instances with size $n \in \{20, 40, 60, 80, 100\}$, the maximum CPU time is equal to 100 seconds. For the instances with $n \in \{150, 200, 250, 300, 350\}$ this time is set to 900 seconds. Finally, for the remaining instances, the maximum CPU time is set to 3600 seconds. For the CPH, the stopping criterion is fixed to 5000 iterations. In order to evaluate the performance of the proposed algorithms, we use the following measures of performance: Error Ratio (ER), Generational Distance (GD), and the Coverage Metric (CM).

Tables 4.5- 4.8 display the experimental results of the tested instances. The first column indicates the name of the instances, which are identified as 2APn-lAu, being n the number of agents (vehicles) and $[l, u]$ the interval of possible values for the allocating costs (c_{ij}). The second column shows the cardinality of the optimal set of non-dominated solutions ($|E^*|$), and the following columns indicate, for each of the four methods, the cardinality of the non-dominated set, the GD metric, the ER metric and the CPU time. The values of $|E^*|$ are obtained with the exact two-phase method proposed in (Przybylski et al. (2008)) for the instances with $n \leq 100$. For the larger instances this algorithm was unable to converge to the optimal set after a CPU time of three hours. Therefore, these classes of instances, the set E^* is found by determining the set of non-dominated solutions among all the solutions of GA, BPR, HGA and CPH.

Tables 4.5- 4.8 present the computational results for the instances with, respectively, $n = 20$, $n \in \{40, 60, 80, 100\}$, $n \in \{150, 200, 250, 300, 350\}$ and $n \in \{400, 450, 500\}$.

Table 4.5: Computational Results for range 20

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP10-1A20	29	11	0.000	0.62	0.00	29	0.000	0.00	1.13	29	0.000	0.00	0.22	29	0.00	0.00	0.08
2AP20-1A20	49	37	0.001	0.37	0.18	49	0.000	0.00	4.05	49	0.000	0.00	2.24	48	0.00	0.02	0.46
2AP30-1A20	98	67	0.000	0.43	0.97	98	0.000	0.00	2.84	98	0.000	0.00	2.05	98	0.00	0.01	1.58
2AP40-1A20	106	96	0.000	0.26	4.27	106	0.000	0.00	7.22	106	0.000	0.00	6.39	105	0.00	0.02	3.80
2AP50-1A20	117	94	0.000	0.38	4.10	117	0.000	0.00	47.45	117	0.000	0.00	12.00	115	0.00	0.02	5.43
2AP60-1A20	181	166	0.000	0.18	61.66	173	0.000	0.08	86.03	181	0.000	0.00	29.28	179	0.00	0.02	11.00
2AP70-1A20	176	164	0.000	0.15	15.82	167	0.000	0.19	62.38	175	0.000	0.02	8.84	173	0.00	0.02	15.10
2AP80-1A20	190	152	0.000	0.29	32.23	169	0.000	0.23	86.41	190	0.000	0.00	30.38	189	0.00	0.03	18.65
2AP90-1A20	216	203	0.000	0.19	26.60	194	0.000	0.32	89.06	216	0.000	0.02	52.16	216	0.00	0.04	29.61
2AP100-1A20	230	222	0.000	0.10	26.24	210	0.000	0.27	98.19	230	0.000	0.01	28.13	230	0.00	0.05	52.28
Average			0.000	0.30	17.21		0.000	0.11	48.48		0.000	0.01	17.17		0.00	0.02	13.80
AP10-2A20	10	7	0.000	0.30	0.02	10	0.000	0.00	0.25	10	0.000	0.00	0.09	10	0.00	0	0.05
2AP20-2A20	40	21	0.000	0.50	0.16	40	0.000	0.00	0.47	40	0.000	0.00	0.49	40	0.00	0	0.49
2AP30-2A20	96	66	0.000	0.44	1.05	96	0.000	0.00	3.22	96	0.000	0.00	5.52	95	0.00	0.01	1.49
2AP40-2A20	109	88	0.000	0.28	1.22	109	0.000	0.00	3.34	109	0.000	0.00	2.64	109	0.00	0.01	3.06
2AP50-2A20	152	128	0.000	0.28	2.20	152	0.000	0.01	31.53	152	0.000	0.00	69.98	152	0.00	0.02	6.14
2AP60-2A20	152	143	0.000	0.16	66.93	146	0.000	0.10	28.41	152	0.000	0.00	11.83	150	0.00	0.03	11.45
2AP70-2A20	163	146	0.000	0.28	31.87	157	0.000	0.13	48.27	163	0.000	0.00	18.84	161	0.00	0.02	13.10
2AP80-2A20	173	157	0.000	0.22	81.19	158	0.000	0.23	91.42	173	0.000	0.01	19.41	173	0.00	0.03	22.39
2AP90-2A20	200	194	0.000	0.17	10.98	182	0.000	0.35	83.91	200	0.000	0.01	28.72	200	0.00	0.01	29.74
2AP100-2A20	225	214	0.000	0.14	16.60	209	0.000	0.29	90.06	225	0.000	0.00	24.20	224	0.00	0.04	40.38
Average			0.000	0.29	21.22		0.000	0.09	38.09		0.000	0.00	18.17		0.00	0.02	12.83
2AP10-3A20	20	9	0.000	0.55	0.02	20	0.000	0.00	0.84	20	0.000	0.00	0.13	20	0.00	0	0.07
2AP20-3A20	49	29	0.000	0.43	0.06	49	0.000	0.00	0.48	49	0.000	0.00	0.45	49	0.00	0.02	0.43

Table 4.5: Computational Results for range 20 (continued).

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP30-3A20	93	61	0.000	0.45	2.09	93	0.000	0.00	1.72	93	0.000	0.00	2.99	91	0.00	0.04	1.41
2AP40-3A20	133	107	0.000	0.35	1.65	133	0.000	0.00	24.02	133	0.000	0.00	2.36	133	0.00	0.02	3.22
2AP50-3A20	132	110	0.000	0.25	9.00	132	0.000	0.01	75.31	132	0.000	0.00	4.50	131	0.00	0.02	5.32
2AP60-3A20	188	167	0.000	0.22	16.22	183	0.000	0.15	95.28	188	0.000	0.03	46.19	188	0.00	0.02	9.97
2AP70-3A20	175	160	0.000	0.19	80.32	163	0.000	0.26	82.30	175	0.000	0.01	13.27	174	0.00	0.04	16.85
2AP80-3A20	191	181	0.000	0.16	40.24	170	0.000	0.26	93.73	191	0.000	0.00	43.06	189	0.00	0.04	22.20
2AP90-3A20	211	197	0.000	0.17	32.96	192	0.000	0.25	61.41	211	0.000	0.03	20.97	210	0.00	0.06	29.18
2AP100-3A20	247	229	0.000	0.18	35.51	230	0.000	0.25	66.45	245	0.000	0.03	41.20	247	0.00	0.05	45.26
Average			0.000	0.30	21.81		0.000	0.11	50.15		0.000	0.00	17.51		0.00	0.03	13.39
2AP10-4A20	15	7	0.000	0.53	0.01	15	0.000	0.00	0.17	15	0.000	0.00	0.16	15	0.00	0	0.08
2AP20-4A20	48	31	0.001	0.38	0.07	48	0.000	0.00	0.47	48	0.000	0.00	1.09	48	0.00	0	0.46
2AP30-4A20	81	48	0.000	0.53	1.33	81	0.000	0.00	3.03	81	0.000	0.00	2.45	80	0.00	0.01	1.34
2AP40-4A20	143	116	0.000	0.29	1.69	143	0.000	0.00	13.41	143	0.000	0.00	4.08	142	0.00	0.01	3.05
2AP50-4A20	151	121	0.000	0.36	3.29	147	0.000	0.03	51.27	149	0.000	0.01	5.24	146	0.00	0.07	6.38
2AP60-4A20	158	141	0.000	0.27	25.77	155	0.000	0.11	96.66	158	0.000	0.01	8.52	158	0.00	0.05	10.01
2AP70-4A20	177	162	0.000	0.22	7.39	171	0.000	0.18	96.06	176	0.000	0.01	18.67	177	0.00	0.01	15.30
2AP80-4A20	195	184	0.000	0.24	12.59	184	0.000	0.23	76.31	195	0.000	0.03	13.44	195	0.00	0.06	20.78
2AP90-4A20	190	174	0.000	0.18	30.21	173	0.000	0.27	59.94	190	0.000	0.01	18.89	188	0.00	0.05	28.88
2AP100-4A20	227	213	0.000	0.16	71.24	198	0.000	0.29	76.05	224	0.000	0.02	44.53	226	0.00	0.07	48.96
Average			0.000	0.33	15.36		0.000	0.09	47.34		0.000	0.01	11.71		0.00	0.03	13.52
2AP10-5A20	14	9	0.000	0.36	0.00	14	0.000	0.00	0.14	14	0.000	0.00	0.14	14	0.00	0	0.07
2AP20-5A20	39	16	0.000	0.59	0.12	39	0.000	0.00	0.44	39	0.000	0.00	0.55	38	0.00	0.03	0.44
2AP30-5A20	68	35	0.000	0.51	0.13	68	0.000	0.00	2.44	68	0.000	0.00	2.72	66	0.00	0.04	1.29
2AP40-5A20	133	114	0.000	0.28	2.26	133	0.000	0.00	21.36	133	0.000	0.00	4.63	133	0.00	0.02	3.68
2AP50-5A20	131	118	0.000	0.18	13.01	131	0.000	0.01	46.03	131	0.000	0.00	13.55	126	0.00	0.05	5.55

Table 4.5: Computational Results for range 20 (continued).

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP60-5A20	164	140	0.000	0.32	15.11	161	0.000	0.09	89.56	164	0.000	0.01	10.53	164	0.00	0.03	10.37
2AP70-5A20	189	169	0.000	0.17	24.64	176	0.000	0.22	99.47	189	0.000	0.01	48.52	186	0.00	0.06	16.08
2AP80-5A20	183	172	0.000	0.15	42.55	174	0.000	0.23	70.03	183	0.000	0.03	14.67	183	0.00	0.04	22.66
2AP90-5A20	228	215	0.000	0.14	21.16	212	0.000	0.35	91.44	228	0.000	0.01	17.94	226	0.00	0.04	28.94
2AP100-5A20	213	207	0.000	0.10	78.70	200	0.000	0.30	96.78	212	0.000	0.02	40.83	212	0.00	0.04	40.57
Average			0.000	0.29	19.77		0.000	0.11	51.77		0.000	0.01	15.41		0.00	0.03	12.97
2AP10-6A20	10	10	0.000	0.00	0.02	10	0.000	0.00	0.11	10	0.000	0.00	0.14	10	0.00	0	0.09
2AP20-6A20	51	22	0.000	0.57	0.08	51	0.000	0.00	0.44	51	0.000	0.00	0.47	51	0.00	0	0.48
2AP30-6A20	92	54	0.000	0.48	0.17	92	0.000	0.00	1.30	92	0.000	0.00	11.16	90	0.00	0.02	1.59
2AP40-6A20	117	70	0.000	0.48	2.55	117	0.000	0.00	8.39	117	0.000	0.00	3.20	116	0.00	0.01	3.18
2AP50-6A20	145	132	0.000	0.24	4.34	144	0.000	0.01	29.20	144	0.000	0.01	3.72	145	0.00	0.01	5.69
2AP60-6A20	170	152	0.000	0.28	19.50	162	0.000	0.14	90.17	170	0.000	0.01	11.30	170	0.00	0.01	11.19
2AP70-6A20	198	185	0.000	0.17	10.69	184	0.000	0.25	84.38	198	0.000	0.02	19.63	198	0.00	0.03	15.81
2AP80-6A20	193	186	0.000	0.16	13.18	186	0.000	0.34	99.27	193	0.000	0.01	13.17	193	0.00	0.05	22.79
2AP90-6A20	207	194	0.000	0.13	54.99	180	0.000	0.32	81.02	207	0.000	0.00	27.14	207	0.00	0.01	32.04
2AP100-6A20	218	206	0.000	0.09	26.88	204	0.000	0.31	95.80	218	0.000	0.00	24.30	218	0.00	0.02	45.37
Average			0.000	0.28	13.24		0.000	0.11	49.01		0.000	0.01	11.42		0.00	0.02	13.82
2AP10-7A20	19	6	0.000	0.68	0.00	19	0.000	0.00	0.19	19	0.000	0.00	0.13	19	0.00	0	0.08
2AP20-7A20	58	34	0.000	0.45	0.10	58	0.000	0.00	1.97	58	0.000	0.00	0.92	58	0.00	0	0.50
2AP30-7A20	95	64	0.000	0.39	0.43	95	0.000	0.00	11.02	95	0.000	0.00	5.77	95	0.00	0	1.64
2AP40-7A20	108	80	0.000	0.34	31.04	108	0.000	0.00	5.44	108	0.000	0.00	2.31	108	0.00	0.01	2.69
2AP50-7A20	124	86	0.000	0.43	1.71	124	0.000	0.00	32.09	124	0.000	0.00	27.67	119	0.00	0.09	5.83
2AP60-7A20	159	152	0.000	0.16	8.06	155	0.000	0.15	59.98	159	0.000	0.01	39.28	158	0.00	0.06	8.91
2AP70-7A20	181	171	0.000	0.14	27.02	173	0.000	0.20	81.59	181	0.000	0.02	39.11	180	0.00	0.04	16.82
2AP80-7A20	199	194	0.000	0.15	63.89	186	0.000	0.29	78.28	198	0.000	0.02	17.14	198	0.00	0.05	21.75

Table 4.5: Computational Results for range 20 (continued).

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP90-7A20	169	161	0.000	0.19	63.17	152	0.000	0.37	85.09	169	0.000	0.02	52.36	168	0.00	0.02	25.70
2AP100-7A20	226	214	0.000	0.12	30.93	214	0.000	0.22	75.03	223	0.000	0.02	24.88	226	0.00	0.03	40.12
Average			0.000	0.31	22.63		0.000	0.11	43.07		0.000	0.01	20.96		0.00	0.03	12.40
2AP10-8A20	21	7	0.000	0.67	0.01	21	0.000	0.00	0.16	21	0.000	0.00	0.11	21	0.00	0	0.08
2AP20-8A20	51	32	0.001	0.43	0.08	51	0.000	0.00	0.20	51	0.000	0.00	0.55	50	0.00	0.02	0.54
2AP30-8A20	92	56	0.000	0.49	0.45	92	0.000	0.00	2.77	92	0.000	0.00	9.34	92	0.00	0	1.58
2AP40-8A20	124	103	0.000	0.25	1.44	124	0.000	0.00	31.88	124	0.000	0.00	4.72	123	0.00	0.02	3.22
2AP50-8A20	156	122	0.000	0.38	5.07	153	0.000	0.03	25.84	156	0.000	0.00	37.67	155	0.00	0.02	6.09
2AP60-8A20	162	142	0.000	0.26	15.34	161	0.000	0.07	69.42	162	0.000	0.02	8.61	162	0.00	0.01	10.51
2AP70-8A20	184	178	0.000	0.13	14.94	176	0.000	0.20	96.08	183	0.000	0.02	15.72	184	0.00	0.03	14.85
2AP80-8A20	208	206	0.000	0.11	10.27	187	0.000	0.33	88.88	208	0.000	0.01	14.91	208	0.00	0.04	22.59
2AP90-8A20	220	208	0.000	0.10	21.13	205	0.000	0.25	45.89	220	0.000	0.02	42.53	219	0.00	0.06	35.11
2AP100-8A20	216	207	0.000	0.12	18.81	194	0.000	0.34	97.56	216	0.000	0.02	40.81	216	0.00	0.05	43.14
Average			0.000	0.31	8.75		0.000	0.09	45.87		0.000	0.01	17.50		0.00	0.03	13.77
2AP10-9A20	15	8	0.000	0.47	0.00	15	0.000	0.00	1.75	15	0.000	0.00	0.16	15	0.00	0	0.10
2AP20-9A20	63	36	0.000	0.44	0.06	63	0.000	0.00	1.27	63	0.000	0.00	0.50	63	0.00	0	0.45
2AP30-9A20	83	51	0.000	0.47	0.56	83	0.000	0.00	1.34	83	0.000	0.00	2.67	83	0.00	0	1.32
2AP40-9A20	108	82	0.000	0.33	3.18	107	0.000	0.01	14.02	108	0.000	0.00	8.09	108	0.00	0.02	3.05
2AP50-9A20	141	115	0.000	0.32	2.21	140	0.000	0.01	77.64	141	0.000	0.00	6.19	140	0.00	0.03	5.99
2AP60-9A20	172	162	0.000	0.10	10.92	169	0.000	0.07	98.91	172	0.000	0.00	13.63	172	0.00	0.04	10.04
2AP70-9A20	184	175	0.000	0.23	4.52	171	0.000	0.24	99.28	184	0.000	0.01	98.11	183	0.00	0.03	15.67
2AP80-9A20	182	172	0.000	0.15	60.72	167	0.000	0.27	79.14	182	0.000	0.01	11.73	182	0.00	0.05	20.81
2AP90-9A20	203	187	0.000	0.23	56.49	183	0.000	0.38	95.39	203	0.000	0.01	16.95	203	0.00	0.04	29.81
2AP100-9A20	213	199	0.000	0.16	56.58	198	0.000	0.26	93.25	213	0.000	0.00	23.88	213	0.00	0.03	43.46

Table 4.5: Computational Results for range 20 (continued).

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
Average			0.000	0.29	19.52		0.000	0.10	56.20		0.000	0.01	18.19		0.00	0.02	13.07
2AP10-10A20	11	8	0.007	0.36	0.03	11	0.000	0.00	0.19	11	0.000	0.00	0.11	11	0.00	0	0.08
2AP20-10A20	68	44	0.000	0.47	0.13	68	0.000	0.00	0.64	68	0.000	0.00	0.77	68	0.00	0	0.49
2AP30-10A20	107	64	0.000	0.53	2.80	107	0.000	0.00	8.86	107	0.000	0.00	37.64	107	0.00	0.02	1.35
2AP40-10A20	121	82	0.000	0.40	0.57	121	0.000	0.00	6.52	121	0.000	0.00	6.20	120	0.00	0.01	2.85
2AP50-10A20	165	120	0.000	0.42	7.17	162	0.000	0.02	95.48	164	0.000	0.01	4.53	164	0.00	0.02	5.81
2AP60-10A20	180	143	0.000	0.28	35.68	171	0.000	0.14	92.64	178	0.000	0.02	41.00	175	0.00	0.06	9.56
2AP70-10A20	198	181	0.000	0.24	38.59	187	0.000	0.23	64.78	198	0.000	0.02	37.95	198	0.00	0.04	15.92
2AP80-10A20	194	183	0.000	0.19	8.93	175	0.000	0.30	96.53	193	0.000	0.02	73.91	192	0.00	0.06	25.87
2AP90-10A20	182	169	0.000	0.16	90.93	161	0.000	0.38	91.08	182	0.000	0.01	87.38	181	0.00	0.04	27.42
2AP100-10A20	211	200	0.000	0.15	71.33	192	0.000	0.34	99.36	211	0.000	0.02	21.66	211	0.00	0.07	39.48
Average			0.001	0.33	25.62		0.000	0.11	55.61		0.000	0.01	31.11		0.00	0.03	12.88

Table 4.6: Computational Results for range 40, 60, 80 and 100

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP10-1A40	21	10	0	0.52	0.02	21	0	0.00	0.52	21	0	0.00	0.28	20	0.00	0.05	0.09
2AP20-1A40	66	23	0	0.65	0.07	66	0	0.00	1.58	66	0	0.00	1.50	59	0.00	0.11	0.55
2AP30-1A40	109	76	0	0.43	0.43	109	0	0.00	6.94	109	0	0.00	4.61	92	0.00	0.17	1.44
2AP40-1A40	186	96	0	0.60	58.19	186	0	0.00	67.86	186	0	0.00	45.34	141	0.00	0.25	3.32
2AP50-1A40	216	178	0	0.28	13.43	216	0	0.02	89.94	216	0	0.00	96.78	191	0.00	0.13	6.69
2AP60-1A40	253	189	0	0.42	11.69	223	0	0.30	98.91	248	0	0.05	87.53	208	0.00	0.23	9.86
2AP70-1A40	331	270	0	0.41	80.52	281	0	0.42	94.09	312	0	0.14	99.00	262	0.00	0.24	15.50
2AP80-1A40	355	280	0	0.39	80.99	291	0	0.41	95.88	337	0	0.09	96.97	305	0.00	0.20	21.86
2AP90-1A40	432	367	0	0.34	77.40	342	0	0.53	99.99	407	0	0.11	82.38	387	0.00	0.14	29.78
2AP100-1A40	429	380	0	0.30	94.14	333	0	0.51	94.52	415	0	0.07	89.11	401	0.00	0.13	41.79
Average			0	0.42	41.69		0	0.16	65.02		0	0.03	60.35		0.00	0.16	13.09
2AP10-1A60	17	10	0.001	0.47	0.02	17	0	0.00	94.52	17	0	0.00	0.80	15	0.00	0.12	0.02
2AP20-1A60	66	19	0	0.71	0.04	66	0	0.00	0.28	66	0	0.00	1.94	35	0.00	0.48	0.27
2AP30-1A60	139	78	0	0.50	13.11	139	0	0.00	0.75	139	0	0.00	9.78	98	0.00	0.31	0.84
2AP40-1A60	259	161	0	0.56	60.18	258	0	0.00	8.55	259	0	0.00	48.45	159	0.00	0.40	1.65
2AP50-1A60	304	187	0	0.50	17.59	299	0	0.02	42.16	295	0	0.04	92.03	187	0.00	0.42	3.36
2AP60-1A60	374	275	0	0.42	47.33	320	0	0.31	85.73	358	0	0.07	89.22	240	0.00	0.43	5.34
2AP70-1A60	460	335	0	0.42	46.60	370	0	0.43	91.33	437	0	0.12	98.31	262	0.00	0.46	8.00
2AP80-1A60	498	369	0	0.47	63.83	384	0	0.55	97.50	450	0	0.20	98.23	278	0.00	0.52	11.99
2AP90-1A60	571	462	0	0.37	93.94	370	0	0.69	98.42	491	0	0.26	99.19	360	0.00	0.44	15.87
2AP100-1A60	585	476	0	0.35	85.14	381	0	0.64	96.98	497	0	0.25	97.14	305	0.00	0.51	17.96
Average			0	0.48	42.78		0	0.20	61.62		0	0.05	63.51		0.00	0.41	6.53
2AP10-1A80	25	9	0	0.64	0.02	25	0	0.00	99.13	25	0	0.00	99.19	16	0.00	0.36	0.06
2AP20-1A80	94	58	0	0.45	0.73	94	0	0.00	0.44	94	0	0.00	0.52	59	0.00	0.37	0.30

Table 4.6: Computational Results for range 40, 60, 80 and 1000 (continued).

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP30-1A80	158	57	0	0.66	0.38	158	0	0.00	1.17	158	0	0.00	1.73	74	0.00	0.53	0.68
2AP40-1A80	218	124	0	0.52	6.40	218	0	0.00	11.50	218	0	0.00	8.41	113	0.00	0.50	1.66
2AP50-1A80	375	228	0	0.54	55.71	360	0	0.07	42.86	366	0	0.05	93.56	183	0.00	0.53	3.26
2AP60-1A80	431	254	0	0.58	41.73	347	0	0.36	92.98	392	0	0.17	92.72	157	0.00	0.65	4.52
2AP70-1A80	477	337	0	0.44	98.41	344	0	0.50	98.30	417	0	0.21	86.61	176	0.00	0.66	7.06
2AP80-1A80	677	482	0	0.51	90.81	451	0	0.64	97.72	561	0	0.32	97.30	278	0.00	0.62	13.83
2AP90-1A80	691	550	0	0.39	76.09	484	0	0.64	94.22	601	0	0.25	97.53	334	0.00	0.56	19.19
2AP100-1A80	845	637	0	0.49	93.85	521	0	0.65	99.77	656	0	0.37	99.59	319	0.00	0.67	24.19
Average			0	0.52	46.41		0	0.24	63.81		0	0.10	67.72		0.00	0.54	7.48
2AP10-1A100	13	7	0	0.46	0.00	13	0	0.00	99.97	13	0	0.00	98.69	11	0.00	0.15	0.05
2AP20-1A100	82	38	0	0.57	0.16	82	0	0.00	0.08	82	0	0.00	0.23	44	0.00	0.46	0.28
2AP30-1A100	169	56	0	0.67	1.07	169	0	0.00	5.72	169	0	0.00	1.27	58	0.00	0.64	0.64
2AP40-1A100	243	120	0	0.60	13.27	243	0	0.00	71.81	243	0	0.00	33.80	96	0.00	0.60	1.49
2AP50-1A100	301	143	0	0.65	94.98	294	0	0.03	55.59	295	0	0.04	45.59	72	0.00	0.76	2.89
2AP60-1A100	470	307	0	0.50	94.95	380	0	0.35	97.50	431	0	0.14	68.08	173	0.00	0.64	4.76
2AP70-1A100	573	387	0	0.47	65.95	437	0	0.43	99.81	509	0	0.21	97.14	201	0.00	0.65	8.80
2AP80-1A100	671	413	0	0.58	87.14	460	0	0.58	98.30	512	0	0.38	96.05	221	0.00	0.68	11.59
2AP90-1A100	722	497	0	0.52	98.39	427	0	0.64	98.36	537	0	0.40	95.45	221	0.00	0.71	16.71
2AP100-1A100	947	656	0	0.49	87.33	545	0	0.69	99.41	723	0	0.40	98.00	263	0.00	0.73	23.02
Average			0	0.55	54.32		0	0.23	72.65		0	0.12	63.43		0.00	0.61	7.02

Table 4.7: Computational Results for $n \in \{150, 200, 250, 300, 350\}$

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP150-1A20	234	234	0	0.04	153.13	223	0	0.31	879.30	234	0	0.01	90.22	234	0	0.01	90.36
2AP150-1A40	577	538	0	0.22	683.64	468	0	0.56	861.97	551	0	0.10	898.58	445	0	0.31	82.38
2AP150-1A60	856	776	0	0.30	835.08	613	0	0.70	852.95	779	0	0.23	860.61	512	0	0.48	85.86
2AP150-1A80	1298	1107	0	0.34	892.88	870	0	0.69	896.72	1124	0	0.26	860.38	616	0	0.58	76.84
2AP150-1A100	1273	1014	0	0.45	805.71	801	0	0.75	894.27	1099	0	0.34	873.36	402	0	0.70	78.49
Average			0	0.27	674.09		0	0.60	877.04		0	0.19	716.63		0	0.43	82.79
2AP200-1A20	271	264	0	0.07	112.59	243	0	0.36	847.75	271	0	0.00	61.90	271	0	0.03	189.16
2AP200-1A40	657	614	0	0.19	876.33	554	0	0.59	884.16	640	0	0.07	742.63	526	0	0.30	202.81
2AP200-1A60	1040	920	0	0.36	893.37	759	0	0.68	876.75	959	0	0.20	809.56	669	0	0.46	207.10
2AP200-1A80	1363	1166	0	0.42	893.76	800	0	0.79	899.77	1201	0	0.32	879.56	612	0	0.62	201.29
2AP200-1A100	1633	1336	0	0.47	892.60	955	0	0.77	896.34	1262	0	0.46	890.36	520	0	0.72	218.04
Average			0	0.30	733.73		0	0.64	880.95		0	0.21	676.80		0	0.43	203.68
2AP250-1A20	265	255	0	0.10	185.70	254	0	0.43	590.61	265	0	0.00	247.34	264	0	0.06	346.60
2AP250-1A40	776	728	0	0.22	862.57	688	0	0.60	846.02	753	0	0.06	867.35	617	0	0.30	403.06
2AP250-1A60	1080	1002	0	0.28	893.05	797	0	0.69	899.39	1024	0	0.13	876.04	698	0	0.44	415.49
2AP250-1A80	1402	1264	0	0.34	891.72	1056	0	0.70	897.58	1279	0	0.20	881.96	693	0	0.56	463.08
2AP250-1A100	1672	1518	0	0.30	896.92	1122	0	0.73	891.89	1529	0	0.22	897.45	696	0	0.63	411.33
Average			0	0.25	745.99		0	0.63	825.10		0	0.12	754.03		0	0.40	407.91
2AP300-1A20	299	289	0	0.11	494.56	274	0	0.44	890.00	299	0	0.00	694.81	299	0	0.02	554.85
2AP300-1A40	805	719	0	0.32	886.08	628	0	0.66	845.92	706	0	0.20	945.42	710	0	0.19	751.83
2AP300-1A60	1173	1018	0	0.41	893.99	856	0	0.70	898.10	847	0	0.35	934.90	851	0	0.35	744.57
2AP300-1A80	1455	1253	0	0.41	898.70	1058	0	0.71	895.53	899	0	0.45	910.42	848	0	0.47	726.46
2AP300-1A100	1564	1292	0	0.46	897.80	1251	0	0.63	892.66	832	0	0.53	1001.81	849	0	0.52	809.18

Table 4.7: Computational Results for $n \in \{150, 200, 250, 300, 350\}$ (continued).

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
Average			0	0.34	814.23		0	0.63	884.44		0	0.31	897.47		0	0.31	717.38
2AP350-1A20	290	281	0	0.05	400.77	256	0	0.56	724.05	284	0	0.05	1262.45	286	0	0.03	1093.64
2AP350-1A40	845	743	0	0.37	898.10	678	0	0.66	896.67	745	0	0.21	1354.22	744	0	0.19	1245.35
2AP350-1A60	1337	996	0	0.54	899.52	1116	0	0.63	898.82	978	0	0.36	1488.82	982	0	0.33	1223.79
2AP350-1A80	1455	1130	0	0.53	898.06	1164	0	0.66	898.22	898	0	0.45	1434.13	903	0	0.44	1246.18
2AP350-1A100	1556	1113	0	0.55	899.84	1247	0	0.65	898.33	847	0	0.51	1703.95	842	0	0.51	1551.15
Average			0	0.41	799.26		0	0.63	863.22		0	0.32	1448.71		0	0.30	1272.02

Table 4.7 presents the results for $n \in \{150, 200, 250, 300, 350\}$. It can be observed that when the size of the problems increases, the performance of the genetic algorithm is better than the performance of the cost perturbation heuristic.

Table 4.8: Computational Results for $n = 400, 450$ and 500

Instances	E^*	**GA**				**BPR**				**HGA**				**CPH**			
	$\|E^*\|$	$\|GA\|$	GD(GA)	ER(GA)	CPU	$\|BPR\|$	GD(BPR)	ER(BPR)	CPU	$\|HGA\|$	GD(HGA)	ER(HGA)	CPU	$\|CPH\|$	GD(CPH)	ER(CP)	CPU
2AP400-1A20	268	264	0	0.05	2261.89	255	0	0.38	2955.52	268	0	0.00	705.28	268	0	0.01	1706.680
2AP400-1A40	889	845	0	0.20	3490.77	777	0	0.66	3589.65	869	0	0.05	3578.66	741	0	0.23	1987.270
2AP400-1A60	1425	1319	0	0.30	3568.69	1153	0	0.66	3544.94	1352	0	0.13	3567.02	984	0	0.39	1694.310
2AP400-1A80	1861	1679	0	0.34	3578.88	1416	0	0.71	3557.58	1644	0	0.25	3571.63	965	0	0.53	2176.660
2AP400-1A100	2119	1861	0	0.33	3593.11	1575	0	0.75	3579.52	1867	0	0.26	3576.41	893	0	0.63	1943.120
Average			0	0.24	3298.67		0	0.63	3445.44		0	0.14	2999.80		0	0.36	1901.61
2AP450-1A20	293	293	0	0.01	2188.91	272	0	0.43	3578.22	293	0	0.00	1253.90	288	0	0.02	2197.720
2AP450-1A40	895	845	0	0.23	2863.62	751	0	0.62	2925.80	868	0	0.08	2891.61	736	0	0.23	2547.080
2AP450-1A60	1438	1280	0	0.37	3563.08	1129	0	0.70	3469.67	1296	0	0.21	3596.64	939	0	0.41	3022.470
2AP450-1A80	1847	1644	0	0.40	3598.77	1395	0	0.72	3565.76	1645	0	0.26	3587.95	943	0	0.55	2636.690
2AP450-1A100	2219	1873	0	0.45	3102.61	1669	0	0.72	3396.98	1929	0	0.29	2865.98	1020	0	0.59	3137.690
Average			0	0.29	3584.33		0	0.64	3584.44		0	0.17	3588.29		0	0.36	2708.33
2AP500-1A20	288	277	0	0.08	1108.87	270	0	0.43	3366.03	288	0	0.00	1481.22	283	0	0.02	3110.070
2AP500-1A40	967	908	0	0.25	3559.83	816	0	0.70	3284.53	920	0	0.10	3581.62	808	0	0.22	3960.460
2AP500-1A60	1492	1292	0	0.44	3591.99	1209	0	0.73	3587.11	1306	0	0.24	3596.34	995	0	0.40	4419.000
2AP500-1A80	1890	1695	0	0.36	3597.90	1388	0	0.75	3586.79	1527	0	0.38	3567.77	993	0	0.53	4046.210
2AP500-1A100	2264	1917	0	0.45	3596.38	1806	0	0.69	3559.04	1788	0	0.38	3595.57	1050	0	0.58	3989.330
Average			0	0.31	3091.00		0	0.66	3476.70		0	0.22	3164.50		0	0.35	3905.01

In terms of the ER measure, the HGA can be considered as the best one for the class of instances with $n = 20$ (Table 4.5), with an average value of 0.007. For the majority of the instances, HGA finds all the solutions of the optimal front. In average, the ER measures for GA, BPR and CPH are equal to 0.29, 0.12 and 0.27, respectively. In terms of the GD measure, we can see that, although these three methods do not find the whole optimal front, the obtained front is very close to it since the values of GD tend to zero (Figures 4.2 and 4.3).

For the class of instances with $n \in \{40, 60, 80, 100\}$, the HGA can still be considered as the best one in terms of the ER measure, with an average value of 0.11. The ER measures for GA, BPR and CPH are, in average, equal to 0.50, 0.26 and 0.43, respectively. Note that the ER increases when the coefficients c_{ij} are in $[1, 60]$, $[1, 80]$ and $[1, 100]$. This occurs specially for CPH. For the larger instances (Tables 4.7 and 4.8), the HGA continues to be the best one in terms of the ER measure, with average values of 0.23 and 0.18, respectively. For these instances, the performance of GA increases (the average values of ER are respectively 0.31 and 0.28) whereas the performance of BPR declines (the average values of ER are respectively 0.63 and 0.64). For the CPH, the average values of ER, for these instances, are respectively 0.43 and 0.36.

In terms of CPU time, the obtained results show that, in average, GA, BPR and HGA are similar, whereas CPH consumes less computational time.

Finally, again in terms of the GD measure, although the methods do not find the whole optimal front, the obtained front is very close to it since the values of GD tend to zero (Figures 4.4–4.6).

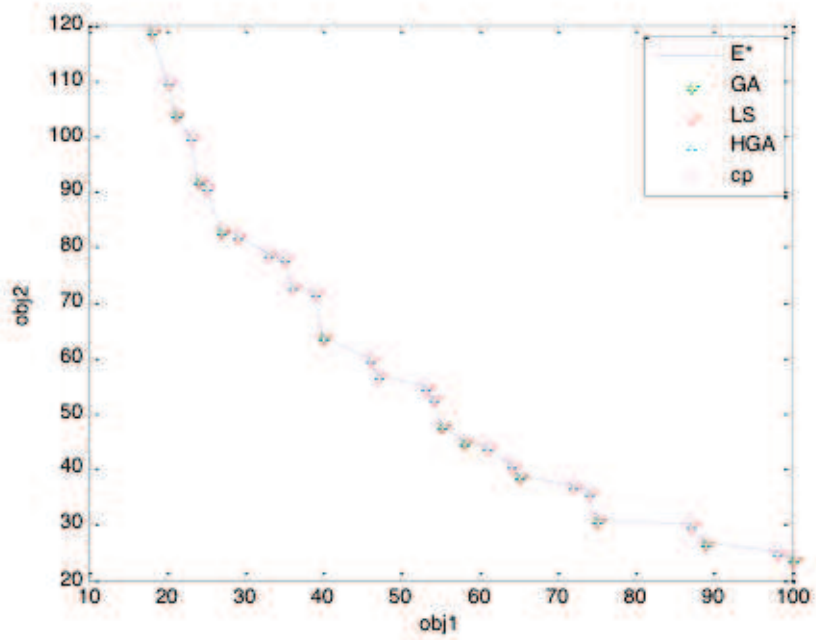

Figure 4.2: Pareto fronts of 2AP10-1A20

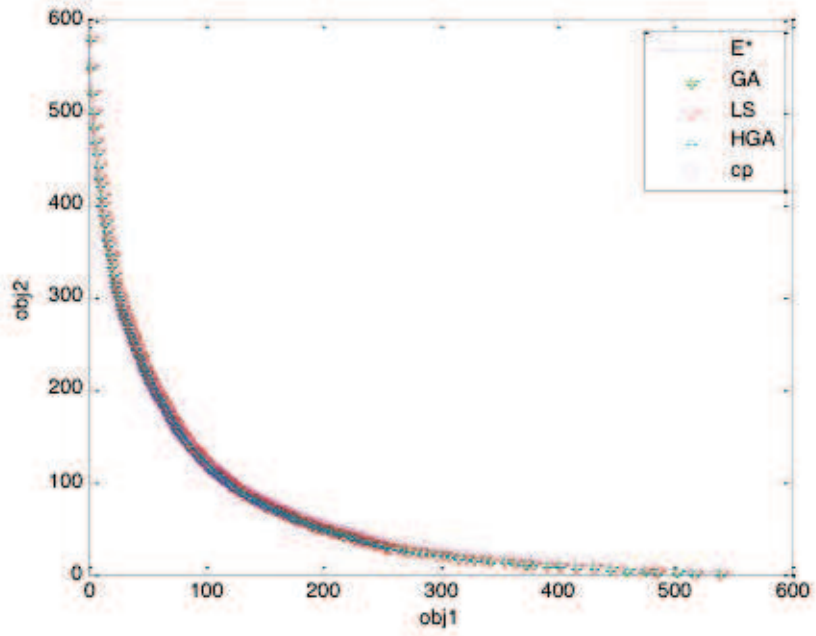

Figure 4.3: Pareto fronts of 2AP100-10A20

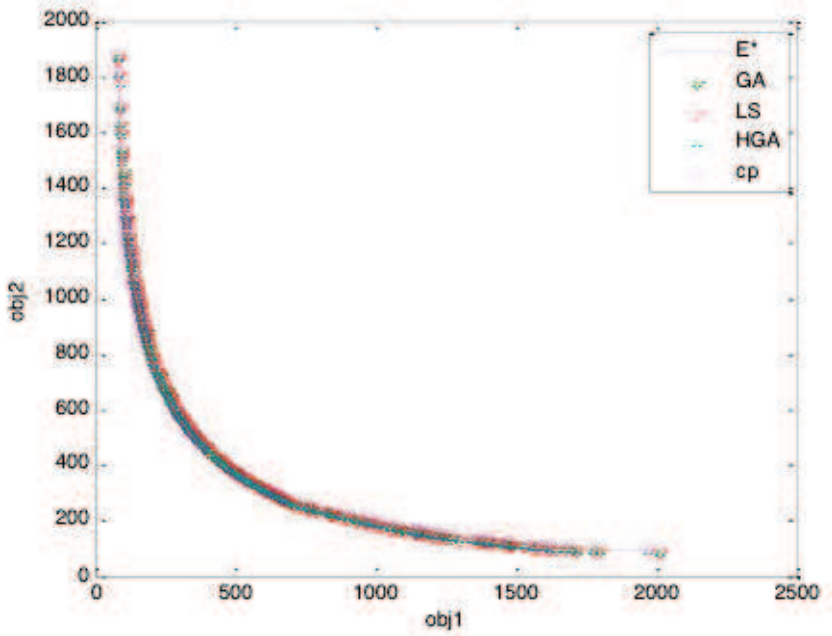

Figure 4.4: Pareto fronts of 2AP50-1A80

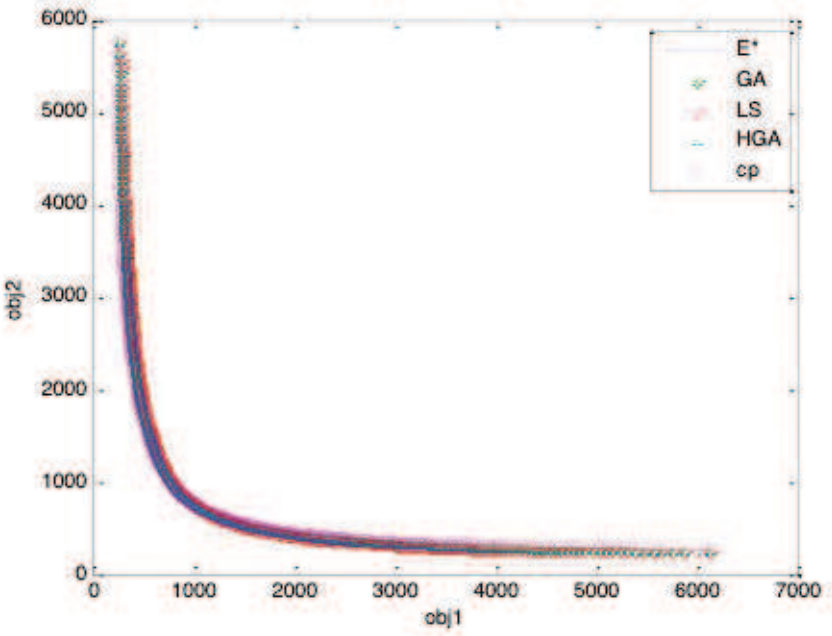

Figure 4.5: Pareto fronts of 2AP250-1A60

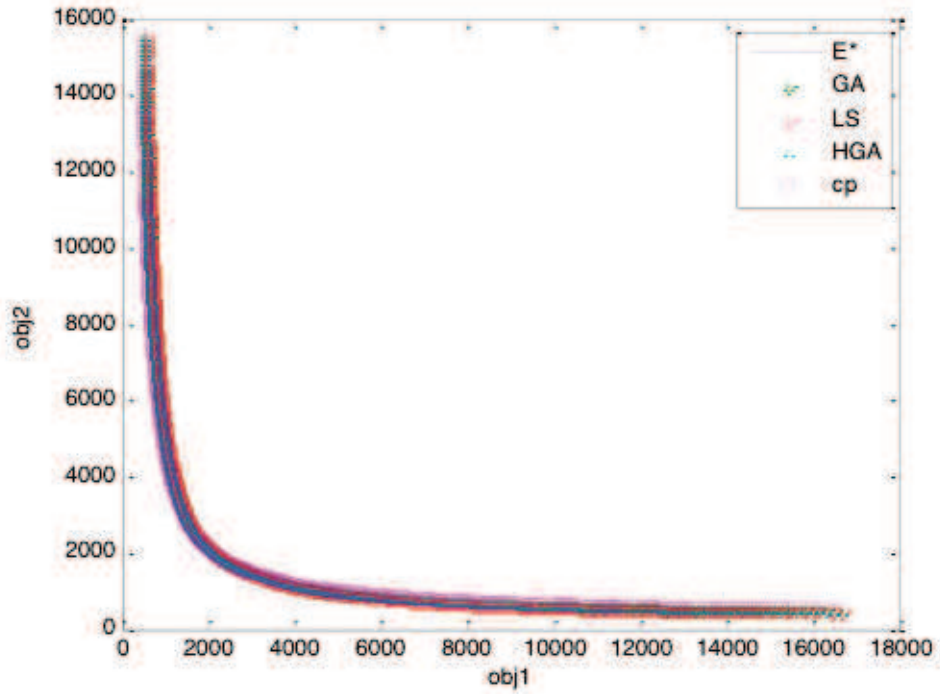

Figure 4.6: Pareto fronts of 2AP500-1A100

4.6 Conclusion

In this chapter, we proposed metaheuristics for the bi-objective assignment problem. More precisely, we proposed a two phase method where the first phase generates the supported efficient set and the second phase approximates the unsupported efficient set. For this four metaheuristics are proposed namely a multi-objective genetic algorithm, a bi-objective path relinking and a hybrid approach. Moreover, a local search procedure was introduced for balancing the diversification and the intensification in the search process. The experimental results show the efficiency of our proposed algorithms for large instances in comparison with an exact method from the literature.

Chapter 5

Dynamic assignment for parking slot problem

5.1 Introduction

A dynamic assignment problem consists of solving a sequence of assignment problems over time. At each time period, decisions must be made in what concerns which resources and tasks to assign to each other. Assignments that are made at earlier time periods affect the assignments that are made during later time periods, and information about the future is often uncertain. Some examples of dynamic assignment problems include dispatching truck drivers to deliver loads, pairing locomotives with trains and assigning company employees to jobs (Spivey and Powell (2003)).

(Geng and Cassandras (2011)) propose a "smart parking" system for an urban environment based on a dynamic resource allocation approach. The goal of the system is to provide an optimal assignment of users (drivers) to the parking slots, respecting the overall parking capacity. The quality of an assignment is measured using a function that combines proximity to destination and parking cost. At each decision point, the system considers two queues of users: WAIT queue (consists of users who wait to be assigned to resources) and RESERVE queue (consists of users who have already been assigned and have reserved a resource in some earlier decision point). An optimal allocation of all users in both WAIT queue and RESERVE queue at each decision point is determined by solving a mixed integer linear programming problem. Simulation results show that

the "smart parking" approach provides near-optimal resource utilization and significant improvement compared to classical guidance-based systems.

In (Mejri et al. (2013)), the authors present an approach for assigning parking slots using a semi-centralized structure that uses Parking Coordinators (PC). They propose a new approach to guide drivers to parkings. It aims to ensure driver satisfaction and improve the occupancy balance between parking areas. They propose two variants: in the first variant, each PC affects a parking independently of the others; in the second variant, the PCs interact with their near neighbors, to assign a location with the constraint of balancing the load of each car park. The idea behind this approach is to use PCs to gather vehicles (drivers) queries during a certain time window and assign a parking slot according to these vehicles preferences. Both variants of the solution were simulated. In the first one, the PCs take parking decisions regardless of their environment and neighbors. In the second variant, these computers (*i.e.* PCs) exchange informations with their close neighbors. The simulation results showed that the second variant outperformed the first one, specially when the number of conflicting demands between PCs increased. The results for the second variant are very close to the case with a decision-centralized system, while being more scalable. These preliminary results showed that cooperation between the parking coordinators is strongly recommended.

5.2 Mixed integer programming formulation

A driver represents a user and a parking slot represents a resource. A driver $i \in I = \{1, 2, \ldots, n\}$, aiming to visit a given destination, starts looking for a parking slot by launching a request at a non–deterministic moment. These moments are not known in advance and are discovered during the assignment process. A parking $j \in J = \{1, 2, ...m\}$ has a certain number of available slots that change during the time. Both drivers requests and available parking slots appear in a non–deterministic way and can change their state at any moment. We assume that each resource, each driver and each destination has a known location associated to it in a two–dimensional Euclidean space.

The density of drivers requests is not uniform, it varies from a time period to another during the same day. For example, the requests density for a parking slot near to a

hospital rises in the visiting hours and declines at other times. In the same way, the requests density of a parking slot around public administration buildings depends on the working hours. Therefore, to take into account this feature, each day of the time horizon is subdivided into a fixed number T of equally spaced time periods. Periods that represent peak hours, normal hours, ... off-peak hours.

The objective of assigning vehicles to parking slots is to provide a global satisfaction of all customers. For instance, when the decision is performed request per request independently using a FIFO (First In First Out) rule, it may not be satisfactory because it can have a negative impact on the future situations. The following example, presented in Figure 5.1, illustrates the impact of the starting choice on the future decision when using a FIFO rule. We assume that we have one available parking slot in parking p_1 and another in parking p_2. Moreover, we suppose that vehicle v_1 launches a request for a parking slot near to destination d_1 and vehicle v_2 looks for a place near to destination d_2. In addition, it is assumed that vehicle v_1 arrives before vehicle v_2 into the system. If we aim to minimize the distance between the parking and the destination, the FIFO rule will suggest to assign vehicle v_1 to parking p_1 and vehicle v_2 to parking p_2. As the vehicle v_1 request is handled first, regardless of the vehicle v_2 request. The result is "good" for deriver r_1 but "bad" for driver r_2 and bad for the global system. This is due to the fact that the future information concerning the second driver is excluded. However, if vehicle v_1 is assigned to parking p_2 and vehicle v_2 to parking p_1, driver r_1 preserves his satisfaction degree as the distance from parking p_1 to destination d_1 is similar to the distance from parking p_2 to destination d_1, but the degree of satisfaction of driver r_2 is increased.

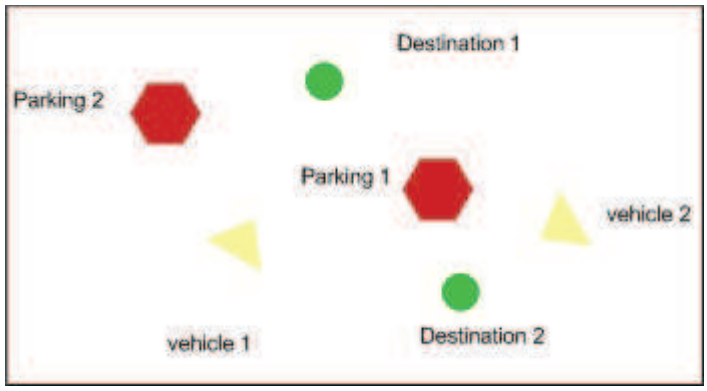

Figure 5.1: Illustrative example

To escape from this drawback, two alternatives may be presented: the first one consists

in collecting a subset of the requests in a given moment and solving the associated problem. The second one consists in establishing a forecast process to model the future information. In this section, we will discuss the first alternative and in a later section we will propose a forecast approach. The operation of collecting requests needs a collecting time period. We define a partition of time to gather requests and to determine the number of available slots in each parking created by the outgoing vehicles. Each time period, $t = 1, 2, ..., T$, of a day is partitioned into fixed K equally spaced sub-periods. We denote by k the index of the k^{th} sub-period. The process of assignment is performed in a discrete given moment, at the end of each sub-period called "decision point". Each decision point has also an index denoted by k.

At each decision point k, we denote by N_k the set of new requests gathered during the sub-period k. Due to the limited capacity of the parking, some requests may not be satisfied at each decision point k. We denote by R_k the set of yet unassigned vehicles at the decision point k. Therefore, the set of vehicles that have to be assigned at the next decision point $k+1$ is $E_{k+1} = N_{k+1} \cup R_k$, the sum of new requests N_{k+1} and the non-handled requests up-to-now R_k.

To formally define the dynamic assignment problem, we need the following:

- **Parameters:**
  - $dest_i^k$: the location of the destination of vehicle i from the set E_k in the two-dimensional Euclidean space.
  - loc_i^k: the location of vehicle i from the set E_k in the two-dimensional Euclidean space.
  - $d_{i,j}^k$: the distance between the location of the vehicle i from the set E_k and the parking j at the decision point k.
  - $d_{[i],j}^k$: the walking distance, *i.e.* distance between the destination $dest_i^k$ of the vehicle i from the set E_k and the parking j at the decision point k.
  - V: the velocity of vehicles. We assume that all vehicles have the same velocity.
  - V': the walking velocity of a driver needed to reach the destination after parking the vehicle. We assume that this velocity is the same for all drivers.

- w_i^k: the waiting time of a vehicle i between the launch of the request and the decision point k.
- C_j^k: the capacity of the parking j at the decision point k. This parameter designates the number of available slots in the parking j at the k^{th} decision point.

- **Variables:** We introduce the binary decision variables defined as follows:

$$x_{i,j}^k = \begin{cases} 1 & \text{if the vehicle } i \text{ is assigned to parking } j \text{ at the decision point } k, \\ 0 & \text{otherwise.} \end{cases}$$

- **Criteria:** As the occupancy time of a parking slot begins from the assignment of the requests, which is not necessary the arrival time to the parking, the objective of the decision maker is to minimize the total distance between all vehicles and their assigned parking slots. This maximizes the occupancy and satisfies the parking manager. However, to guarantee a good quality of service to the customers, it is also necessary to minimize two elements:
  - the distance traveled between the assigned parking slot and the customer final destination. This could be done by choosing the closest possible available parking slot;
  - the waiting time that separates the launch of the request and the decision to assign the vehicle to a parking slot. This could be done by introducing a queuing factor to provide a priority for the vehicles having longer waiting times.

  These objectives are aggregated into a single weighted objective as follows:

$$Minf^k(x) = \sum_{i \in I} \sum_{j \in J} x_{i,j}^k (\lambda_1 \frac{d_{i,j}^k}{V} + \lambda_2 \frac{d_{[i],j}^k}{V'} - \lambda_3 w_i^k) \tag{5.1}$$

  where λ_1, λ_2 and λ_3 are non-negative values denoting the weights of each criterion. The non-positive coefficient associated to the third criterion is used as a priority factor.

- **Constraints:** The main constraints in the considered problem at each decision point deal with the limited capacity of each parking and the satisfaction of the drivers requests. However, as the total number of available parking slots at each decision point is not necessarily equal to the total requests to be assigned, the constraints will be different. In this subsection, the constraints of the problem are expressed according to the availability of the parking slots and the number of launched requests at the considered decision point. Three possible cases will be considered.

  - **Case 1:** The number of available slots is larger than the number of requests at the k^{th} decision point:

$$\sum_{i \in I} x_{i,j}^k \leq C_j^k \qquad \forall j \in J \tag{5.2}$$

$$\sum_{j \in J} x_{i,j}^k = 1 \qquad \forall i \in I \tag{5.3}$$

    Constraints (5.2) guarantee that, at each decision point k, the number of assigned cars in the parking j cannot exceed its capacity. Constraints (5.3) ensure that each driver i is assigned to one and only one parking slot.

  - **Case 2:** The number of available slots is less than the number of requests at the k^{th} decision point:

$$\sum_{i \in I} x_{i,j}^k = C_j^k \qquad \forall j \in J \tag{5.4}$$

$$\sum_{j \in J} x_{i,j}^k \leq 1 \qquad \forall i \in I \tag{5.5}$$

    In this case, we formulate the constraints such that all the slots in the parking j will be occupied and some drivers may not be assigned to any parking slot.

  - **Case 3:** The number of available slots is equal to the number of requests at the k^{th} decision point:

$$\sum_{i \in I} x_{i,j}^k = C_j^k \qquad \forall j \in J \tag{5.6}$$

$$\sum_{j \in J} x_{i,j}^k = 1 \qquad \forall i \in I \tag{5.7}$$

In this last case, the offer of the parking slots is equal to the demand and thus each parking slot should be occupied and each driver should find a parking slot.

For the ease of use, we propose to formulate our problem as a simple assignment problem, where the capacity of each agent is equal to one. This is done by disaggregating the parking slots. Instead of considering the slots through the capacity of each parking we consider the available slots individually. Therefore, the problem becomes an assignment of drivers to parking slots instead of an assignment of drivers to parking. Let J_j^k be the set of available slots in a parking j in the set J at the decision point k. The binary decision variables will be defined, from now on, as follows:

$$x_{i,h}^k = \begin{cases} 1 & \text{if the vehicle } i \text{ is assigned to parking slot } h \text{ at the decision point } k, \\ 0 & \text{otherwise.} \end{cases}$$

The new proposed formulation of our problem can be presented as follows:

$$Minf^k(x) = \sum_{i=1}^{n} \sum_{j \in J} \sum_{h \in J_j^k} x_{i,h}^k (\lambda_1 \frac{d_{i,h}^k}{V} + \lambda_2 \frac{d_{[i],h}^k}{V'} - \lambda_3 w_i^k), \tag{5.8}$$

subject to the following constraints:

– **Case 1:** The number of available slots is larger than the number of requests at the k^{th} decision point:

$$\sum_{i \in I} x_{i,h}^k \leq 1 \qquad \forall j \in J, \forall h \in J_j^k \tag{5.9}$$

$$\sum_{j \in J} \sum_{h \in J_j^k} x_{i,h}^k = 1 \qquad \forall i \in I \tag{5.10}$$

– **Case 2:** The number of available slots is less than the number of requests at the k^{th} decision point:

$$\sum_{i \in I} x_{i,h}^{k} = 1 \qquad \forall j \in J, \forall h \in J_j^k \tag{5.11}$$

$$\sum_{j \in J} \sum_{h \in J_j^k} x_{i,h}^{k} \leq 1 \qquad \forall i \in I \tag{5.12}$$

– **Case 3:** The number of available slots is equal to the number of requests at the k^{th} decision point:

$$\sum_{i \in I} x_{i,h}^{k} = 1 \qquad \forall j \in J, \forall h \in J_j^k \tag{5.13}$$

$$\sum_{j \in J} \sum_{h \in J_j^k} x_{i,h}^{k} = 1 \qquad \forall i \in I \tag{5.14}$$

The considered problem is dynamic because, during a given day, the demand for parking slots is not uniform and the number of requests launched by the drivers changes from one period $t \in T$ of the day to another. Therefore, assignments that are made at earlier periods affect which assignments can be made during later periods. Moreover, the information about the future periods is uncertain. Our goal is to efficiently manage the requests in the time to deal with this uncertainty. To do so, we propose to establish a forecasting process based on a learning effect. We introduce a penalty term in the objective function that determines for each parking and each period of the day if the current assignment has or not an impact on the future ones. In other words, if the future demand around a parking j is going be higher, the current penalty should be big, in order to leave more available slots in this parking for the future period. Otherwise, the penalty will be small, in order to make the slots of this parking more attractive for the current assignment. The value of these penalties are calibrated through a learning process.

Let p_j^t denote the penalty term associated to parking j at the period of time t $(t = 1, 2, ..., T)$. It should be noted that for each slot in the parking j, the penalty is equal to the parking penalty p_j^t. In addition, between two consecutive time periods t_1 and t_2, such that $t_1 < t_2$ and for a decision points k such that $t_1 \leq k < t_2$, we set $p_j^k = p_j^{t_1}$.

The new objective function can be written as follows:

$$Minf^k(x) = \sum_{i=1}^{n} \sum_{j \in J} \sum_{h \in J_j^k} x_{i,h}^k (\lambda_1 \frac{d_{i,h}^k}{V} + \lambda_2 \frac{d_{[i],h}^k}{V'} - \lambda_3 w_i^k + p_j^k) \tag{5.15}$$

5.3 Estimation of distribution algorithms

According to the framework of the classical Genetic Algorithm (GA), the process of recombination occurs during meiosis, resulting from crossovers between parental chromosomes. Through this process, the offspring inherit different combinations of genes from their parents regardless the link between them. Moreover, the tuning of the parameters (population size, probabilities of crossover and mutation, etc) and the prediction of the movements of the populations are difficult tasks to perform in a GA. These drawbacks motivated the development of the Estimation of Distribution Algorithm (EDA).

The EDA was introduced to estimate the correlation between genes and uses this information during the search process. In an EDA there are neither crossover nor mutation operators. It was first introduced by (Mühlenbein and Paass, 1996), and it is a stochastic optimization technique that explores the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions. Starting with a population of individuals (candidate solutions), generally randomly generated, this algorithm selects good individuals with respect to their fitness. Then, a new distribution of probability is estimated from the selected candidates. Next, new offspring are generated from the estimated distribution. The process is repeated until the termination criterion is met.

This model-based approach to optimization has allowed EDAs to solve successfully many large and complex problems such as the quadratic assignment problem (Zhang et al., 2006), the 0–1 knapsack problem (Li et al., 2004), the n–queen problem (Paul and Iba, 2002), the traveling salesman problem (Robles et al., 2002).

EDA typically works with a population of individuals generated at random. At each generation, a subset of the most promising solutions is selected by a selection operator with respect to a fitness function. The main phase of the algorithm is to estimate, from informations contained in the selected individuals, a probability distribution. Then, a

new individual is generated from the constructed probability model. The new solution may be incorporated into the previous population if it satisfies a replacement criterion. Otherwise, it will be rejected. Finally, the process is reiterated until a stopping criterion is satisfied. The main steps of a basic EDA are summarize in Algorithm 13.

Algorithm 13: Basic EDA

Generate an initial population of P individuals

repeat

Select a set of Q parents from the current population P with a selection method;

Build a probabilistic model for the set of selected parents;

Create a new offspring to P according to the estimated probability distribution;

Replace some individuals in the current population P with new individuals;

until *a stopping criterion is met*;

The choice of the probabilistic model is not a trivial task. Many works have focused on the way to establish the distribution of probability that allows capturing the features of the promising solutions. Three classes of EDA may be presented according to the chosen probabilistic model and the degree of dependency between the variables of the problem. The first class called *univariate model*, assumes no dependencies between variables of candidate solutions, wich is to say that all variables are independent. The second one, called *bivariate model*, assumes only pairwise dependencies between these variables and the last class, called *multivariate model*, assumes multiple dependencies between variables. In the next sections, we describe the main principles of those methods.

5.3.1 Univariate models

In this section, we discuss the simplest approaches of EDA where it is assumed that the problem variables are independent. Under this assumption, the probability distribution of any individual variable should not depend on the values of any other variables. The common characteristic between all the models belonging to this category is to consider that, for each generation g, the n-dimensional joint probability distribution decomposes the probability of a candidate solution into a product of n univariate and independent probability distributions such that $p^g(x) = \prod_{i=1}^{n} p^g(x_i)$.

- **Population based incremental learning:** It was introduced by (Baluja, 1994) and considers binary variables. Here, at each generation g, a candidate solution $x \in \{0,1\}^n$ of a current population P is encoded by a vector of probability $p^g(x) = (p^g(x_1), p^g(x_2), \ldots, p^g(x_n))$, where $p^g(x_i)$ denotes the probability of the component x_i to take value "1". Initially, all positions are equiprobable and all probabilities are set to 0.5. Then, based on Q selected individuals, the probability vector is updated according to the following expression

$$p^{g+1}(x) = (1-\alpha)p^g(x) + \frac{\alpha}{Q}\sum_{k=1}^{Q} x_k^g \tag{5.16}$$

where x_k^g is the k^{th} best individual in the population at the g^{th} generation and α is the learning rate. It is easy to observe that in equation (5.16) each component is evaluated independently of others and thus no interaction is considered. In (Sebag and Ducoulombier, 1998), the authors proposed an adaptation of the population based incremental learning to continuous domain. Each element of the mean vector is estimated, at generation $g+1$, by the following equation

$$\hat{\mu}_k^{g+1} = (1-\alpha)\hat{\mu}_k^g + \alpha(x_{1^*}^g + x_{2^*}^g - x_w^g) \tag{5.17}$$

where solutions $x_{1^*}^g$ and $x_{2^*}^g$ are the two best solutions and solution $x^g *_w$ is the worst solution discovered in the current generation. Moreover, the authors proposed some heuristics to estimate the variance vector.

- **Stochastic hill climbing with learning by vectors of normal distributions:** It was developed by (Rudlof and Köppen, 1996) specifically for the continuous domain. The parameters of the density function, the mean vector $\hat{\mu}$ and the variance vector $\hat{\sigma}$ are estimated using:

$$\hat{\mu}^{g+1} = \hat{\mu}^g + \alpha(b^g - \hat{\mu}^g) \tag{5.18}$$

$$\hat{\sigma}^{g+1} = \beta \times \hat{\sigma}^g \tag{5.19}$$

where α denotes the learning factor, b^g denotes the barycenter of the B best individuals in the g^{th} generation and $0 < \beta < 1$ denotes a fixed constant.

- **Univariate marginal distribution algorithm:** This algorithm, proposed by (Mühlenbein, 1997), behaves differently from the two previous algorithms. It estimates the joint probability distribution $p^g(x)$ of the selected individuals at each generation. In the case of binary variables, the probability vector is estimated from marginal frequencies and $p^g(x_i)$ is set by counting the number of occurences of "1", $f_k(x_i = 1)$ for $k = 1, 2, ..., Q$, in the set of selected individuals. In order to generate new individuals, each variable is generated according to $p^g(x_i)$ as follows

$$p^g(x_i) = \frac{1}{Q} \sum_{k=1}^{Q} f_k(x_i = 1) \tag{5.20}$$

  For continuous domains, the Univariate Marginal Distribution Algorithm is designed, through statistical tests, to find the density function that best fits the variables. Then, using the maximum likelihood estimates, the evaluation of the parameters is performed.

5.3.2 Bivariate models

In order to make the interactions between variables more realistic, this class of models takes into account pairwise dependencies. In this class of EDA, we focus only on the Mutual Information Maximization for Input Clustering (MIMIC) proposed by (Bonet et al., 1996) as it is used for both continuous and discrete domains. In MIMIC, the conditional dependencies of $p^g(x)$ are defined by a Markovian chain in which each variable is conditioned by the previous one. Therefore, in each generation, the MIMIC uses a permutation framework of ordered pairwise conditional probabilities, which can be written as follows:

$$p^g_\pi(x) = p^g(x_{i_1}|x_{i_2})p^g(x_{i_2}|x_{i_3}) \dots p^g(x_{i_{n-1}}|x_{i_n}) \tag{5.21}$$

where $\pi = \{i_1, i_2, ..., i_n\}$ is a permutation of the indexes $1, 2, ..., n$. The objective is to find the best $p^g_\pi(x)$ as closely as possible to the complete joint probability $p^g(x) = p^g(x_1|x_2, \dots, x_n)p^g(x_2|x_3, \dots, x_n) \dots p^g(x_{n-1}|x_n)p^g(x_n)$. The degree of similarity between $p^g_\pi(x)$ and $p^g(x)$ is measured bu using the Kullback-Leibler distance. The same idea was used by (Larrañaga et al., 2000) to extend this algorithm to the continuous space.

5.3.3 Multivariate models

This section discusses the models that do not impose any restriction about the dependencies among variables.

- **Estimation of multivariate normal density algorithms (EMNA):** It was developed by (Larrañaga et al., 2001). At each generation g, the multivariate normal density function is estimated. Therefore, the vector of mean $\hat{\mu}^g$ and the variance-covariance matrix $\hat{\sigma}^g$ are estimated using their maximum likelihood estimates:

$$\hat{\mu}_k^g = \frac{1}{Q}\sum_{r=1}^{Q} x_{k,r}^g \qquad k = 1, 2, ..., Q \tag{5.22}$$

$$(\hat{\sigma}_k^g)^2 = \frac{1}{Q}\sum_{r=1}^{Q}(x_{k,r}^g - \hat{\mu}_k^g)^2 \qquad k = 1, 2, ..., Q \tag{5.23}$$

$$(\hat{\sigma}_{j,k}^g)^2 = \frac{1}{Q}\sum_{r=1}^{Q}(x_{j,r}^g - \hat{\mu}_j^g)(x_{k,r}^g - \hat{\mu}_k^g) \qquad j \neq k = 1, 2, ..., Q. \tag{5.24}$$

  Finally, the new individuals are generated following the estimated function. An adaptive version of this algorithm was also developed by (Larrañaga et al., 2001). In this algorithm, the first model is estimated according to the multivariate normal density function. Next, one individual $x_{current}^g$ is generated from the current density function. Depending on the fitness of this individual, it will be kept for the next population or not. If the answer is yes, the new individual is introduced in the population, and it is necessary to update the parameters of the multivariate normal density function as follows:

$$\hat{\mu}^{g+1} = \hat{\mu}^g + \frac{1}{Q}(x_{current}^g - x_Q^g) \tag{5.25}$$

$$
\begin{aligned}
(\hat{\sigma}_{j,k}^{g+1})^2 &= (\hat{\sigma}_{j,k}^{g})^2 - \frac{1}{Q^2}(x_{k,current}^{g} - x_{k,Q}^{g}) \sum_{r=1}^{Q} (x_{j,r}^{g} - \hat{\mu}_{j}^{g}) \\
&- \frac{1}{Q^2}(x_{j,current}^{g} - x_{j,Q}^{g}) \sum_{r=1}^{Q} (x_{k,r}^{g} - \hat{\mu}_{k}^{g}) \\
&+ \frac{1}{Q^2}(x_{k,current}^{g} - x_{k,Q}^{g})(x_{j,current}^{g} - x_{j,Q}^{g}) \\
&- \frac{1}{Q}(x_{k,Q}^{g} - \hat{\mu}_{k}^{g+1})(x_{j,Q}^{g} - \hat{\mu}_{j}^{g+1}) \\
&+ \frac{1}{Q}(x_{k,current}^{g} - \hat{\mu}_{k}^{g+1})(x_{j,current}^{g} - \hat{\mu}_{j}^{g+1})
\end{aligned}
\tag{5.26}
$$

Moreover, the authors proposed an incremental version of EMNA. The main differences comparing to the previous one are that each generated individual is added to the population regardless of its fitness and the update rules are given by:

$$
\hat{\mu}^{g+1} = \frac{Q^g}{Q^g + 1}\hat{\mu}^{g} + \frac{1}{Q^g + 1}x_{current}^{g} \tag{5.27}
$$

$$
(\hat{\sigma}_{j,k}^{g+1})^2 = \frac{Q_g}{Q_g + 1}\hat{\sigma}_{j,k}^{g} + \frac{1}{Q_g + 1}(x_{k,current}^{g} - \hat{\mu}_{k}^{g})(x_{j,current}^{g} - \hat{\mu}_{j}^{g}) \tag{5.28}
$$

It should be noted that the size of the population increases as the algorithm evolves.

- **Estimation of Gaussian network algorithms:** This algorithm was developed by (Larrañaga et al., 2000). The first step is to induce the Gaussian network from the data. The authors present three different induction models: edge–exclusion tests, Bayesian score + search and penalized maximum likelihood + search. Once the induction is done, a new individual is created according to the scheme of the learned network.

- **Iterative density-estimation evolutionary algorithm:** It was proposed by (Bosman and Thierens, 1999). It uses the Bayesian factorization and mixture distributions for learning probabilistic models. Moreover, the iterative density-estimation evolutionary algorithm uses the truncated distribution for sampling the new individuals and only part of the population is replaced in each generation.

5.4 Estimation of distribution algorithm with reinforcement learning

Generally speaking, the estimation of distribution algorithm proceeds as follows. First, an initial population of candidate solutions is generated randomly. Then, a subset of solutions is selected from the initial population. At this moment comes the main step of the algorithm, consisting of building a probability model based on the selected solutions to create a new "good" solution. Finally, the step of replacement decides if the new solution should be kept or not. The algorithm continues until it reaches a stopping criterion. In our context, we propose to apply the EDA to find good values for the penalties p_j^k.

Therefore, the problem considered in this section consists of finding the best values for the matrix of the the penalties p_j^k to be used in our dynamic assignment problem. This current problem will be referred as the Penalties Calibration Problem (PCP). Our proposed algorithm follows these steps:

- **Encoding solution:** A solution of the PCP problem is encoded by a matrix π where the rows represent the parking $j \in J$ and the columns correspond to the time periods $t = 1, 2..., T$. The intersection between each row j and each column t represents the penalty term p_j^t. Figure(5.2).

$$\pi = \begin{pmatrix} p_1^1 & p_1^2 & ... & p_1^T \\ p_2^1 & p_2^2 & ... & p_2^T \\ : & : & ... & : \\ : & : & ... & : \\ p_m^1 & p_m^2 & ... & p_m^T \end{pmatrix}$$

Figure 5.2: Encoding solution

- **Evaluation and forecasting:** Each solution of the PCP problem is evaluated according to the objective function described in (5.15). The evaluation of any given solution of the PCP is the total assignment cost, over a day, for the dynamic assignment problem with the corresponding penalties. That is to say, a solution of the PCP is a set of penalties. These penalties are used in equation (5.15); the

problem is solved for all the decision points, the total cost of period t is given by $f^t(x) = \sum_{k=1}^{K} f^k(x)$ and the total cost of a day d is $f^d(x) = \sum_{t=1}^{T} f^t(x)$. After that, a smoothing technique is used to take into account the forecasting of the demand. At the end, the evaluation of any solution π of PCP is given by the following equation:

$$F(x) = \sum_{q=0}^{\delta} \alpha(1-\alpha)^q f^{d-q}(x), \tag{5.29}$$

where $0 < \alpha < 1$ is the smoothing factor which represents the weight of the previous observations. Therefore, the EDA consists to minimize $F(x)$.

- **Initial population:** The initial population of P solutions is randomly generated. This means that we generate P matrices π such that each penalty $p^t_{j,r}$ (penalty for parking j during period t in the PCP solution r) of matrix π_r is generated according to a uniform distribution.

- **Selection:** From the initial population we propose to select Q solutions according to the ranking of the objective functions $F(x)$ defined in equation (5.29).

- **Probabilistic model:** In order to generate new candidate solutions, in our proposition we use the probabilistic model of the univariate marginal distribution algorithm for Gaussian models (Larrañaga and Lozano, 2002) where the parameters mean and standard deviation of a solution are extracted from population information during the optimization process. The two parameters to be estimated at each generation for each variable are the mean $\hat{\mu}_j$ and the standard deviation $\hat{\sigma}_j$. Their respective maximum likelihood estimates are:

$$\hat{\mu}^t_j = \bar{p^t_j} = \frac{1}{Q} \sum_{r=1}^{Q} p^t_{j,r} \tag{5.30}$$

$$\hat{\sigma}^t_j = \sqrt{\frac{1}{Q} \sum_{r=1}^{Q} (p^t_{j,r} - \bar{p^t_j})^2} \tag{5.31}$$

- **Replacement:** We compare the new solution with the worst solution in the current population. If the new solution is best than this solution, then the worst solution is removed from the population and it is replaced with the new one.

- **Stopping criterion:** The stopping criterion indicates when the search finishes. We set a maximum number of iterations and a maximal computational time in our algorithm.

- **Local search procedure for dynamic assignment problem:** The local search procedure was proposed to improve the performance of the algorithm through problem decomposition. Instead of tackling the whole complex assignment problem at the same time, the problem is divided into a set of smaller sub-problems, each of which can be solved easily in terms of computational time. The purpose of the decomposition scheme is to break down a large problem into smaller ones.

  If the number of vehicles that appear in the system and the number of considered parking are large, then the number of variables and constraints taken into account for solving the whole problem at each decision point may be huge. The idea is to decompose the area of the system (city) into a set of regions and solve an assignment problem for each one of them. At each iteration, a region is selected randomly, we record the set of requests Ω from vehicles in this region and the set of parking lots L existing in the same region.Then, the problem formulated by the associated variables is solved while taking into account the assignments of the remaining regions.

  Therefore, we set a threshold on the number of vehicles n_{max} from which the local search procedure is applied. The procedure starts from an initial solution randomly generated. Then, we select at random some decision variables to be fixed and optimize the remaining sub-problem, according to the objective function. The process is repeated until a given stopping criterion is reached (Algorithm 14).

  The framework of the proposed algorithm with forecast is given in Figure 5.3 and Algorithm 15. It should be noted that if we set π_{best} to 0, we obtain the assignment algorithm without forecasting process. We denote by AA_{EDA} and AA the assignment algorithm with and without EDA, respectively.

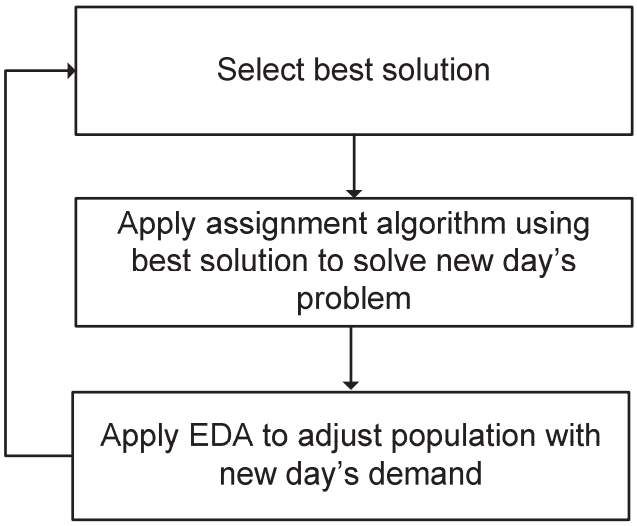

Figure 5.3: The proposed algorithm

Algorithm 14: Pseudo-code of the local search procedure

repeat

Select a set L of parking slots at random at the decision point k;

Find the set Ω associated to those spaces at the decision point k;

Solve the assignment problem of (Ω, L, π_{best});

until *A stopping criterion is met*;

Algorithm 15: Pseudo-code of the assignment algorithm with forecast process based on EDA

$R_0 = \oslash$;

for $k = 1, 2, ..., (K \times T \times D)$ **do**

Find N_k, R_k and loc_i^k;

$E_k = \{N_k \bigcup R_{k-1}\}$;

if $|E_k| < n_{max}$ **then**

(A_k, R_k)=Apply assignment algorithm $\left(E_k, J_j^k, \pi_{best}\right)$;

else

(A_k, R_k)=Apply local search procedure $\left(E_k, J_j^k, \pi_{best}\right)$;

Update loc_i^k of R_k

5.5 Computational results

In our experiments, we developed a simulation environment using the C++ programming language to reproduce the features of a real world problem. The simulation tests are

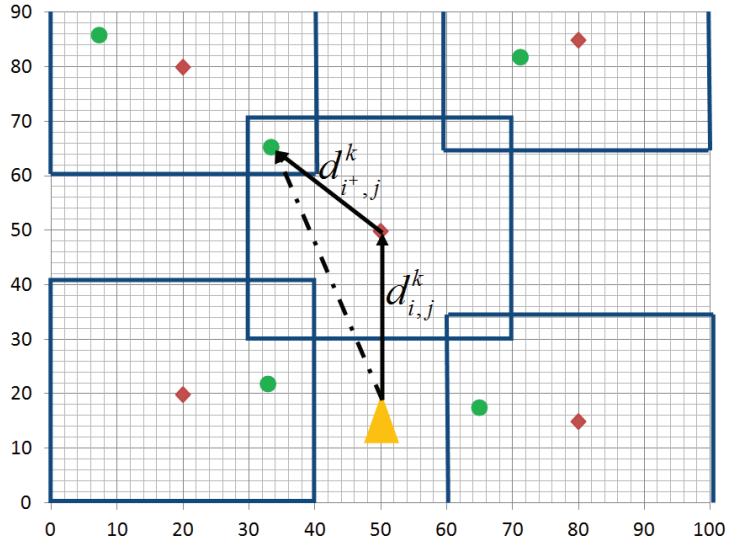

Figure 5.4: Map with 5 parkings

generated in the two dimensional Euclidean space with different number of parkings. We assume that we have the map of the locations of the parking. Figures 5.4-5.8 present these maps with 5, 7, 10, 15 and 20 parking, respectively. Each figure represents a problem instance. The parking are denoted by red diamonds. Then, each parking represents the center of a region of 20 square sizes.

In Figure 5.4, the regions are denoted by the blue squares. These regions define the density of the parking slots at each period $t = 1, 2, \ldots, 6$ at each day, and may overlap.

In the simulation, we consider that the parking opens at 07:00 a.m. and closes at 07:00 p.m. That is to say, each day consists of 6 periods of two hours and the time horizon of the simulations lasts for 120 days. It is assumed that all the parking have the same capacity of 200 slots. Moreover, the number of occupied slots in each parking, at the beginning of each day, is generated according to the uniform distribution in the range [50, 100]. Initially, the parkings are partially occupied as some vehicles can stay overnight in the parking slots. We assume that all parking slots can be used by any vehicle without any time limit. If a vehicle is assigned to a parking, the system selects any available slot in that parking. The frequencies of enter/exit of each parking for each region are randomly generated for each period according to the normal distribution, with the means provided in Tables 1-5 for each instance problem and the variance equal 0.5. These frequencies define the number of requests associated to each region. For generating the number of requests we used the truncated normal distribution because we only wish to consider data within a particular

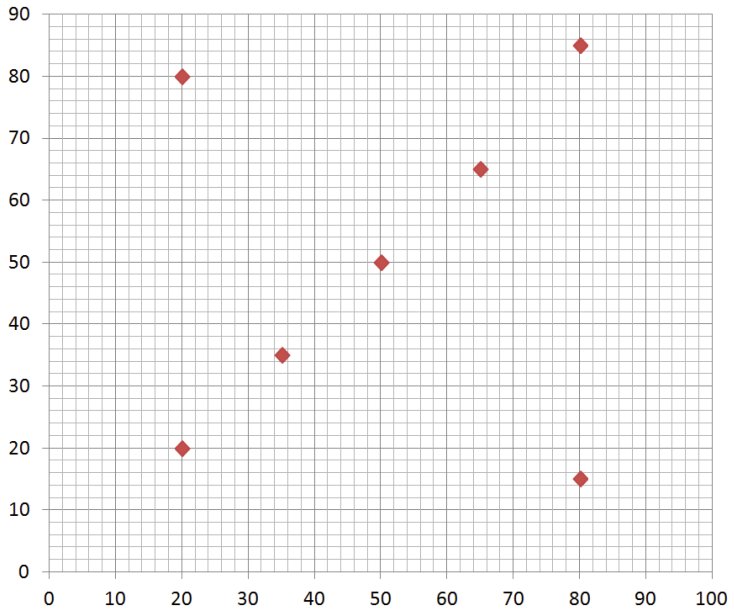

Figure 5.5: Map with 7 parkings

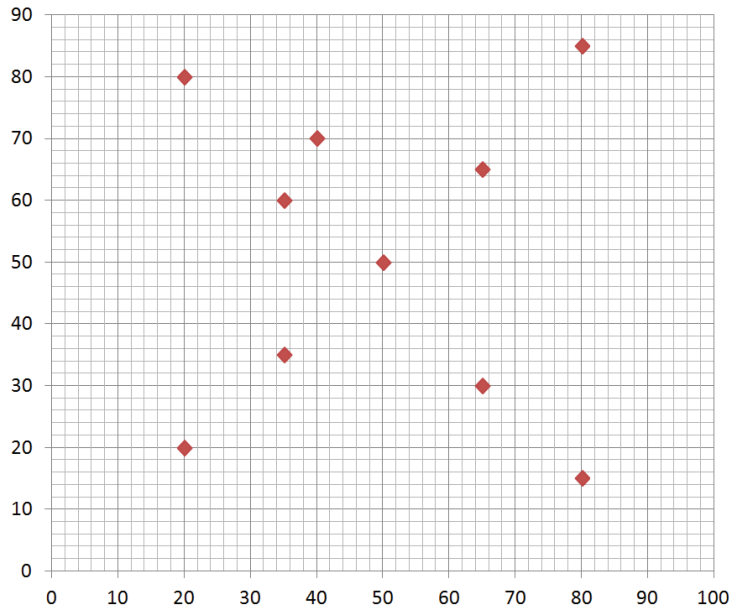

Figure 5.6: Map with 10 parkings

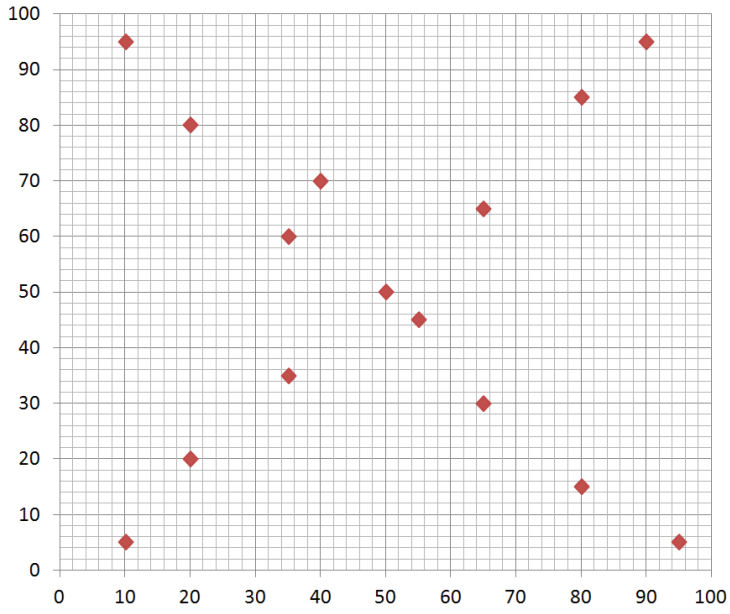

Figure 5.7: Map with 15 parkings

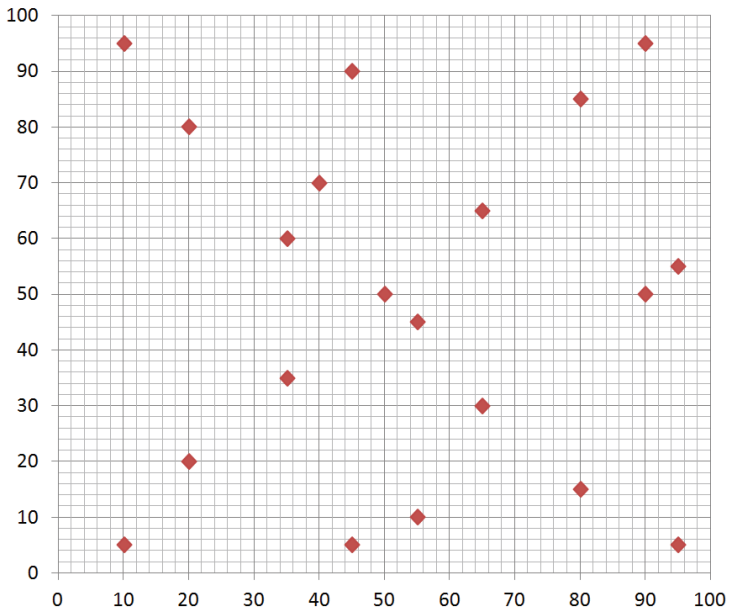

Figure 5.8: Map with 20 parkings

Instance		t_1	t_2	t_3	t_4	t_5	t_6
Parking 1	Enter	1	5	2	2	2	1
	Exit	1	2	1	1	4	1
Parking 2	Enter	2	2	5	3	1	1
	Exit	4	4	5	3	1	1
Parking 3	Enter	4	1	2	2	1	2
	Exit	2	1	2	2	2	1
Parking 4	Enter	2	1	3	4	4	2
	Exit	2	1	3	2	4	1
Parking 5	Enter	2	2	2	1	3	3
	Exit	2	1	2	1	2	4

Table 5.1: Instance problem with 5 parkings

range of interest to us. It should be noted that these parameters were set experimentally and the same distribution is used for all days. Each time period t, is partitioned into 120 equally spaced decision points where the decisions take place, *i.e.* each minute the problem will be solved taking into account the frequencies of enter/exit of each parking. The same distribution is used for all decision points $k = 1, 2, \ldots, K = 120$ between two consecutive time periods. For each generated request, a vehicle and its destination appear on the map. The location of the vehicle can be generated anywhere and the location of the destination must be generated within the considered region. We note that more than one vehicle may have the same destination.

In Figure 5.4, the vehicle and the destinations are denoted by the yellow triangle and the green circle respectively. Therefore, for each region, for each decision point, the total number of requests is computed by adding the number of new generated requests and the number of requests not assigned at the previous decision point. The vehicles not assigned in the previous period are assumed to move in the direction of their destination, as denoted by the dashed line in Figure 5.4. Thus, their distances $d_{i,j}^k$ are recomputed.

The performance measure employed in our numerical study was the average relative percentage deviation in terms of the objective functions at each day d:

$$\Delta_d = 100 \times \frac{f^d(AA_{EDA}) - f^d(AA)}{f^d(AA)},$$

where AA denotes the proposed algorithm without the learning factor and AA_{EDA} the proposed algorithm with learning factor. As mentioned above, the total assignment cost of period t in a given day is, $f^t(x) = \sum_{k=1}^{K} f^k(x)$ and the total cost of each day d,

Instance		t_1	t_2	t_3	t_4	t_5	t_6
Parking 1	Enter	1	1	2	1	2	2
	Exit	1	1	2	1	5	2
Parking 2	Enter	1	1	2	3	5	2
	Exit	1	1	5	4	3	2
Parking 3	Enter	4	2	4	1	1	3
	Exit	2	1	3	1	1	2
Parking 4	Enter	2	4	2	3	2	1
	Exit	2	3	2	3	2	2
Parking 5	Enter	2	2	2	2	5	2
	Exit	2	2	1	2	4	2
Parking 6	Enter	1	1	5	1	1	1
	Exit	2	2	4	1	2	2
Parking 7	Enter	5	1	1	1	2	1
	Exit	2	2	1	1	1	1

Table 5.2: Instance problem with 7 parkings

Instance		t_1	t_2	t_3	t_4	t_5	t_6
Parking 1	Enter	1	1	1	1	2	2
	Exit	1	2	1	1	1	1
Parking 2	Enter	2	2	1	2	1	1
	Exit	4	2	2	2	1	2
Parking 3	Enter	4	1	2	2	2	3
	Exit	2	2	1	2	1	2
Parking 4	Enter	2	1	1	2	1	2
	Exit	2	2	2	4	2	2
Parking 5	Enter	2	1	1	1	5	1
	Exit	2	2	2	2	3	2
Parking 6	Enter	1	2	1	2	5	4
	Exit	2	1	2	2	3	2
Parking 7	Enter	2	2	1	2	2	2
	Exit	1	4	1	4	2	1
Parking 8	Enter	2	3	1	2	4	1
	Exit	1	5	1	2	3	1
Parking 9	Enter	2	2	2	1	1	4
	Exit	1	2	1	1	1	3
Parking 10	Enter	2	1	2	2	4	1
	Exit	2	2	1	1	3	1

Table 5.3: Instance problem with 10 parkings

Instance		t_1	t_2	t_3	t_4	t_5	t_6
Parking 1	Enter	1	2	2	2	2	2
	Exit	1	1	1	2	2	1
Parking 2	Enter	1	1	1	5	2	3
	Exit	1	1	2	3	1	2
Parking 3	Enter	4	3	2	2	1	1
	Exit	2	5	2	1	2	1
Parking 4	Enter	2	2	1	1	3	2
	Exit	2	2	1	1	4	1
Parking 5	Enter	2	1	2	1	2	4
	Exit	2	2	1	1	1	5
Parking 6	Enter	1	1	4	1	1	1
	Exit	2	1	3	1	2	1
Parking 7	Enter	2	2	2	4	1	1
	Exit	1	4	2	5	2	1
Parking 8	Enter	2	1	2	5	1	2
	Exit	1	1	2	2	1	1
Parking 9	Enter	2	1	2	1	2	4
	Exit	1	2	1	2	2	4
Parking 10	Enter	2	1	1	3	2	2
	Exit	1	1	2	4	2	2
Parking 11	Enter	2	3	2	2	5	1
	Exit	1	2	2	1	3	1
Parking 12	Enter	2	2	4	1	2	1
	Exit	5	1	5	1	2	1
Parking 13	Enter	5	1	1	1	1	2
	Exit	4	1	1	2	2	2
Parking 14	Enter	5	1	1	1	1	1
	Exit	4	2	1	2	1	1
Parking 15	Enter	3	2	2	2	1	2
	Exit	2	5	2	2	1	1

Table 5.4: Instance problem with 15 parkings

Instance		t_1	t_2	t_3	t_4	t_5	t_6
Parking 1	Enter	1	1	2	4	2	2
	Exit	1	1	2	2	1	2
Parking 2	Enter	2	2	4	2	2	2
	Exit	4	4	5	2	1	1
Parking 3	Enter	4	1	1	2	1	2
	Exit	2	1	1	1	2	2
Parking 4	Enter	2	1	1	2	1	2
	Exit	2	2	1	2	1	1
Parking 5	Enter	2	1	2	2	2	2
	Exit	2	1	2	2	2	2
Parking 6	Enter	1	3	2	2	4	1
	Exit	2	2	2	1	2	2
Parking 7	Enter	2	2	2	2	2	2
	Exit	1	1	2	1	1	2
Parking 8	Enter	2	1	2	2	2	1
	Exit	1	1	2	1	2	2
Parking 9	Enter	2	1	2	2	2	1
	Exit	1	2	1	1	1	2
Parking 10	Enter	2	2	1	1	2	1
	Exit	2	5	1	2	5	2
Parking 11	Enter	2	2	2	1	1	5
	Exit	1	2	1	2	1	4
Parking 12	Enter	1	1	2	2	2	2
	Exit	2	1	1	2	2	2
Parking 13	Enter	5	2	1	5	2	3
	Exit	4	2	2	4	2	2
Parking 14	Enter	5	5	3	1	2	1
	Exit	4	4	2	1	1	2
Parking 15	Enter	3	1	2	1	1	1
	Exit	2	2	2	1	2	2
Parking 16	Enter	2	3	1	1	4	4
	Exit	2	4	2	2	3	4
Parking 17	Enter	2	2	4	1	1	1
	Exit	1	1	2	2	1	1
Parking 18	Enter	3	1	2	2	1	2
	Exit	3	2	1	4	2	2
Parking 19	Enter	2	3	1	1	2	1
	Exit	2	3	1	1	3	2
Parking 20	Enter	4	1	1	4	1	1
	Exit	2	1	2	3	2	2

Table 5.5: Instance problem with 20 parkings

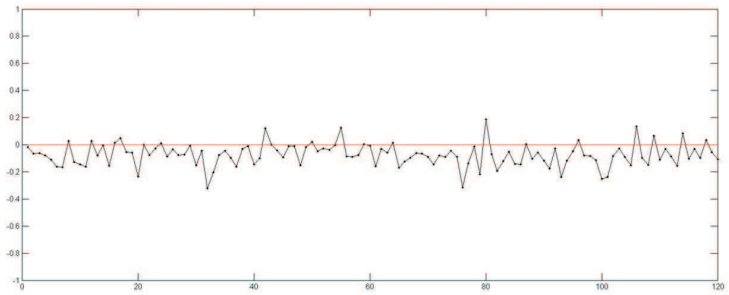

Figure 5.9: Computational results for the instance with 5 parkings

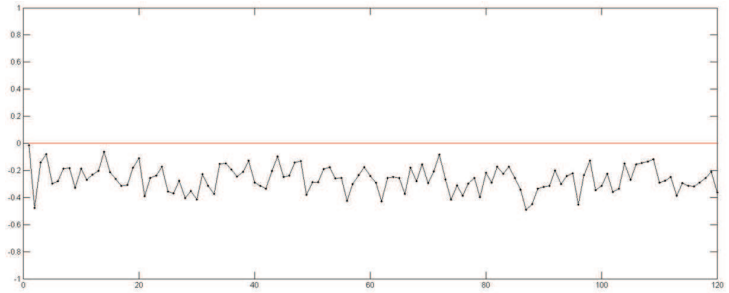

Figure 5.10: Computational results for the instance with 7 parkings

$f^d(x) = \sum_{k=1}^{6} f^t(x)$. It is assumed that all objectives have the same weights.

Figures 5.9-5.13 show the evolution of Δ_d during the 120 days of the simulation. It can be clearly seen that the values of Δ_d are below the 0-line. Therefore, the learning effect by forecasting the requests has improved the total assignment costs. Moreover, we observe that the curve begins to decline from the early periods. This shows that the learning speed of AA_{EDA} is very fast.

Furthermore, we conclude that the number of parkings has an impact on the performance of the AA_{EDA} because the small instance with 5 parkings has some peaks above the 0-line. So, the increase of the number of parkings increases the learning capacity of the algorithm.

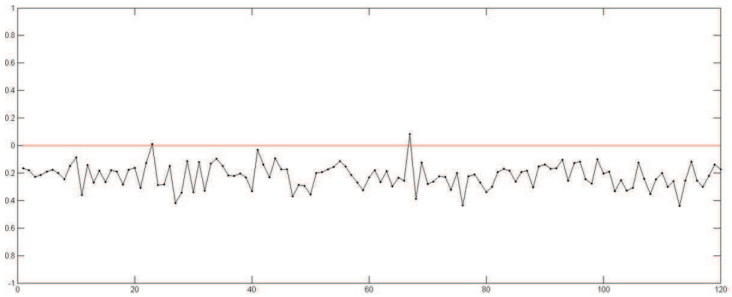

Figure 5.11: Computational results for the instance with 10 parkings

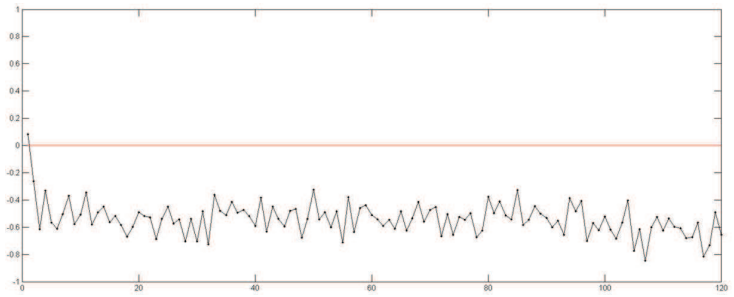

Figure 5.12: Computational results for the instance with 15 parkings

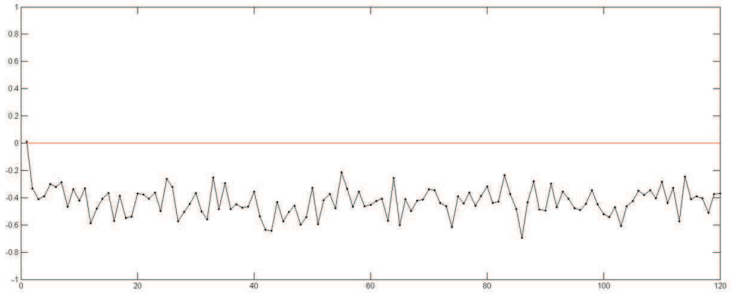

Figure 5.13: Computational results for the instance with 20 parkings

In order to compare the performance of our algorithms we used the unilateral paired $t-test$ procedure (Montgomery, 2001) at the 99% significance level. This procedure consists of comparing the means of two samples coming from paired observations. Let μ_A and μ_B denote the average of the evaluation of the fitness function for respctively, algorithm A algorithm B. The tested hypotheses are:

$$H_0 : \mu_A - \mu_B = 0$$

$$H_1 : \mu_A - \mu_B < 0$$

H_0 implies that the average relative percent deviations of the two algorithms are similar while H_1 implies that the average relative percent deviations of algorithm A are less than the ones of the algorithm B.

In our case, the algorithms A and B denote AA (without learning effect) and AA_{EDA} (with learning effect) respectively. The statistical tests prove that the negative difference between AA_{EDA} and AA is meaningful at the 99% confidence level.

Instances	$t-value$	$p-value$
5 parkings	-9.7913	1.1102e-016
7 parkings	-31.587	1.0713e-059
10 parkings	-29.319	2.7874e-056
15 parkings	-50.3	4.9424e-082
20 parkings	-45.419	5.1241e-077

Table 5.6: Unilateral paired $t-test$ at the 99% significance level ($N = 120$).

5.6 Conclusion

We approached the problem of dynamic assignment for parking slots. A driver, aiming to visit a given destination, starts looking for a parking slot by launching a request at a non-deterministic moment. The parking lot manager has to fulfil these requests by assigning available slots to vehicles. The objectives are to provide a global satisfaction to all customers and to maximize the parking lots occupancy. The problem is dynamic, the requests and the parking lots change over the time. First, the problem is modelled as a sequence of consecutive assignment problems, over the time. These problems are inter-related. At each decision point (a small time window) a static assignment problem is solved, we assign non-handled requests up-to-this point to the current available slots. Second, as the assignments that are made at earlier periods affect which assignments can be made during later periods, we propose to establish a forecasting process based on a learning effect. We introduce penalty terms in the objective function. The values of these penalties are calibrated through a learning process using the Estimation of Distribution Algorithm (EDA). We notice that solving each assignment problem at every decision point can be time consuming, depending on the number of the concerned requests and available slots. A local search procedure is proposed to improve the performance of our approach through problem decomposition and, hence, to reduce the solving time. Instead of tackling the whole complex assignment problem at the same time, the problem is divided into a set of smaller sub-problems. We tested our approach with and without the learning effect. Our approach is efficient since we were able to manage a set of parking lots, of up-to 20 parking lots, during a horizon of 120 days, which corresponds to assignment problems

with up-to 4000 parking slots to manage and 86400 requests to handle. The results also show the benefit of the learning effect. The total cost of the solutions with learning effect is less than the cost of the solutions without learning effect (a student test is used to prove the difference between these two methods).

Chapter 6

Conclusion and further research

6.1 Contributions of the thesis

This thesis contributes to the improvement of the parking management system. In particular, we studied the multi-objective assignment problem of tasks to agents in static and dynamic environments, where tasks correspond to drivers or cars, agents correspond to parking or parking slots, and the objective is to satisfy both the drivers and the parking managers. This thesis proposes efficient and robust algorithms that help to save time and money for drivers and to increase the income of parking managers. The problem is formulated as a multi-objective assignment problem in static and dynamic environments. The main contributions of our work to this management problem can be summarized as follows. We developed four new two-phase heuristics to approximate the set of the efficient solutions for the bi-objective shortest path problem. The first phase uses a standard dichotomic algorithm to generate the supported efficient set. For the second phase, we developed four metaheuristics to generate an approximation of the non-supported efficient solutions: a cost perturbation method, a path relinking, a genetic algorithm and a hybrid genetic algorithm. The proposed approaches were tested on instances of the Bi-objective Shortest Path problem and the results are significant. The percentage of true Pareto-optimal solutions generated by the proposed algorithms ranges, approximately, between 87% and 100%, and the CPU time consumed to obtain these solutions is of 4.6 seconds for the networks with 1000 nodes and 53.66 seconds for the networks with 5000 nodes, which is faster than the labeling algorithm BRUM. We applied the same heuristics devel-

oped in Chapter 3 to the bi-objective assignment problem. The experimental results show the efficiency of our proposed algorithms for large instances in comparison with an exact method from the literature. We tackled the dynamic assignment problem for parking slots. We developed two approaches. In the first approach, the problem was modeled as a sequence of consecutive static assignment problems; in the second approach, we introduced a reinforcement learning method where history an assignment over several days are taken into account. For the learning process, we developed an Estimation of Distribution Algorithm using a Gaussian Univariate Marginal Distribution. In these approaches, the objectives are aggregated in a single objective function to keep the focus on the resolution technique. The aggregation is done to find a best compromise solution that takes into account the preferences of the parking manager, favoring one objective over another according to the state of the supply and demand. A platform for dynamic deployment of services can dynamically switch from one approach to another depending on the context. We tested our approach with and without the learning effect. Our approach is efficient since we were able to manage a set of parking lots, of up-to 20 lots, during a horizon of 120 days, which corresponds to an assignment problem with up-to 4000 parking slots to manage and 86400 requests to handle. The results also show the benefits of the learning effect. The total cost of the solutions with learning effect is smaller than the one of the solutions without learning effect.

6.2 Future research directions

The potential research directions may be summarized as follows. The integration of the proposed heuristics, for the shortest path and the dynamic assignment problems could be implemented in the framework for Context Aware Transportation Services (CATS), which could be used for real problems. In order to improve the proposed algorithms, certain periodical events can be taken into account. Therefore the system must be able to configure the algorithm before certain problems occur or when there is a peak demand.

The various proposals could be validated both by simulation and in a realistic environment using data from the network of sensors deployed on the parking LAMIH in the CISIT[1] / SYFRA project. We found that the number of non-dominated solutions could

be very important. An interesting problem is then to sample the Pareto front to offer the user a subset of representative solutions.

The extension of the implemented Estimation of Distribution Algorithm (EDA) to the multi-objective context. In our case, we aggregate the three objectives (the distance to parking, the distance from the parking to the drivers destination and the waiting time for getting a parking slot). However, it will be more interesting to consider the problem as a multi-objective one.

The EDA could be improved through a multivariate distribution model or a Gaussian Network. In our case, we supposed independent penalties, but in real life there is a dependency between them. This is due to the fact that the demands around the parking lots for different periods are interrelated. Some events are cyclical, and in order to take them into account, it is necessary to develop a long-term memory or knowledge base in EDA algorithm to better predict future demands. The developed heuristics for our parking dynamic assignment problem could be easily adapted to other dynamic assignment problems like cab company, task assignment in shops, etc.

1:**Campus International sur la Sécurité et l'Intermodalité dans les Transports.**

Bibliography

Chang Wook Ahn and Rudrapatna S Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *Evolutionary Computation, IEEE Transactions on*, 6(6):566–579, 2002.

Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

Maurice Allais. Pareto, vilfredo: contributions to economics. 1968.

Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Science*, 25:73–78, 1979.

Richard Arnott and John Rowse. Downtown parking in auto city. *Regional Science and Urban Economics*, 39(1):1–14, 2009.

Richard Arnott, Tilmann Rave, and Ronnie Schöb. Alleviating urban traffic congestion. *MIT Press Books*, 1, 2005.

Kay W Axhausen and John W Polak. Choice of parking: stated preference approach. *Transportation*, 18(1):59–81, 1991.

Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar Dasgupta, and Jie Lin. Parking slot assignment games. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 299–308. ACM, 2011.

S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, 1994.

Lakshmi Dhevi Baskar, Bart De Schutter, J Hellendoorn, and Zoltan Papp. Traffic control and intelligent vehicle highway systems: a survey. *IET Intelligent Transport Systems*, 5(1):38–52, 2011.

Prithwish Basu and Thomas DC Little. Networked parking spaces: architecture and applications. In *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, volume 2, pages 1153–1157. IEEE, 2002.

Itzhak Benenson, Karel Martens, and Slava Birfir. Parkagent: An agent-based model of parking in the city. *Computers, Environment and Urban Systems*, 32(6):431–439, 2008.

Harold P Benson. Existence of efficient solutions for vector maximization problems. *Journal of Optimization Theory and Applications*, 26(4):569–580, 1978.

Subir Biswas, Raymond Tatchikou, and Francois Dion. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. *Communications Magazine, IEEE*, 44(1):74–82, 2006.

Jeremy S. De Bonet, Charles L. Isbell, Jr., and Paul Viola. Mimic: Finding optima by estimating probability densities. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, page 424. The MIT Press, 1996.

P. A. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, volume I, pages 60–67. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

V Joseph Bowman Jr. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple criteria decision making*, pages 76–86. Springer, 1976.

J Brumbaugh-Smith and D Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43(2):216–224, 1989.

Rainer E Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems, Revised Reprint*. Siam, 2009.

Felix Caicedo. The use of space availability information in parc systems to reduce search times in parking facilities. *Transportation Research Part C: Emerging Technologies*, 17 (1):56–68, 2009.

Felix Caicedo. Real-time parking information management to reduce search time, vehicle displacement and emissions. *Transportation Research Part D: Transport and Environment*, 15(4):228–234, 2010.

Murat Caliskan, Daniel Graupner, and Martin Mauve. Decentralized discovery of free parking places. In *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, pages 30–39. ACM, 2006.

Murat Caliskan, Andreas Barthels, Björn Scheuermann, and Martin Mauve. Predicting parking lot occupancy in vehicular ad hoc networks. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 277–281. IEEE, 2007.

James F Campbell and André Langevin. The snow disposal assignment problem. *Journal of the Operational Research Society*, pages 919–929, 1995.

Nicolas Cenerario, Thierry Delot, and Sergio Ilarri. A content-based dissemination protocol for vanets: Exploiting the encounter probability. *Intelligent Transportation Systems, IEEE Transactions on*, 12(3):771–782, 2011.

Vira Chankong, Yacov Y Haimes, and D Gemperline. A multiobjective dynamic programming method for capacity expansion. *Automatic Control, IEEE Transactions on*, 26(5):1195–1207, 1981.

Irène Charon and Olivier Hudry. The noising method: a new method for combinatorial optimization. *Operations Research Letters*, 14(3):133–137, 1993.

Shuo-Yan Chou, Shih-Wei Lin, and Chien-Chang Li. Dynamic parking negotiation and guidance using an agent-based platform. *Expert Systems with Applications*, 35(3):805–817, 2008.

P.J. Fleming C.M. Fonseca. Multiobjective genetic algorithms. *IEEE Colloquium on 'Genetic Algorithms for Control Systems Engineering' (Digest No. 1993/130), London, UK: IEEE*, 1993.

C.A.C. Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys 32(2)*, pages 109–143, 2000.

Carlos A Coello Coello, David A Van Veldhuizen, and Gary B Lamont. *Evolutionary algorithms for solving multi-objective problems*, volume 242. Springer, 2002.

J. L. Cohon. *Multiobjective Programming and Planning*. Academic Press, New York, 1978.

Yann Collette and Patrick Siarry. *Multiobjective optimization: principles and case studies*. Springer, 2003.

H William Corley and I Douglas Moon. Shortest paths in networks with vector weights. *Journal of Optimization Theory and Applications*, 46(1):79–86, 1985.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

HG Daellenbach and CA De Kluyver. Note on multiple objective dynamic programming. *Journal of the Operational Research Society*, pages 591–594, 1980.

Amelie Y Davis, Bryan C Pijanowski, Kimberly Robinson, and Bernard Engel. The environmental and economic costs of sprawling parking lots in the united states. *Land Use Policy*, 27(2):255–261, 2010.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

Thierry Delot and Sergio Ilarri. Introduction to the special issue on data management in vehicular networks. *Transportation Research Part C: Emerging Technologies*, 23:1–2, 2012.

Thierry Delot, Nicolas Cenerario, Sergio Ilarri, and Sylvain Lecomte. A cooperative reservation protocol for parking spaces in vehicular ad hoc networks. In *Proceedings of the 6th International Conference on Mobile Technology, Application & Systems*, page 30. ACM, 2009.

Thierry Delot, Sergio Ilarri, Nicolas Cenerario, and Thomas Hien. Event sharing in vehicular networks using geographic vectors and maps. *Mobile Information Systems*, 7 (1):21–44, 2011.

Maged M Dessouky and Brian A Kijowski. Production scheduling of single-stage multi-product batch chemical processes with fixed batch sizes. *IIE transactions*, 29(5):399–408, 1997.

Yanchao Dong, Zhencheng Hu, Keiichi Uchimura, and Nobuki Murayama. Driver inattention monitoring system for intelligent vehicles: A review. *Intelligent Transportation Systems, IEEE Transactions on*, 12(2):596–614, 2011.

L. Thiele E. Zitzler. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3(4)*, pages 257–271, 1999.

Matthias Ehrgott. *Multicriteria optimization*, volume 2. Springer, 2005.

Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-Spektrum*, 22(4):425–460, 2000.

Matthias Ehrgott and Xavier Gandibleux. Multiobjective combinatorial optimization: theory, methodology, and applications. In *Multiple criteria optimization: State of the art annotated bibliographic surveys*, pages 369–444. Springer, 2002.

Erhan Erkut, Stevanus A Tjandra, and Vedat Verter. Hazardous materials transportation. *Handbooks in operations research and management science*, 14:539–621, 2007.

José Figueira, Salvatore Greco, and Matthias Ehrgott. *Multiple criteria decision analysis: state of the art surveys*, volume 78. Springer, 2005.

Liping Fu. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transportation Research Part B: Methodological*, 35(8):749–765, 2001.

Giorgio Gallo and Stefano Pallottino. Shortest path algorithms. *Annals of Operations Research*, 13(1):1–79, 1988.

Mitsuo Gen and Lin Lin. Multi-objective hybrid genetic algorithm for bicriteria network design problem. *Complexity International*, 11(11):73–83, 2005.

Mitsuo Gen, Runwei Cheng, and Qing Wang. Genetic algorithms for solving shortest path problems. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 401–406. IEEE, 1997.

Yanfeng Geng and Christos G Cassandras. Dynamic resource allocation in urban settings: A smart parking approach. In *Computer-Aided Control System Design (CACSD), 2011 IEEE International Symposium on*, pages 1–6. IEEE, 2011.

Yanfeng Geng and Christos G Cassandras. A new smart parking system infrastructure and implementation. *Procedia-Social and Behavioral Sciences*, 54:1278–1287, 2012.

Arthur M Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of Mathematical Analysis and Applications*, 22(3):618–630, 1968.

Keivan Ghoseiri and Behnam Nadjari. An ant colony optimization algorithm for the bi-objective shortest path problem. *Applied Soft Computing*, 10(4):1237–1246, 2010.

Vincent Editeur Giard and Bernard Roy. *Méthodologie multicritère d'aide à la décision*. Editions Economica, 1985.

Phillip B Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. Irisnet: An architecture for a worldwide sensor web. *Pervasive Computing, IEEE*, 2(4):22–33, 2003.

Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

D. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison–Wesley, 1989.

F Guerriero and R Musmanno. Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications*, 111(3):589–613, 2001.

G.G. Yen H. Lu. Rank-density-based multiobjective genetic algorithm and benchmark test function study. *IEEE Transactions on Evolutionary Computation 7(4)*, pages 325–343, 2003.

Michael R Hafner, Drew Cunningham, Lorenzo Caminiti, and Domitilla Del Vecchio. Cooperative collision avoidance at intersections: Algorithms and experiments. *Intelligent Transportation Systems, IEEE Transactions on*, 14(3):1162–1175, 2013.

Yacov Y Haimes, LS Ladson, and David A Wismer. Bicriterion formulation of problems of integrated system identification and system optimization, 1971.

Pierre Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

J. H. Holland. Adaptation in natural and artificial systems. *University of Michigan, Press, Ann Arbor*, 1975.

Jun Inagaki, Miki Haseyama, and Hideo Kitajima. A genetic algorithm for determining multiple routes and its applications. In *Circuits and Systems, 1999. ISCAS'99. Proceedings of the 1999 IEEE International Symposium on*, volume 6, pages 137–140. IEEE, 1999.

D.E. Goldberg J. Horn, N. Nafpliotis. A niched pareto genetic algorithm for multiobjective optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, 27-29. Orlando, FL, USA: IEEE*, 1994.

D.W. Corne J.D. Knowles. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation 8(2)*, pages 149–172, 1985.

Joshua Knowles and David Corne. On metrics for comparing nondominated sets. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 711–716. IEEE, 2002.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Pedro Larrañaga, Ramon Etxeberria, José A. Lozano, and José M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. In A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 201–204, 2000.

Pedro Larrañaga, Jose A. Lozano, and Endika Bengoetxea. Estimation of distribution algorithms based on multivariate normal distributions and Gaussian networks. Technical report, Dept. of Computer Science and Artificial Intelligence, University of Basque Country, 2001.

P. Larrañaga and J.A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

Thananchai Leephakpreeda. Car-parking guidance with fuzzy knowledge-based decision making. *Building and environment*, 42(2):803–809, 2007.

H. Li, Q. Zhang, E. Tsang, and J.A. Ford. Hybrid Estimation of Distribution Algorithm for Multiobjective Knapsack Problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 145–154. 2004.

Lin Lin and Mitsuo Gen. Multiobjective genetic algorithm for bicriteria network design problems. In *Intelligent and Evolutionary Systems*, pages 141–161. Springer, 2009.

Linzhong Liu, Haibo Mu, Xinfeng Yang, Ruichun He, and Yinzhen Li. An oriented spanning tree based genetic algorithm for multi-criteria shortest path problems. *Applied Soft Computing*, 12(1):506–515, 2012.

Rita Malhotra, HL Bhatia, and MC Puri. Bi-criteria assignment problem. *Operations Research*, 19(2):84–96, 1982.

Chris Manzie, Harry Watson, and Saman Halgamuge. Fuel economy improvements for urban driving: Hybrid vs. intelligent vehicles. *Transportation Research Part C: Emerging Technologies*, 15(1):1–16, 2007.

Ernesto de Queros Vieira Martins and JLE Santos. The labelling algorithm for the multiobjective shortest path problem. *Departamento de Matematica, Universidade de Coimbra, Portugal, Tech. Rep. TR-99/005*, 1999.

Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

Patrick McKeown and Brian Workman. A study in using linear programming to assign students to schools. *Interfaces*, 6(4):96–101, 1976.

Naourez Mejri, Mouna Ayari, and Farouk Kamoun. An efficient cooperative parking slot assignment solution. In *UBICOMM 2013, The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 119–125, 2013.

Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer, 1999.

Cheikh Mohamed, Jarboui Bassem, and Loukil Taicir. A genetic algorithms to solve the bicriteria shortest path problem. *Electronic Notes in Discrete Mathematics*, 36:851–858, 2010.

Peter Mooney and Adam Winstanley. An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science*, 20 (4):401–423, 2006.

John Mote, Ishwar Murthy, and David L Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53(1):81–92, 1991.

Heinz Mühlenbein. The equation for response to selection and its use for prediction. *Evol. Comput.*, 5(3):303–346, 1997.

Heinz Mühlenbein and Gerhard Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 178–187, London, UK, UK, 1996. Springer-Verlag. ISBN 3-540-61723-X. URL `http://dl.acm.org/citation.cfm?id=645823.670694`.

K. Deb N. Srinivas. Multiobjective optimization using nondominated sorting in genetic algorithms. *Journal of Evolutionary Computation 2(3)*, pages 221–248, 1994.

Joao Carlos Namorado Climaco and Ernesto Queiros Vieira Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11(4):399–404, 1982.

José Maria A Pangilinan and Gerrit K Janssens. Evolutionary algorithms for the multi-objective shortest path problem. *Enformatika*, 19, 2007.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.

T.K. Paul and H. Iba. Linear and combinatorial optimizations by estimation of distribution algorithms, 2002.

Christian Roed Pedersen, Lars Relund Nielsen, and Kim Allan Andersen. The bicriterion multimodal assignment problem: introduction, analysis, and experimental results. *INFORMS Journal on Computing*, 20(3):400–411, 2008.

NV Phillips. An application of multicriteria optimization to assignment university personnel to parking lots. *ORSA/TIMS*, 1985.

Dana Popovici, Mikael Desertot, Sylvain Lecomte, and Nicolas Peon. Context-aware transportation services (cats) framework for mobile environments. *International Journal of Next-Generation Computing*, 2(1), 2011.

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. Two phase algorithms for the bi-objective assignment problem. *European Journal of Operational Research*, 185(2):509–533, 2008.

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3):149–165, 2010.

C. Newton R. Sarker, K.H. Liang. A new multiobjective evolutionary algorithm. *European Journal of Operational Research 140(1)*, pages 12–23, 2002.

Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009a.

Andrea Raith and Matthias Ehrgott. A two-phase algorithm for the biobjective integer minimum cost flow problem. *Computers & Operations Research*, 36(6):1945–1954, 2009b.

V. Robles, P. de Miguel, and P. Larrañaga. Solving the traveling salesman problem with edas. In *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pages 211–229. 2002.

Stephan Rudlof and Mario Köppen. Stochastic hill climbing with learning by vectors of normal distributions. pages 60–70, 1996.

Günter Rudolph. On a multi-objective evolutionary algorithm and its convergence to the pareto set. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 511–516. IEEE, 1998.

Akihito Sakai, Kozi Mizuno, T Sugimoto, and Takeshi Okuda. Parking guidance and information systems. In *Vehicle Navigation and Information Systems Conference, 1995. Proceedings. In conjunction with the Pacific Rim TransTech Conference. 6th International VNIS.'A Ride into the Future'*, pages 478–485. IEEE, 1995.

Ruhul Sarker and Carlos A Coello Coello. Assessment methodologies for multiobjective evolutionary algorithms. In *Evolutionary Optimization*, pages 177–195. Springer, 2002.

VN Sastry, TN Janakiraman, and SI Mohideen. New algorithms for multi objective shortest path problem. *Opsearch*, 40(4):278–298, 2003.

J.D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. *International Conference on Genetic Algorithm and their applications*, 1985.

Jason R Schott. Fault tolerant design using single and multicriteria genetic algorithm optimization. Technical report, DTIC Document, 1995.

Michèle Sebag and Antoine Ducoulombier. Extending population-based incremental learning to continuous search spaces. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, PPSN V, pages 418–427, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65078-4.

Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent advances and historical development of vector optimization*, pages 222–232. Springer, 1987.

Donald C Shoup. Cruising for parking. *Transport Policy*, 13(6):479–486, 2006.

Donald C Shoup, American Planning Association, et al. *The high cost of free parking*, volume 206. Planners Press Chicago, 2005.

Anders JV Skriver. A classification of bicriterion shortest path (bsp) algorithms. *Asia Pacific Journal of Operational Research*, 17(2):199–212, 2000.

Anders JV Skriver and Kim Allan Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27(6):507–524, 2000.

Francois Soumis, Jacques A Ferland, and Jean-Marc Rousseau. A model for large-scale aircraft routing and scheduling problems. *Transportation Research Part B: Methodological*, 14(1):191–201, 1980.

Michael Z. Spivey and Warren B. Powell. Some fixed-point results for the dynamic assignment problem. *Annals OR*, 124(1-4):15–33, 2003.

Ralph E Steuer and Eng-Ung Choo. An interactive weighted tchebycheff procedure for multiple objective programming. *Mathematical programming*, 26(3):326–344, 1983.

RE Steuer. Multiple criteria optimization: theory, computation, and application. *Willey, New York*, 1986.

H. Ishibuchi T. Murata. Moga: multi-objective genetic algorithms. *Proceedings of 1995 IEEE International Conference on Evolutionary Computation. Perth, WA, Australia: IEEE*, 1995.

Dušan Teodorović and Panta Lučić. Intelligent parking systems. *European Journal of Operational Research*, 175(3):1666–1681, 2006.

Russell G Thompson and Anthony J Richardson. A parking search model. *Transportation Research Part A: Policy and Practice*, 32(3):159–170, 1998.

Vincent T'kindt and J-C Billaut. Multicriteria scheduling problems: a survey. *RAIRO-Operations Research*, 35(02):143–163, 2001.

Sadayuki Tsugawa. Issues and recent trends in vehicle safety communication systems. *IATSS research*, 29(1):7–15, 2005.

Chi Tung Tung and Kim Lin Chew. A bicriterion pareto-optimal path algorithm. *ASIA-PACIFIC J. OPER. RES.*, 5(2):166–172, 1988.

Chi Tung Tung and Kim Lin Chew. A multicriteria pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

Daniel Tuyttens, Jacques Teghem, Ph Fortemps, and K Van Nieuwenhuyze. Performance of the mosa method for the bicriteria assignment problem. *Journal of Heuristics*, 6(3): 295–310, 2000.

EL Ulungu and J Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2):83–104, 1994.

E.L. Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.

David A Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document, 1999a.

David A Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document, 1999b.

David A Van Veldhuizen and Gary B Lamont. Evolutionary computation and convergence to a pareto front. In *Late breaking papers at the genetic programming 1998 conference*, pages 221–228. Citeseer, 1998a.

David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Citeseer, 1998b.

David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm test suites. In *Proceedings of the 1999 ACM symposium on Applied computing*, pages 351–357. ACM, 1999.

MA Venkataramanan and Marc Bornstein. A decision support system for parking space assignment. *Mathematical and computer modelling*, 15(8):71–76, 1991.

Chankong Vira and Yacov Y Haimes. *Multiobjective decision making: theory and methodology*. Number 8. North-Holland, 1983.

M Visée, J Teghem, M Pirlot, and EL Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12(2):139–155, 1998.

Judith YT Wang, Matthias Ehrgott, and Anthony Chen. A bi-objective user equilibrium model of travel time reliability in a road network. *Transportation Research Part B: Methodological*, 2013.

Chi Xie and S Travis Waller. Parametric search and problem decomposition for approximating pareto-optimal paths. *Transportation Research Part B: Methodological*, 46(8): 1043–1067, 2012.

Seong-eun Yoo, Poh Kit Chong, Taehong Kim, Jonggu Kang, Daeyoung Kim, Changsub Shin, Kyungbok Sung, and Byungtae Jang. Pgs: Parking guidance system based on wireless sensor network. In *Wireless Pervasive Computing, 2008. ISWPC 2008. 3rd International Symposium on*, pages 218–222. IEEE, 2008.

Q. Zhang, J. Sun, E. Tsang, and J. Ford. Estimation of distribution algorithm with 2-opt local search for the quadratic assignment problem. In *Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithm*, pages 281–292. Springer-Verlag, 2006.

Yanan Zhao and Emmanuel G Collins Jr. Robust automatic parallel parking in tight spaces via fuzzy logic. *Robotics and Autonomous Systems*, 51(2):111–127, 2005.

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, 2003.