Hyper-heuristic cooperation based approach for bus driver scheduling

Shi Li

SPIM

Thèse de Doctorat

utbm université de technologie Belfort-Montbéliard

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE TECHNOLOGIE BELFORT-MONTBÉLIARD

Hyper-heuristic cooperation based approach for bus driver scheduling

Shi Li

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE TECHNOLOGIE BELFORT-MONTBÉLIARD

N° 2 1 1

THÈSE présentée par

SHI LI

pour obtenir le

Grade de Docteur de
l'Université de Technologie de Belfort-Montbéliard

Spécialité : **Informatique**

Hyper-heuristic cooperation based approach for bus driver scheduling

Soutenue publiquement le 18 Octobre 2013 devant le Jury composé de :

SYLVAIN PIECHOWIAK	Rapporteur	Professeur à Université de Valenciennes et du Hainaut-Cambrésis
IMED KACEM	Rapporteur	Professeur à l'Université de Lorraine-Metz
PARISA GHODOUS	Examinateur	Professeur à l'Université Claude Bernard-Lyon 1
DAVID MEIGNAN	Examinateur	Chercheur à l'Université d'Osnabrück (Allemagne)
ABDERRAFIÂA KOUKAM	Directeur de Thèse	Professeur à l'Université de Technologie de Belfort-Montbéliard
AMIR HAJJAM EL HASSANI	Co-Directeur de Thèse	Maître de conférences (HDR) à l'Université de Technologie de Belfort-Montbéliard

Acknowledgements

Completion of this doctoral thesis would not have been possible without the support and help of several people. First and foremost, I would like to thank my supervisor Professor Dr. Abderrafiâa KOUKAM. In fact, no words can adequately express my gratitude to him for his guidance, support and endless encouragement. He patiently explained many of the concepts, offered numerous helpful suggestions and guided me in the right direction. In addition, his support and help have been invaluable on not only an academic level but also a personal one. Until now, I still remember that he told me "as the Ph.D. time is precious but once in our life, we must put all our effort into it." These words always encourage me, especially in tough times for my Ph.D. work. In a word, it has been lucky enough to be one of his students. I would like also to express my sincere thanks to my co-supervisor Associate Professor Dr. Amir HAJJAM EL HASSANI. I am deeply indebted to him for all his insightful comments, inspiration and knowledge. He helped me every step throughout the doctoral process. He allowed me to follow my own research path and gave me the confidence to explore my research interests. However, the door to his office is always open whenever I have encountered any bumps.

For this thesis I would like to thank my thesis committee members: Dr. Sylvain PIECHOWIAK, Dr. Imed KACEM, Dr. Parisa GHODOUS and Dr. David MEIGNAN for agreeing to serve on my committee and for giving valuable comments, which improved the manuscript notably.

I am thankful to the multiagent and optimization research group for creating and maintaining an excellent academic environment. More specifically, I have to thank Jean-Charles CREPUT, from whom I have learned many things in academic writing. Additionally, I am deeply affected by his enthusiastic and strict attitude that he has for the research. A special thank goes also to Olivier LAMOTTE for providing Optymo data as a real-world case study. Undoubtedly this case study is contributed to the significant development of this thesis. I would like to give my thanks to Gillian BASSO, who worked in the same office and offered me so much help. I thank all other members of our research group including, but not limited to the following names: Nicolas GAUD, Stéphane GALLAND, Vincent HILAIRE, Fabrice LAURI and Cédric BOITTIN.

The road to my Ph.D. degree has been long and winding in some sense like marathon. I am greatly indebted to the many people who in some way contributed to the progress of my work. For this reason, I thank, both past and present, to a group of talented Chinese Ph.D. students Huide ZHOU, Jiawei ZHU, Xuguang HAO, Yishuai LIN, Zhongliang LI,

Weidong LEI, Yong FANG, Dongdong ZHAO, Nanfang YANG, Hongjian WANG, Zhixue ZHENG and Biao Yin for their kind help and friendship. Regrettably, but inevitably, there are lots of other friends I am indebted to. Please forgive me for not listing your names here one by one.

The time in the Ph.D. pursuit was enjoyable in large part due to many friends. I would like to thank my friends, especially Nourredine TABIA, Abderrahim CHARIETE, Kahina AIT ALI, Imad MATRAJI, Fayez Shakil AHMED, Athmane KEBAIRI, Mohamed HARMOUCHE and Julien MOREAU, who are always ready to help me with a smile whenever I needed them. On a personal note, my French level has been improved thanks to them. Otherwise, I would still a person who live in France but could not speak a complete French phrase. *Je ne vous remercierai jamais assez pour ce que vous avez fait pour moi.*

My research work was funded by the China Scholarship Council (CSC). Therefore, I would like also to express my gratitude to this financial support.

I am very grateful for my loving, supportive, encouraging, and patient parents Shuhua CHEN and Tianming LI. Thank you for all their love and encouragement. Without them, this thesis would never have been written. I dedicate this thesis to the memory of my grandfather Dengkun LI, whose role in my memory was immense. The last word of acknowledgment I have saved for my future wife Haige SUN, who has been with me all these years and shared the best and worst moments of my doctoral journey.

Contents

List of definitions

List of Tables

List of Figures

1 Context

All public transit providers around the world face a common problem: scheduling their drivers. Three main aspects should be considered for this problem: all the tasks must be completely assigned, all the duties must be feasible and the number of drivers must be minimal. The bus driver scheduling problem is without doubt a very complex problem. A great deal of research effort on this problem has been made since the 1960's. However, driver scheduling still remains one of the most challenging problems in the planning and scheduling process of public transports [Zhao, 2006]. Thus, the research in this area is still ongoing and new approaches have been constantly sought to solve this problem.

During the past years there is a phenomenal increase in problem sizes in the area of bus driver scheduling due to two factors: rapid expansion of bus lines in cities and explosion in the number of public transport passengers. Therefore, many companies do not require their bus driver scheduling problems solved to optimality or even close to optimality since they are more interested in good enough solutions in a reasonable time. Heuristics have played an important role in such a situation. Although we can find the various heuristic approaches to deal with bus driver scheduling problems in the literature, these algorithms are tailored to the particular companies that may differ in the objective function and in the duty constraints, such as union contract, company regulation, etc. Since the algorithm is dedicated, it is hard to adapt and to apply to other problems, even other instances. Even though metaheuristics were brought in to cope with this drawback, the employed metaheuristics, in most of the metaheuristic studies, fell within problem-dependent methodologies. Research on hyper-heuristics is an attempt to overcome such dependences in metaheuristics. In other words, hyper-heuristic research is motivated by the goal of raising the level of generality for solving a range of problems. In this thesis we focus our attention on a hyper-heuristic approach, which has the potential advantages over existing heuristics or metaheuristics in terms of the flexibility, the modularity and the robustness.

A classical problem when designing metaheuristics is the difficulty to achieve a balance between intensification and diversification. The use of organizational models encourages the design of metaheuristics by the identification of common components [Meignan et al., 2008]. Therefore, we are interested in using the organizational concepts to support the design of approaches within the context of hyper-heuristics in our work.

Like some similar problems, the bus driver scheduling problem can also be solved by mathematical programming, particularly in linear programming. Based on a literature review, column generation is one of the most successful approaches for Crew Scheduling Problems [Desrochers and Soumis, 1989, Ernst et al., 2004, Lübbecke and Desrosiers, 2005]. As we know, the use of mathematical methods can help to obtain optimal solutions, but the required computational complexity will result in exponential time when the problem size is large. In recent years, interest in combination exact and heuristic algorithms has risen considerably among researchers in combinatorial optimization. In our work, we

also attempt to explore the usefulness and potential of this research direction by seeking a heuristic approach in the context of column generation. It should be pointed out that this approach is still in its infancy and needs substantial further research.

2 Objectives and concerns of this work

The main concern of this work can be summarized as follows:

> Propose an approach to solve bus driver scheduling problems more effectively and efficiently.

Our objective can be decomposed in two subcomponents. First, we describe a pattern built upon organizational concepts in order to facilitate the design of cooperative hyper-heuristics. Second, we develop an efficient and adaptive approach to solve the driver scheduling problem by validating the proposed pattern.

2 .1 A multi-level hyper-heuristic pattern based on the Agent Metaheuristic Framework (AMF)

Since hyper-heuristics may have different performances during the search, it makes sense to see whether they can cooperate in some way so that they can exchange useful information to improve the capacity of exploration in the search space. However, the key challenge in cooperative search is the design of cooperation mechanisms and the determination of useful information to exchange between hyper-heuristics. With this in mind, an organizational model called multi-level hyper-heuristic pattern is proposed to facilitate the design of cooperative hyper-heuristics. In fact, this pattern is derived from the Agent Metaheuristics Framework (AMF) proposed by Meignan et al. [2008]. Specifically, Meignan et al. [2008] proposed the AMF for analyzing existing algorithms and encouraging the design of new metaheuristics. In this framework, a metaheuristic is defined as an organization composed of a set of roles which interact in order to find an optimal solution. From this point of view, an organizational model of metaheuristics can be used to describe both population-based metaheuristics and trajectory methods. Within the research covered in the hyper-heuristic literature, to our knowledge, there is scarce research work on the analysis of hyper-heuristics in this way. Therefore, we motivate and describe an organizational view of cooperative search in the context of hyper-heuristics. The resulting pattern aims at supporting the design of cooperative hyper-heuristics with the desired characteristics of flexibility, scalability and generality.

2 .2 TPCH: A multiagent hyper-heuristic approach

To validate the proposed pattern, we develop a multiagent hyper-heuristic approach called Two-phase Cooperative Hyper-heuristic Approach (TPCH). In this approach, we present how hyper-heuristics, referred as agents, can be organized on the metaphor of the coalition. More precisely, TPCH is a parallel computing algorithm with the purpose of accelerating and broadening the search. Parallelization scheme is implemented in a way such that the removal or addition of any agent would not perturb the global functioning of the system. Consequently, TPCH has been designed by considering a parallel hyper-heuristic approach with decentralized strategy, where several agents visit the search space independently. Moreover, the approach, on the one hand, does not rely upon any global information. On the other hand, every agent can communicate with others using information that it receives from other agents. To be precise, this communication can be done by exchanging information about solutions found and learning behaviors of search. Overall, the coalition is composed of several agents, which concurrently explore the search space and cooperate to improve their search abilities.

Agent, which combines a set of low level heuristics, looks more like an intelligent search since it applies some rules based on artificial intelligence principles. Specifically, an agent improves a candidate solution iteratively by selecting and applying a heuristic from the set of low level heuristics when solving a given problem. The agent uses an individual learning to adapt the heuristic selection in response to each low level heuristic performance by updating a weight matrix. At the same time, a cooperative learning is used to share the behaviors among the agents. The combination of individual learning and cooperative learning enables the agent to find the best of sequence of low level heuristics to apply during the search process.

Such an approach allows not only to speed up the convergence to the best solution, but also to apply consistently over diverse sets of problem instances without excessive efforts. To illustrate, the application of this approach is conducted on a variety of datasets including real world scheduling problems.

3 Structure of the thesis

After a brief overview of the concerns of this work we present the organization of this thesis. Following the previous analysis we have divided the thesis in two main parts. The structure of the thesis is illustrated in Fig. 1.

The **first part** (chapters 1 and 2) presents the state of the art concerning the problem of bus driver scheduling and hyper-heuristics. We first introduce the operational context in which the problem is solved and two formal models used in our work for bus driver scheduling. Then, we give an overview of related work on problem-solving methods including both exact and heuristic methods (chapter 1). Finally, we review the literature on

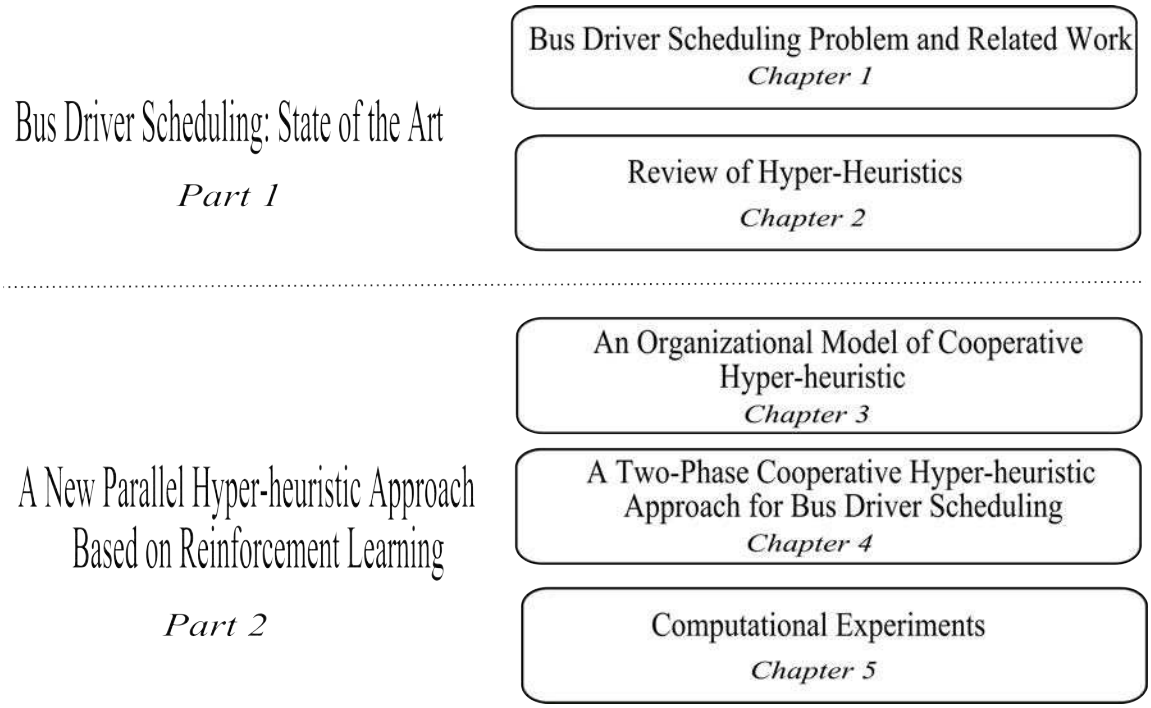

Figure 1: Structure of the thesis

hyper-heuristics (chapter 2).

The **second part** (chapters 3, 4 and 5) presents the two-phase cooperative hyper-heuristic approach (TPCH), which is a parallel hyper-heuristic approach based on reinforcement learning. Briefly speaking, we start by introducing the Agent Metaheuristic Framework (AMF). Based on this framework, we describe an organization as a pattern to design the approaches which aim at making use of the synergies among hyper-heuristics (chapter 3). The two-phase cooperative hyper-heuristic approach is built from this pattern. We give the details of this approach to the bus driver scheduling problem (chapter 4). Finally, we study the influence of the role of the main algorithmic components in the proposed approach, and applies the approach to a variety of artificial benchmark datasets and to real-world problem instances (chapter 5).

This thesis consists of five chapters, and is organized in the following way:

Chapter 1 describes the problem domain. This chapter firstly summarizes the background of bus driver scheduling problems. Then we present the problem formulations used by our proposed approaches. Finally, we summarize the optimization techniques that are applied to bus driver scheduling problems.

Chapter 2 overviews the hyper-heuristic literature presenting the ways in which the term "hyper-heuristic" has been interpreted and applied. As an emerging research direction, the field of hyper-heuristics is increasingly related to some other research in the literature. Thus, this chapter also discusses these current research trends.

Chapter 3 takes the first steps towards the design of cooperative hyper-heuristics. We

introduce the organizational concepts. Then, the relationship between organization theory and cooperative hyper-heuristics is studied. Finally, we present an organizational view of cooperative hyper-heuristics called multi-level hyper-heuristic pattern, which is based on the Agent Metaheuristic Framework (AMF) proposed by Meignan et al. [2009].

Chapter 4 presents our approach to solve the problem of bus driver scheduling. The principles and the major components are depicted in this chapter. Indeed, this approach is built from the pattern given in chapter 3. Therefore, we can consider our approach as a case study to present how concepts from organizational theory and multiagent system may contribute to the design of new cooperative search within the context of hyper-heuristics.

Chapter 5 continues the study of the proposed approach by testing the approach on different problem instances. The experimental results show that our approach outperforms some of heuristics to the bus driver scheduling problem of the literature.

PART I

Bus Driver Scheduling: State of the Art

CHAPTER 1

THE BUS DRIVER SCHEDULING PROBLEM

1 Introduction

Bus driver scheduling is a major planning problem arising from bus companies. Thus, research has been widely developed in recent years. It is motivated by the following three main reasons:

• Bus companies face an increasing cost pressure due to the expensive cost of drivers. As a consequence, scheduling drivers is important to make massive savings.

• Some traditional methods cannot meet future transportation needs because scheduling drivers is becoming complex due to increased larger problem sizes and complexity of labor rules.

• Developments in computer, in conjunction with new algorithms, which have advanced remarkably can help public bus operators to improve their existing tools.

As stated above, public bus operators are motivated to seek the most efficient bus driver schedules in their operational planning on the one hand. On the other hand, bus driver scheduling is considered as a type of crew scheduling problem well-known to be NP-hard [Kwan and Kwan, 2007, Leung, 2004]. Although a lot of work has been done on studying bus driver scheduling problem, research on solving this problem still continues, especially on exploring the possibility of obtaining efficiently near-optimal solutions.

The main objective of this chapter is to present the bus driver scheduling problem, to describe the formulations used in this thesis, and to give the overview of the different approaches applied in the literature. The chapter is organized as follows: Section 2 introduces the contexts under which the scheduling process is performed and presents some definitions that allow us to state this problem in a formal way. Section 3 describes in detail two formulations for the bus driver scheduling problem. At last, we review some related research works on the bus driver scheduling techniques in section 4 .

2 Problem description

2 .1 Public transport planning process

For several decades now, a variety of methods has been applied for solving problems in public transport bus services. However, bus companies are still faced with an important challenge to improve service quality and reduce operating costs. Addressed as a whole, the global problem of public bus service is very complex to solve because they involve passengers, buses and drivers that are subject to individual preferences and constraints. In order to fulfill this role, the transport operations are required to establish and accomplish a public transport planning process. In general, the public transport planning can be divided in different steps: Network Design, Service Frequency Setting, Timetabling, Vehicle Scheduling, Crew Scheduling (bus driver scheduling) and Crew Rostering. In Fig. 1.1, we illustrate the relationship between these steps in the public transport planning process. The subject of our work is related to the step of crew scheduling, however, the whole process will be briefly described below.

In network design, the bus network structure includes all information which concerns the areas covered by the transport service. The result of the network design should include a set of bus routes. A set of bus lines composes the transport service by providing the buses traveling on these routes. For each line, the frequency is determined by the passenger demand. After setting frequencies, the next step is to construct a timetable resulting in journeys characterized by a start and final location, and a start and end time. Then, the vehicle scheduling assigns vehicles to journeys resulting in a schedule for each vehicle. A schedule for a vehicle can be split into several vehicle blocks, where their lengths are determined by the total operating time of the bus. On such a block, a sequence of tasks can be defined, where each task needs to be assigned to one crew. The crew scheduling generates daily duties for drivers. The roster scheduling is a long term crew planning (e.g. half a year) comprising their days off and holidays. For a survey in the public transport planning see [Freling, 1997, Lourenco et al., 2001], as well as a series of books arising from the Computer-Aided Transit Scheduling conferences [Daduna et al., 1995, Desrochers and Rousseau, 1992, Hickman et al., 2008, Lo, 2009, Voss and Daduna, 2001, Wren, 1981].

2 .2 Bus driver scheduling

Our focus in this thesis is to deal with Crew Scheduling Problem also designated as Bus Driver Scheduling Problem (BDSP), which is widely considered to be one of the more challenging problems in public transport planning. It is an important part of the public transport planning from an economic point of view since it determines most of the wages paid to the drivers, which are a very large cost element that accounts for about 45% of the total operating cost [Meilton, 2001]. However, bus driver scheduling problems are well-

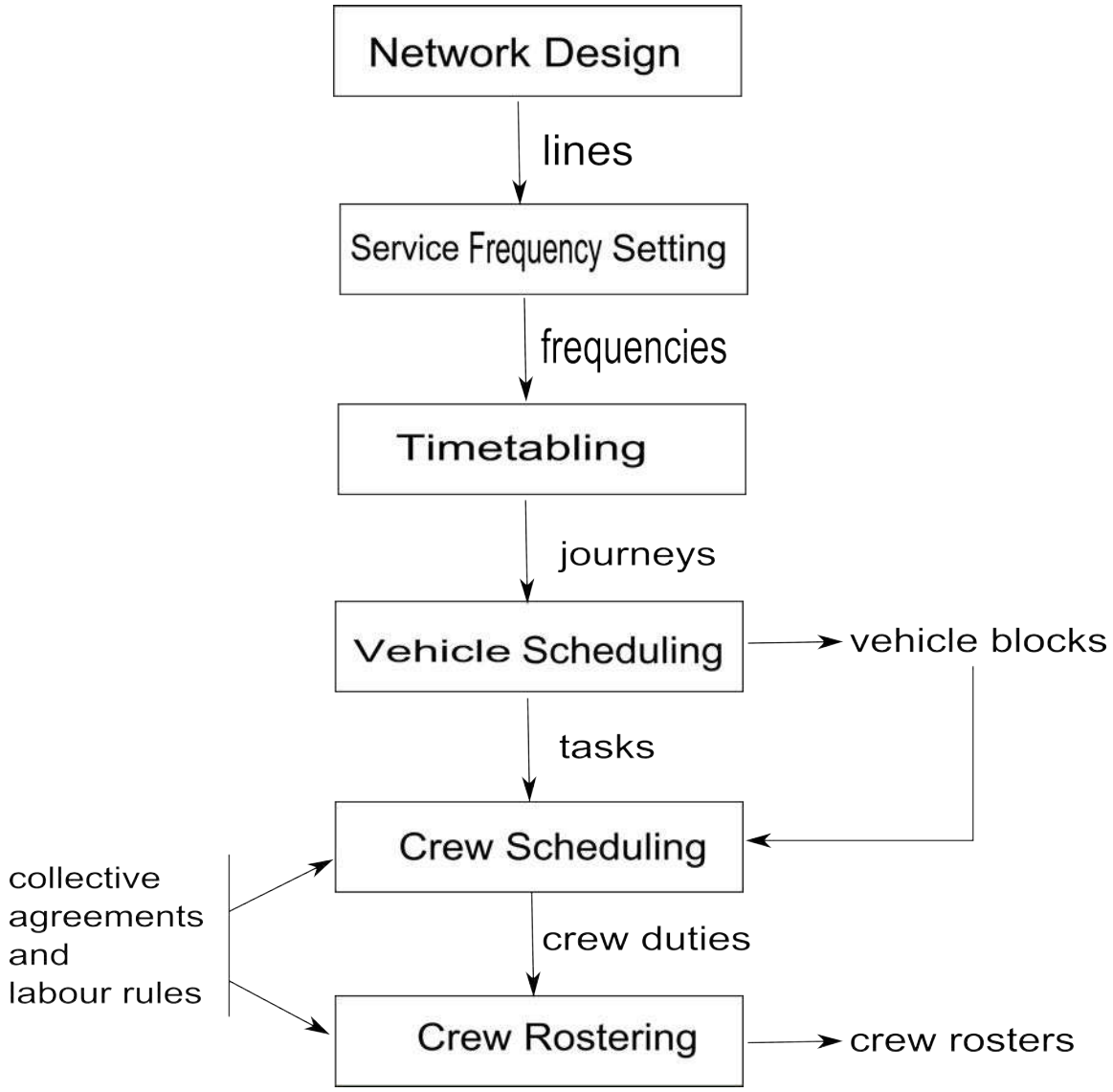

Figure 1.1: Public transport planning process

known NP-hard problems [Leung, 2004]. Meanwhile, since the constraints according to labor rules and requirements differ from country to country, even company to company, the evaluation criteria and objectives may differ as well. In this sense, it is also an extremely complex part of the transportation planning process.

In fact, there is a strong connection between the scheduling of vehicles and bus drivers. Before scheduling the drivers' tasks, the vehicle units normally have to be scheduled resulting in obtaining a set of vehicle block. A *vehicle block* is a sequence of journeys to be done by one vehicle from the time that it leaves the depot until it returns to the depot in one day. Table 1.1 displays an example of a block in a vehicle schedule. In Table 1.1, the main body describes a sequence of journeys operated by a vehicle, e.g. a journey from Valdoie at 06:05 to Paquis at 06:10. The header of this table denotes a stop name (indicated by *Stop*), the time when the vehicle arrives at a stop (indicated by *Time*) and whether the

Table 1.1: Example of a block in a vehicle schedule

Bus 1		
Stop	Time	Relief point
Valdoie	06:05	Yes
Paquis	06:10	No
Martinet	06:25	No
Briand	06:30	No
Madrid	06:35	Yes
Champ de Mars	06:40	No
Mieg	06:45	No
Saget	06:50	No
La Douce	06:55	Yes
Saget	07:00	No
Mieg	07:05	No
Champ de Mars	07:45	No
Madrid	07:50	Yes
Briand	07:55	No
Martinet	08:00	No
Paquis	08:15	No
Valdoie	08:20	Yes

stop is a relief point where one driver can replace another (indicated by *Relief point*). It should be noted that not all the stops are feasible locations for relieving drivers. In practice, relief points are typically at terminals and the appointed stops which are accessible. The bus driver scheduling problem involves partitioning the vehicle blocks into a set of legal driver duties that should reflect the operator's definition of efficiency [Lourenco et al., 2001]. For driver scheduling purposes, a block is usually represented in a graphical format [Parker and Smith, 1981] showing a sequence of bus stops while identifying the bus number (i.e. block number) and a set of relief points. Fig. 1.2 is a graphical representation of the bus block of Table 1.1. Note that only those relief points where drivers can change are marked. In this example, *Valdoie*, *La Douce* and *Madrid* are designated as relief points, which are denoted as V, D and M respectively in Fig. 1.2. The interval between two consecutive relief points on a block must be worked by a single driver since there are no other opportunities to change divers except at relief points during this period. Hence, this interval, from the driver scheduling point of view, is considered as a task. From this figure, resulting four tasks, such as task 1 with working time from $06{:}05$ at *Valdoie* to $06{:}35$ at *Madrid*, can be identified. It should be pointed out that the problem size can be reduced in such a way that some relief opportunities are omitted.

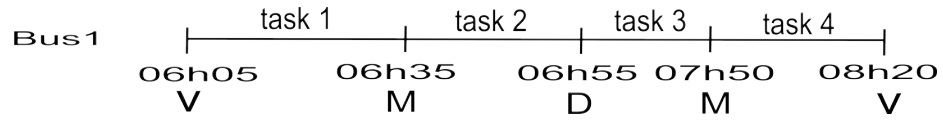

Figure 1.2: Example of a block with relief points

2 .3 Definition of the terms

As explained above, it is necessary to gather a set of vehicle blocks showing the driving work to be covered. An example of two bus blocks with a duty is illustrated in Fig. 1.3. From this example, we can see that drivers can only be relieved at some designated places called *relief locations*, which are represented by letter codes, such as A, B and C in the figure. The times when a vehicle is at the relief locations are marked on the horizontal timescale, they are known as *relief times*. Each pair of *relief location* and *relief time* is a *relief opportunity*. The work between two consecutive relief opportunities on a same vehicle is called a *piece of work (or task, or trip)* for the driver. The work of a driver in a day is known as a *duty (or shift)*. Note that not all relief opportunities will be used to relieve drivers, and therefore a driver may cover a number of consecutive pieces of work in a same vehicle, called a *spell*. It can be seen obviously that an example of a duty built among two vehicle blocks is given in this figure. The duty is composed of two spells from the two vehicle blocks.

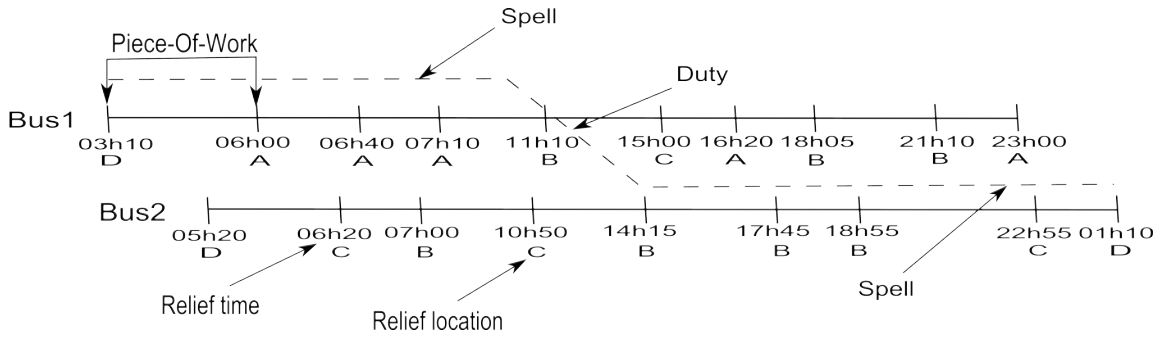

Figure 1.3: Relationship between vehicle blocks and pieces of work

Duty types are classified according to when the drivers start their work. Thus, the legal duties can be divided into the following four types:

Early Duty signs on early in the morning taking the buses out of garage before the morning peak. The working time starts between 05h00 and 07h00.

Day Duty begins in the morning and ends in the afternoon. The working time starts between 07h00 and 17h00.

Late Duty begins in the afternoon and ends in the night. The working time starts between 17h00 and 20h00.

Night Duty works in the late evening buses returning the buses to the garage. The working time starts between 20h00 and 24h00.

Clearly, the above is one of the possible classifications. Therefore, it evidently exists other classifications in practice.

3 Mathematical formulations

A great variety of problems from practice can be formulated as a set covering problem including scheduling, manufacturing, service planning, information retrieval, etc [Lan et al., 2007]. The set covering problem may be defined as follows. There are many elements contained in several sets (elements may be contained in more than one set), and the goal is to find the smallest number of sets so that every element is represented in the selected sets. The set partitioning is a special case, when overlapping is not allowed, i.e. one element can not be covered by several sets.

In the formulation adopted in the second part of this thesis, we consider a set partitioning formulation of the bus driver scheduling problem. It should be noted that the bus driver scheduling problem has been formulated as a set covering when we study a column generation heuristic.

3 .1 Set partitioning and set covering models

Let $S = \{1, 2, \ldots, n\}$ be the set of legal duties and $M = \{1, 2, \ldots, m\}$ be the set of pieces of work to be covered. We can define the problem as the construction of a matrix (a_{ij}), where duties appear in columns and pieces of work in lines. Each element $a_{ij} \in \{0, 1\}$, $i \in M$, $j \in S$, of the matrix is such that $a_{ij} = 1$ if duty j covers pieces of work i, $a_{ij} = 0$ otherwise. Moreover, duty j has to satisfy constraints imposed by labour rules and company requirements. We will detail the constraints below. The set partitioning can be formulated as follows:

$$\min W_1 \sum_{j=1}^{n} c_j x_j + W_2 \sum_{j=1}^{n} x_j \tag{1.1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = 1, \quad i = 1, 2, \ldots, m, \tag{1.2}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n, \tag{1.3}$$

where c_j is the cost of duty j, x_j is equal to 1 if duty j is used in the solution and 0 otherwise, and W_1 and W_2 are weight constants. Constraint (1.2) means that there is only one driver in each vehicle at any time. Note that the term $\sum_{j=1}^{n} c_j x_j$ is the total cost, whereas $\sum_{j=1}^{n} x_j$ is the number of duties and hence of drivers.

By changing the constraint (1.2) into the following form, it becomes a set covering problem, where pieces of work may be covered more than once in a schedule.

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1, 2, \ldots, m, \tag{1.4}$$

Such a formulation is interesting because set covering concerns more duties than set partitioning and can be solved easily.

3 .2 Constraints of legal duty

We now detail precisely what is a legal duty, and define the cost function. Let $ST(t)$ and $FT(t)$ respectively be the starting and ending time of piece of work $t \in M$. Let $DS(t)$ and $AS(t)$ respectively correspond to the depart and arrival station (relief location) of piece of work $t \in M$. To each duty $j \in S$ is associated an ordered set of pieces of work $D_j = \{t_{j1}, t_{j2}, \ldots, t_{jk_j}\}, t_{jl} \in M$, that defines the (a_{ij}) matrix coefficients of the set partitioning problem. Let $WTD_j = FT(t_{jk_j}) - ST(t_{j1})$ denote the total working time of duty j. Furthermore, let MWT denote the maximum working time and NWT denote the normal working time that are allowed by labour rules and specific company requirements. Note that t_{jl} and $t_{j(l+1)}$ are two consecutive pieces of work in duty j. A duty $j \in S$ is a legal duty if it satisfies the following constraints:

$$WTD_j \leq MWT, \quad j = 1, 2, \ldots, n, \tag{1.5}$$

$$FT(t_{jl}) \leq ST(t_{j(l+1)}), \quad j = 1, 2, \ldots, n, \tag{1.6}$$

$$AS(t_{jl}) = DS(t_{j(l+1)}), \quad j = 1, 2, \ldots, n. \tag{1.7}$$

Constraint (1.5) enforces that the working time must not exceed the maximum working time. Constraint (1.6) assures that the ending time of piece of work t_{jl} must not exceed the starting time of the next piece of work $l + 1$ in duty j. Constraint (1.7) guarantees that the arrival station of a piece of work is the same as the depart station of the next piece of work in duty j.

We consider the cost c_j associated with a duty j as an aggregative function given by the following formula:

$$c_j = OV_j + IT_j, \quad j = 1, 2, \ldots, n, \tag{1.8}$$

with

$$OV_j = \max(0, WTD_j - NWT), \quad j = 1, 2, \ldots, n, \tag{1.9}$$

$$IT_j = \max(0, NWT - WTD_j) + \ldots$$
$$+ \sum_{l=1}^{k_j-1} \max(0, ST(t_{j(l+1)}) - FT(t_{jl})), \quad j = 1, 2, \ldots, n. \tag{1.10}$$

The OV_j term defines the extra-time over the normal time of work established by law for driver j, called over-time. The IT_j term is the idle time of driver j, that is, the time during which a driver has no task to do.

4 Related work

The approaches for the bus driver scheduling problem can be roughly classified into two groups.

The first one is the Generate and Select (GaS) approach [Fores, 1996, Fores et al., 2001, Kwan and Kwan, 2007, Leung, 2004]. GaS consists of two main steps. In generating step, GaS builds a suitable large number of feasible duties. These duties satisfy all the constraints. The aim of the selection step is to select a subset of duties to cover all vehicle work. Usually, mathematical programming methods are employed in this step. Some meta-heuristic methods for the selection process have been proposed in recent years (e.g. [Dias et al., 2002, Li and Kwan, 2003]). One of the advantages when using this type of approach is that the duty generation module is separated from the duty selection module. This makes the approach adaptable to different situations since the separation between duty generation process and duties selection allow to adjust only the first module for each transportation company and, therefore, is very convenient for implementation reasons [Lourenco et al., 2001]. Unfortunately, the number of candidate duties is usually enormous even in the case of a small problem, which precludes the duty selection module from finding an optimal solution in reasonable time.

The other one is the constructive approach, which constructs an initial solution and then tries to iteratively improve it [Aickelin et al., 2009, De Leone et al., 2011]. Obviously, there are several motivations for applying this approach to the bus driver scheduling problem. First, the constructive approach works directly with integer solutions but GaS usually needs to solve the linear programming relaxation. Second, the enormous size of problems leads us to apply more efficient approaches. The constructive approach may obtain a good approximation of the set of solutions in practical time due to the computational efficiency. Third, the constructive approach can provide flexibility in handling variations of the model under special constraints originated by bus companies. Fourth, the constructive approach is relatively simple to implement and allows specific information to be exploited.

Strictly, the bus driver scheduling techniques can be divided into three groups: Heuristic

methods, Mathematical programming methods and Metaheuristic methods. In the following subsections, we give a brief literature review of about these three main approaches for solving the bus driver scheduling problem.

4 .1 Early heuristic methods

Heuristic methods were used predominantly in driver scheduling between the early 1960's and the late 1970's [Manington and Wren, 1975, Parker and Smith, 1981]. The early heuristics were useful in some applications. Indeed, this is because computers were not powerful enough to run the mathematical solvers in that period. Moreover, the early heuristics were able to meet the specific requirements for public transit companies since they were customized for these individual companies. A useful review relevant to these early approaches has been given by Wren and Rousseau [1995]. We summarize some implementations of the system based on heuristic methods in this area as follows.

RUCUS (Run Cutting and Scheduling) was developed in the later 1960's [Wren and Rousseau, 1995], which was installed in a number of bus companies in the USA and Canada. The system firstly constructs a good initial schedule, while a series of optimisation programs refine it.

TRACS (Techniques for Running Automatic Crew Scheduling) was developed at the University of Leeds from 1970 [Kwan et al., 1996]. The system had many similarities with RUCUS, which firstly construct an initial solution and then heuristically refined it.

HOT (Hamburg Optimisation Techniques) was developed and used by the schedulers at the Hamburger Hochbahn AC since the 1970's [Daduna and Mojsilovic, 1988]. The driver scheduling process is basically heuristic and may need some extensions to adapt to the needs of new users.

COMPACS (COMPuter Assisted Crew Scheduling) was an interactive driver scheduling system which is described by Wren and Chamberlain [1988]. The system first estimates the number of drivers needed to cover the vehicle work and then guides the decision maker to built up the schedule interactively using the estimated number of drivers.

4 .2 Mathematical programming methods

Although the heuristic systems were successful on some applications, these approaches were limited in solving the bus driver scheduling problems because a large amount of manual intervention was needed. In the later 1970's, research in driver scheduling methods has stepped into the period of mathematical programming approaches. The bus driver scheduling problem can be commonly modeled as a set covering or set partitioning integer linear program (ILP) [Shepardson, 1981]. Therefore, many linear programming and integer programming methods have been proposed [Mitra and Welsh, 1981, Ryan and Foster, 1981] that can address such models. We review three systems IMPACS, TRACS II and HASTUS

as follows.

IMPACS (Integer Mathematical Programming for Automatic Crew Scheduling) is a driver scheduling system developed by the University of Leeds in the later 1970's [Wren and Smith, 1988], which was used to plan transport serivices in the city of London (England) in 1984 and in Greater Manchester Buses in 1985. IMPACS is based on a set covering model. Firstly, a large number of possible duties with associated costs are generated, a subset is then selected to cover all the pieces of work at minimum cost. Meanwhile, it also provides a decomposition module for solving large problem since the number of variables and constraints are too large to be handled by computers in some cases.

TRACS II (Techniques for Running Automatic Crew Scheduling, Mark II) has been developed since 1994 specifically to satisfy the needs of rail and bus driver schedules [Kwan et al., 1996], which has been successfully installed in several transport companies (including First Group, the largest bus company in the UK). In fact, this system is a new generation of IMPACS. Therefore, it follows almost the same approach as IMPACS, but the components have been considerably redesigned to cope with the complexity of rail and bus operations and to incorporate new algorithmic advances.

HASTUS has been developed originally in 1974 by the University of Montreal's Center for Research on Transportation, and in collaboration with GIRO company [Blais and Rousseau, 1988], which has been widely used throughout a lot of cities in the world, such as Montreal, New York and Nantes (France). The system is an integrated and modular software solution for transit scheduling, operations, and passenger relations. A crew scheduling method called Crew-opt is one part of the system. The Crew-opt approach solves the driver scheduling problem formulating a set covering (or set partitioning) model in order to incorporate a column generation technique. The approach used involves several steps. It first generates a set of feasible duties. A subset of duties is then selected and, based on this subset, a linear programming solution is obtained by relaxing the integrality of the duty variables. After an LP solution is obtained, more feasible duties with negative reduced costs can be created to add into LP in order to improve the current solution. The problem of generating new feasible duties is formulated as a shortest path problem with constraints. The process of finding the relaxed LP and the generation of shifts with negative reduced costs is repeated until the LP optimum is reached. Then the process enters a branch and bound phase which also uses a column generation method to solve the ILP at each node of the branch and bound tree.

4 .3 Metaheuristic methods

Mathematical programming can only solve small problems because of the combinatorial nature of the scheduling constraints [Zhao, 2006] which make the problem NP-hard. Namely, practical solutions would be hard to find in short time. More recently, metaheuristic methods have been widely used for efficiently seeking near-optimal solutions to NP-hard prob-

lems. There are many metaheuristic approaches applied to the bus driver scheduling problem, such as Tabu search and Genetic algorithms [Aickelin et al., 2009, Cavique et al., 1999, De Leone et al., 2011, Dias et al., 2002, Li, 2005, Shen and Kwan, 2001].

Cavique et al. [1999] presented a Tabu search approach for the crew scheduling problem. Starting with an initial solution constructed by a traditional run-cutting approach, two alternative improvement algorithms, which are embedded in a tabu search framework, are used to reduce the number of duties in the initial solution. These two heuristic algorithms for a real crew scheduling problem were presented. However, these algorithms only construct one or two spell duties. The complexity will increase dramatically when duties with three or more spells are allowed.

Dias et al. [2002] applied a genetic algorithm to the bus driver scheduling problem. The application of genetic algorithms extends the traditional approach of Set Covering/Set Partitioning formulations, using a new coding scheme in order to incorporate the user's knowledge in a quite natural way. The performance of this algorithm was evaluated with standard test airline crew scheduling problems, and with real problems forming several medium-size Portuguese urban bus companies.

Li [2005] used a novel evolutionary approach which is called a self-adjusting approach for driver scheduling. It incorporates the idea of fuzzy evaluation into a self-adjusting process, combining the features of iterative improvement and consecutive perturbation, to explore solution space effectively and obtain superior schedules. Experiments with benchmark tests using data from the transportation industry demonstrate the success of the proposed approach in solving large size problems.

Aickelin et al. [2009] introduced a new technique called Evolutionary Squeaky Wheel Optimization to solve driver scheduling problems by using the original idea of Squeaky Wheel Optimization and incorporating two additional steps (Selection and Mutation) for added evolution. The experiments have demonstrated that this approach performs very competitively on two different domains of personnel scheduling: bus and rail driver scheduling and hospital nurse scheduling.

The latest research which addresses the problem of bus driver scheduling can be found in De Leone et al. [2011]. A Greedy Randomized Adaptive Search Procedure (GRASP) has been proposed and numerical results carried out on a set of instances show the effectiveness of the designed metaheuristic approach.

5 Summary

In this chapter we first introduced the public transport planning process. This planning process helps trace the information flow into and out of the bus driver scheduling part. Moreover, this knowledge may be useful because sometimes the key to a better schedule might consider a preceding step in the planning process.

This chapter also gives the description of mathematical models. The driver scheduling problem is generally formulated as a set covering/partitioning model. The set partitioning model is best suited for the bus driver scheduling problem while the set covering model allows the over-covering of tasks. This over-covering is not attractive, even not acceptable especially when assigning one driver is cheaper than assigning several drivers to a task. The main advantage of a set covering model over the set partitioning model is its flexibility which allows more rapid computation of a feasible solution. Additionally, solving the set covering model may produce a solution that contains very little over-covering in some cases.

Finally, this chapter reviews the driver scheduling approaches, which can be mainly divided into three groups: Heuristic methods, Mathematical programming methods and Metaheuristic methods. Early computerized methods for driver scheduling were purely heuristic and often needed large amounts of manual intervention. Moreover, most of early heuristic methods are based on the consecutive approaches which relied on good initial schedules constructed based on human schedules' knowledge. Therefore, early heuristics were limited in solving the bus driver scheduling problem. Afterwards, mathematical programming methods started to be used. In practice, there are several projects that have been developed to design planning systems using these methods, such as HASTUS. Some of these successful systems have been used by transportation companies in several countries and have through long development and experience working to fit with these companies' requirements. It should be noted that column generation algorithms have been widely applied to tackle the driver scheduling problem in these methods. Although a lot of research effort has been directed towards the development of methods, the driver scheduling problem is still open. Firstly, most of the companies need fast methods to obtain good enough solutions that can help the decision marker timely. Secondly, flexibility should be improved since there are some aspects of scheduling that are hard to incorporate in a linear programming model. Thirdly, the present mathematical approaches are hard to explain to schedulers. Sometimes, these systems must be manipulated to produce driver schedulers for different bus schedules. Such manipulation is frustrating and perhaps obscure to schedulers who have no knowledge of mathematical programming. Hence, all of these make us to explore other areas where improvement can be made. The balance between quality of solutions and computational time leads to the use of metaheuristic approaches, particularly considering the very large size of real problems. Thus far, there are many metaheuristic approaches applied to the bus driver scheduling problem.

It is obvious that driver scheduling has provided interesting and challenging problems to researchers. Research on this problem is still going on with the aim of developing new solution approaches or improving existing ones that will allow to solve larger instances and to address additional complexities.

CHAPTER 2

REVIEW OF HYPER-HEURISTICS

1 Introduction

The term hyper-heuristic was first used in 1997 to describe a protocol that combines several artificial intelligence methods in the context of automated theorem proving [Burke et al., 2010a]. In the context of combinatorial optimization, however, the term was independently used in 2000 to describe *heuristic to choose heuristics* [Ross, 2005]. Unlike metaheuristics which search in the space of solutions, hyper-heuristics search a space of heuristics. In this sense, they differ from most application of metaheuristics.

Several hyper-heuristic approaches have been proposed in the literature. Over the past decade, hyper-heuristics have been successfully investigated for a number of optimization problems (e.g. Burke et al. [2003b, 2007], Cowling et al. [2001a], Misir et al. [2010], Ochoa et al. [2009a], Ouelhadj and Petrovic [2008]). The underlying principle in using a hyper-heuristic approach is that different heuristics have different strengths and weaknesses and it makes sense to try to combine them (the heuristics) in an intelligent manner so that the strengths of one heuristic can compensate for the weaknesses of another [Burke et al., 2003a].

In this chapter we will discuss some research works related to hyper-heuristics. Section 2 firstly gives a brief history of hyper-heuristics. This is followed by introducing the fundamental difference between metaheuristics and hyper-heuristics in section 3 . Finally, a classification of hyper-heuristic approaches is summarized in section 4 .

2 A brief history

Although the term hyper-heuristic has been coined relatively recently, the ideas behind hyper-heuristics are not new. They can be traced back to the early 1960s, Fisher and Thompson [1963] proposed a method of combining scheduling rules using "probabilistic learning". In their far-sighted work, they concluded *(1) an unbiased random combination of scheduling rules is better than any of them taken separately; (2) learning is possible*. This is especially pioneer and worthy since at that time, computational search methodologies were far from mature, not even the idea of metaheuristics existed and only relatively

unsophisticated local search techniques were available.

In 1973, self-adaptation was originally introduced by Rechenberg [1971] for evolution strategies (ES), later developed by Rechenberg and Schwefel [1974] and Fogel et al. [1991]. This notion similar to hyper-heuristic means that some parameters are varied during a run in a specific manner: the parameters are included in the chromosomes and co-evolve with the solutions. These approaches related to the idea of searching over a space of possible algorithm which were proposed early in the history of evolutionary algorithms can also be considered as antecedents of hyper-heuristics.

Afterwards, more and more researchers have perceived that employing heuristic methods to solve intractable optimization problems like scheduling often suffers from narrowness in the range of problems to which they can be effectively applied. In 1993, a hill-climbing algorithm operates on a search space of control strategies for satellite communication is proposed in Gratch and Chien [1993]. Notice that the term "hyper-heuristic" was not still use in that time but this adaptive heuristic has been already close to the concept of hyper-heuristic. Namely, the selection of an heuristic strategy is effected until some information can be obtained with respect to which strategy is expected to perform most effectively in solving a problem instance or class of instances.

In fact, the quest for robust heuristics that are able to solve more than one problem was always ongoing. In 1995, a system called TEACHER was designed for learning and improving Heuristic Methods (HM) used in problem solving [Ieumwananonthachai and Wah, 1995]. This system employed a genetic-based machine learning approach, and was successfully applied to a whole range of different problem domains such as process mapping, load balancing on a network of workstations, routing and testing, etc. Then in 1996, Minton [1996] presents his work, called Multi-tac, which is the other learning system that synthesizes heuristic constraint satisfaction programs. Multi-tac takes a library of generic algorithms and heuristics and specializes them for a particular application.

Before the first time "hyper-heuristic" term used, there is actually another approach, called "Squeaky Wheel" Optimization (SWO), considered also as an antecedent to hyper-heuristic that it finds solutions quickly by operating on two search spaces: the traditional solution space and a new priority space. In 1998, Squeaky Wheel optimization method is introduced by Joslin and Clements [1999]. This is a search technique for solving a wide range of optimization problems. In SWO, a greedy algorithm is used to construct, a solution which is then analyzed to find the trouble spots, i.e., those elements, which, if improved, are likely to improve the objective function score. The results of the analysis are used to generate new priorities that determine the order in which the greedy algorithm constructs the next solution. This Construct/Analyze/Prioritize cycle continues until some limits are reached, or an acceptable solution is found. This work is developed by Aickelin et al. [2009]. Evolutionary SWO (ESWO), which is a recent extension to SWO, is designed to improve the intensification by keeping the good components of solutions and only using

SWO to reconstruct other poorer components of the solution.

In 1997, the term hyper-heuristic was first time used in Denzinger et al. [1997]. They used it to describe a protocol that chooses and combines several AI methods. Three years later, Cowling and Soubeiga [2000] used it independently to describe the idea of "heuristics to choose heuristics". This definition has been more clear to describe in Burke et al. [2003a] as a heuristic selection process used to choose heuristics, which have been generalized to designate a search method or learning mechanism for selecting or generating heuristics to solve hard computational search problems [Burke et al., 2010a].

More recently, a survey of hyper-heuristics can be found in Burke et al. [2013]. They describe some relevant intellectual roots and early approaches developed before 2000, and define the five following types of early approaches: *Automated heuristic sequencing, Automated planning systems, Automated parameters control in evolutionary algorithms, Automated learning of heuristic methods,* and *"Squeaky Wheel" Optimization*.

Today, hyper-heuristic has emerged as effective search techniques that have been applied to various problem domains, such as production scheduling (Ochoa et al. [2009b], Vázquez-Rodríguez and Petrovic [2010]), the timetabling problem (Burke et al. [2003b, 2007], Ochoa et al. [2009a]), the bin packing problem (López-Camacho et al. [2011], Sim et al. [2012]), the vehicle routing problem (Garrido et al. [2009], Garrido and Riff [2010]), etc.

3 Metaheuristics vs Hyper-heuristics

In this section we will see the difference between heuristics and hyper-heuristics. Over the last few decades, great efforts on a new kind of approximate algorithm which basically tries to combine basic heuristic methods aimed at effectively and efficiently exploring a search space. This kind of algorithms are nowadays commonly called *metaheuristics*. The term metaheuristic derives from two composition of two Greek words, which are explicated as follows. *Heuriskein (ancient Greek)* means "to find out, discover", while the prefix *meta* means "beyond, in an upper level".

Up to now various metaheuristics have been proposed in the literature, such as tabu search [Glover and Laguna, 1998], simulated annealing [Kirkpatrick et al., 1983], and evolution computation [Calégari et al., 1999]. Surveys and current research on metaheuristics can be found in Blum and Roli [2003] and Milano and Roli [2004]. In spite of the fact that metaheuristic methods have been successfully applied to many areas in recent years, we cannot ignore that they are often designed specifically and derived from prior experiences with the particular problem domains. Once the problem is changed (even slightly), the performance of the already developed specific-tailored metaheuristic may decrease dramatically for a new problem. Significant parameter tuning may also be necessary for the purpose of adapting the algorithms to the new problem or a new problem instance. The "No

Free Lunch" theorem [Wolpert and Macready, 1997] states that there is no one algorithm that is superior to any other algorithm across all classes of problems. If an algorithm outperforms other algorithms on a specific class of problems, there must be another class of problems for which this algorithm is worse than the others. This drawback of metaheuristics has motivated research to design algorithm which can be applied in many different situations.

The methodology of hyper-heuristics is motivated by the goal of increasing the level of generality of metaheuristics. The idea behind one type of hyper-heuristic is that better algorithmic performance could be achieved by the combination of many different heuristics, each with different relative performances. Namely, some heuristics may have the "individual flaws" in certain scenarios where other heuristics may perform better. Cowling et al. [2001a] describe it as managing the choice of which lower-level heuristic method should be applied at any given time, depending on the characteristics of the region of the solution space currently under exploration. Thus, it is a process which, when given a particular problem instance, manages the selection of problem-specific heuristics to apply until a stopping condition is met. Note that although low level heuristics could be metaheuristics, they are usually simple and easily implemented heuristics. The hyper-heuristics aim to tackle not only a specific problem or problem instance but a batch of problems. Furthermore, hyper-heuristics aim to develop algorithms that are more generally applicable rather than challenge the "No Free Lunch Theorem".

4 A classification of hyper-heuristic approaches

Despite being a rather young area of research, the methodology of hyper-heuristics has already involved a wide range of different approaches and techniques. Several works have made efforts towards providing classification schemes for hyper-heuristics.

In Soubeiga [2003], hyper-heuristics are classified into two groups: with learning and without learning. Hyper-heuristics in the first group include approaches which use several heuristics or neighborhood structures at each decision point, but select the heuristics to call according to a predetermined sequence. In the later group, the hyper-heuristics is equipped with a learning mechanism which dynamically change the preference of each heuristic based on their historical performance.

In Bai [2005], hyper-heuristics are classified into two types of methodologies: constructive and local search. Constructive hyper-heuristics build a solution incrementally by adaptively selecting heuristics, from a pool of constructive heuristics, at different stages of the constructive process. While local search hyper-heuristics start from a complete initial solution and iteratively select, from a pool of neighborhood structures, appropriate heuristics to lead the search in a promising direction.

In Cotta et al. [2008], hyper-heuristics are classified into four categories: (1) hyper-

heuristics based on the random choice of low level heuristics, (2) greedy and peckish hyper-heuristics, which requires preliminary evaluation of all or a subset of the hyper-heuristics in order to select the best performing one, (3) metaheuristics based hyper-heuristics, and (4) hyper-heuristics employing learning mechanisms to manage low level heuristics.

A current state-of-the-art classification is given by Burke et al. [2010a], where hyper-heuristics are classified according to two criteria: the nature of the heuristic search space and the source of feedback during learning. This classification is illustrated in Fig. 2.1. According to the first criterion, hyper-heuristics can be divided into two distinct groups: heuristic selection and heuristic generation. In the first group, a set of pre-existing heuristics is provided to select for solving the target problem. The task, in these hyper-heuristics, is to find a "best" sequence of applications of these heuristics for solving the problem. For the second group, the low level corresponds to a set of basic components of heuristics. In this group of hyper-heuristics, the process requires to evolve new heuristics by making use of these components for solving the target problem. The second criterion considers the source of feedback used by the hyper-heuristic. Using this criterion, we can distinguish three different types of hyper-heuristics due to the fact that they use online learning, offline learning and no learning. In online learning hyper-heuristics, a mechanism is used to modify the search strategy while the algorithm is solving an instance of a problem. In offline learning hyper-heuristics, the search strategy is defined in a way that trains a set of instances before solving problem instances. In addition to the two type of learning hyper-heuristics just discussed, no-learning hyper-heuristics can be considered as a third type, referring to those that do not use any feedback from the search process.

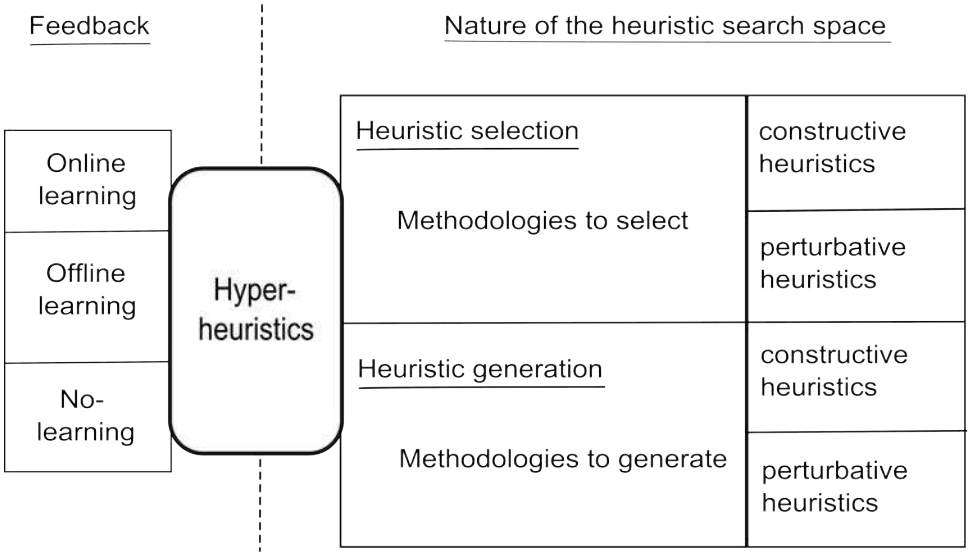

Figure 2.1: A classification of hyper-heuristic approaches [Burke et al., 2010a]

With respect to the first criterion in Burke et al. [2010a], the classifications can be further refined according to whether the hyper-heuristic controls construction or perturbation low level heuristics (see in Fig. 2.1). A hyper-heuristic that controls construction low level

heuristics builds a solution incrementally. It starts with an empty solution, and then selects the most suitable construction heuristics to gradually build a complete solution. A hyper-heuristic that controls perturbation low level heuristics starts with a complete initial solution and iteratively selects the appropriate perturbation heuristics to improve the current solution.

From the literature, the most fundamental hyper-heuristic categories can be clearly summarized as: generation hyper-heuristics and selection hyper-heuristics. One category creates new heuristics using basic components of heuristics. The motivation behind this category is to evolve new heuristics for solving the target problem. The other category selects existing heuristics. More precisely, this type of hyper-heuristics is provided with a set of heuristics. The task is thus to discover a good sequence of applications of these heuristics for efficiently solving the problem. In our work, we focus on selection hyper-heuristics. That is, the hyper-heuristic developed for the problem of bus driver scheduling in this thesis falls into the second category.

4 .1 Generation hyper-heuristics

The purpose of generation hyper-heuristics is to generate new heuristics from a set of heuristic components. Although there are a number of potential advantages of generation hyper-heuristics, this category of hyper-heuristics is less well studied in the literature. Indeed, many of the previous studies on the generation hyper-heuristics use genetic programming [Bader-El-Den and Poli, 2008, Burke et al., 2006, 2009b, Keller and Poli, 2008], which is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task [Koza, 1999]. As stated in Burke et al. [2010b], most examples of using genetic programming as a hyper-heuristic are offline in that a training set is used for generating a program that acts as a heuristic, which is thereafter used on unseen instances of the same problem. The motivation behind this work is to generate reusable heuristics. Namely, once a heuristic is evolved, it can be reused to any new problem instances. However, research on "disposable" hyper-heuristics has also been conducted [Keller and Poli, 2007]. In other words, these approaches are created for just one problem instance, rather than for unseen instances. In the following, we discuss two representative examples of generation hyper-heuristic using genetic programming.

Burke et al. [2006] proposed a genetic programming hyper-heuristic to automatically generate good heuristics for one dimensional bin packing. The genetic programming system chooses between a set of low level building blocks to evolve a heuristic. The heuristics generated by this system are functions consisting of arithmetic operators and properties of the problem. The best evolved heuristics are shown to be same to the human designed 'best-fit' heuristic on unseen problem instances.

In Keller and Poli [2007], the authors presented a linear genetic programming hyper-heuristic for the travelling salesman problem. The approach evolves programs which rep-

resent the repeated application of the simple local search operators. The programs are sentences of a language defined by a grammar. The system is first evolves sequence of 2-opt and 3-opt swap heuristics. Conditional and loop components are then added to the grammar, to increase the complexity of the evolved heuristics. Experimentation with benchmark instances shows that the results obtained are competitive with the best known results from literature.

4 .2 Selection hyper-heuristics

The majority of hyper-heuristics so far can be classified as selection hyper-heuristics. Such hyper-heuristics utilize a set of existing heuristics to improve an initial solution iteratively. More precisely, the most appropriate heuristic is chosen from the set of heuristics to employ at each step. At the high level, a hyper-heuristic interacts with the problem domain via these heuristics and gathers problem independent information such as the number of heuristics, the quality change in a candidate solution after applying a selected heuristic, or the success of a heuristic [Burke et al., 2012]. Obviously, an important issue here is how to decide the most appropriate heuristic at each step. In this respect, the selection hyper-heuristics use two major components to operate: heuristic selection and acceptance criterion [Burke et al., 2012, Özcan et al., 2008, 2010]. From this point of view, we present below the different heuristic selection mechanisms together with acceptance criteria that are employed by the existing hyper-heuristics.

In Cowling et al. [2001a], the authors experimented with a number of heuristic selection mechanisms *Simple Random*, *Random Permutation*, *Random Descent*, *Random Permutation Descent*, *Greedy* and *Choice Function*. We introduce briefly these selection mechanisms as follows. Simple Random chooses a low level heuristic randomly, applying it once, until some stopping criterion is met. Random Descent chooses a low level heuristic randomly and apply it until the candidate solution in hand is improved. Random Permutation generates a random initial permutation of the low level heuristics and applies each low-level heuristic once in the provided order. Random Permutation Descent is similar to the random permutation, but it applies the selected heuristic repeatedly as long as the solution improves. The Greedy method applies all low level heuristics to the current solution and chooses the one that generates the most improved solution. Choice Function is the most complex one. It analyzes both the performance of each low level heuristic and each successively applied pair of low level heuristics. In terms of acceptance criteria, the authors considered two simple methods: (i) all moves are accepted (AM), and (ii) only improving moves are accepted (OI). Experimental results show that the Choice Function that is combined with all moves accepted within a hyper-heuristic performed better than the rest.

Kendall and Mohamad [2004] proposed *Great Deluge* as the acceptance criterion and *Simple Random* as the heuristic selection to a mobile telecommunications network problem. Great deluge accepts all improving moves. However, non-improving moves are also

accepted if the objective value of the candidate solution is better or equal to a dynamically changing threshold value which depends on the current step and overall duration of the experiment [Kiraz et al., 2013].

Apart from these selection and acceptance mechanisms, several metaheuristic-based strategies for designing hyper-heuristics have been proposed in the literature, such as simulated annealing [Bai and Kendall, 2005], tabu search [Kendall and Hussin, 2005] and ant algorithm [Burke et al., 2005, Chen et al., 2007].

A simulated annealing based hyper-heuristic is experimented in Bai and Kendall [2005] for the shelf space allocation problem. The basic idea behind this approach is that simulated annealing is used to guide the selection and acceptance of the low level heuristics. In fact, greedy and choice function hyper-heuristics are also investigated in their work but the simulated annealing performed best. Specifically, for a maximisation problem, the pseudocode of the algorithm is given in Algorithm 1. The experimental results show that this approach produced high quality solutions in different problem situations.

Algorithm 1: Pseudocode for the simulated annealing based hyper-heuristic [Bai and Kendall, 2005]

```
1  Define an objective function f and a set of heuristics H;
2  Define a cooling schedule: starting temperature t_s > 0, a temperature reduction
   function φ and a number of iterations for each temperature nrep;
3  Select an initial solution s_0;
4  while the stopping criteria is not met do
5      Randomly select a heuristic h ∈ H;
6      iteration_count = 0;
7      for iteration_count = 0, ..., nrep do
8          iteration_count ++;
9          Applying h to s_0, get a new solution s_1;
10         δ = f(s_1) − f(s_0)
11         if δ ≥ 0 then
12             s_0 = s_1;
13         end
14         else
15             Generate a random x uniformly in the range (0, 1);
16             If x < exp(δ/t), then s_0 = s_1;
17         end
18     end
19     Set t = φ(t);
20 end
```

Kendall and Hussin [2005] proposed a Tabu Search heuristic selection method. A tabu list is used to monitor the performance of a collection of low level heuristics. Then, it makes tabu heuristics that have been applied too many times so that it allows to apply other heuristics which is not in the tabu list. This hyper-heuristic is outlined in Algorithm 2.

Experiments carried out on examination timetabling datasets from the literature show that this approach is able to produce good quality solutions.

Algorithm 2: Pseudocode for the tabu search hyper-heuristic approach[Kendall and Hussin, 2005]

```
1 Construct initial solution
2 while the terminating condition is not met do
3 |   Consider heuristics that are not tabu
4 |   Choose the best heuristic (with the best improvement)
5 |   Apply chosen heuristic and make the heuristic tabu
6 |   Update solution
7 |   Update the tabu status of heuristics in the tabu list
8 end
```

An ant algorithm hyper-heuristic is introduced by Burke et al. [2005] to solve the Project Presentation Scheduling Problem. Briefly, there is a network in which each vertex represents a low level heuristic. A number of hyper-heuristics, called ants, are located uniformly among the vertices and carry initial solutions. Each ant traverses particular edges and reach the next vertex. Once an ant arrives at a new vertex it applies the low level heuristic at that node. Using a similar idea, another ant algorithm hyper-heuristic is proposed by Chen et al. [2007] to solve the traveling tournament problem.

5 Summary

In this chapter we have reviewed hyper-heuristic techniques in the literature and identified reasons why the research is getting more and more popular. The aim of hyper-heuristic research is to provide the potential for increasing the level of generality of search methodologies, where it operates on a search space of heuristics rather than directly on a search space of problem solutions. From what we can see from existing papers on hyper-heuristics, the development of hyper-heuristics is going to play a major role in the field of optimization. Therefore, it is reasonable to believe that further research can be carried out in a more wide range of application areas.

So far, the most of research on hyper-heuristics has mainly focused on the development of independent hyper-heuristics where a single hyper-heuristic controls a set of low level heuristics to solve the problem. Clearly, there is much ground for further research within hyper-heuristics. We believe that research efforts on parallel execution and cooperation are worth devoting to hyper-heuristics. For one thing, parallel and distributed approaches can be used to provide more powerful and robust problem solving environments. For another, the use of cooperative approaches within a hyper-heuristic framework can be considered as novel ways to combine different independent hyper-heuristics. In the next chapter we

investigate on an agent-oriented approach. The goal of this study is to explore the cooperative search mechanisms within a hyper-heuristic framework. This promising research area opens up the possibility of having parallel execution of multiple hyper-heuristics that can cooperate by sharing the information, e.g., the best solutions.

PART II

A New Parallel Hyper-heuristic Approach Based on Reinforcement Learning

CHAPTER 3

AN ORGANIZATIONAL MODEL OF COOPERATIVE HYPER-HEURISTIC

1 Introduction

In Burke et al. [2009a], the authors stated that it is often still difficult to easily apply heuristic search methods to new problems, or even new instances of similar problems. These difficulties arise mainly from the significant number of parameter or algorithm choices involved when using these type of approaches, and the lack of guidance to proceed when selecting them. In such a situation, research in hyper-heuristics has gained attention. Recent studies on hyper-heuristics include not only the improvement hyper-heuristic performances facing different problems instances, but also the development of hyper-heuristic frameworks with the characteristics of simplicity and generality. Some of these previous studies can be found in several articles [Meignan et al., 2008, Ouelhadj and Petrovic, 2009, Özcan et al., 2009].

For many years, a technique has been developed under the general term of "artificial intelligence" (AI). Roughly speaking, AI refers to an intelligent system that simulates a certain form of human reasoning, knowledge, and expertise for solving one (or several) given problem(s). In some respects, the growth of this field has been spurred by the advances in distributed, coordinated and concurrent problem solving. Hence, Distributed Artificial intelligence (DAI) underlying AI has been established and motivated by the scientific community. The definition of DAI can be clearly summarized in the similar words of Chaib-Draa et al. [1992] as follows:

Distributed artificial intelligence systems were conceived as a group of intelligent entities that activated by cooperation, by coexistence or by competition.

The interests given by researchers for DAI lead to implement it in many ways. Multiagent systems can be considered as typical DAI systems in which several agents interact or work together in order to achieve goals. Indeed, multiagent approaches and metaheuristics are two independent problem-solving paradigms with different characteristics. Recently, these two paradigms have been combined, resulting in agent-based metheuristic algorithms that have been widely proposed, particularly for nature-inspired, hybrid and distributed metaheuristics [Kazemi et al., 2009, Leitão et al., 2012, Zhao et al., 2005]. The main advantages of using multiagent approach for metaheuristics include:

1) It may take advantage of the distribution and robustness inherent to multiagent systems by applying some agent properties within algorithms, such as communication, cooperation and learning.

2) It may offer the promise of higher computational speeds for solving complex problems by using the inherent asynchrony and parallelism in agents.

3) It may bring a new perspective in problem solving, by designing hybrid algorithms.

The aim of this chapter is to describe a multi-level hyper-heuristic pattern by following the previous work in Meignan et al. [2008]. More precisely, Meignan et al. [2008] proposed the Agent Metaheuristic Framework (AMF), which is based on an organizational model describing metaheuristics in terms of roles. These roles correspond to the main components or tasks in a metaheuristic: intensification, diversification, memory and adaptation or self-adaptation. From this point of view, we attempt to introduce an organizational view of cooperative hyper-heuristics.

This chapter is organized as follows: Section 2 overviews some related work on metaheuristic and hyper-heuristic frameworks. Section 3 presents the conceptions of Role-Interaction-Organization model and the AMF model. In section 4 we present the organizational view for cooperative hyper-heuristics.

2 Metaheuristic and hyper-heuristic frameworks

It is critical to develop a useful framework to make metaheuristics or hyper-heuristics more simple and adaptable, since the algorithms are defined as specification once some common key properties have been recognized. Moreover, it is useful to compare existing algorithms and to provide a general description. In this section, we overview some conceptual and agent-oriented viewpoints to propose the frameworks in the literature, which enable the design and implementation of metaheuristics or hyper-heuristics in different ways.

***I&D Frame*, (*Intensification and Diversification Frame*)**

An important problem in the design of metaheuristics is to achieve the balance between two contrasting needs: on one side, needs to intensively search in areas of search space offering high quality solutions, and on the other side, needs to move to unexplored areas of search space in order to diversity the search. These two opposed guides go under the names, respectively, of diversification and intensification. A way of visual components of metaheuristic on intensification and diversification has been proposed by Blum and Roli [2003] in **I&D** Frame (Intensification and Diversification Frame). In this formalization, metaheuristics are analyzed in terms of intensification and diversification components (**I&D** components). These components correspond to operators, strategies or actions used to conduct the search, which are depicted as a triangle with the three corners (see in Fig. 3.1). The **OG** corner corresponds to those components guided only by the objective function.

The **NOG** corner refers to those components guided by other functions rather than the objective one, again without using any random component. The third corner, denoted **R**, comprises those components that are totally random, which plays an important role in many metaheuristics. This component approach has tried to help analyzing existing heuristics and designing new heuristics.

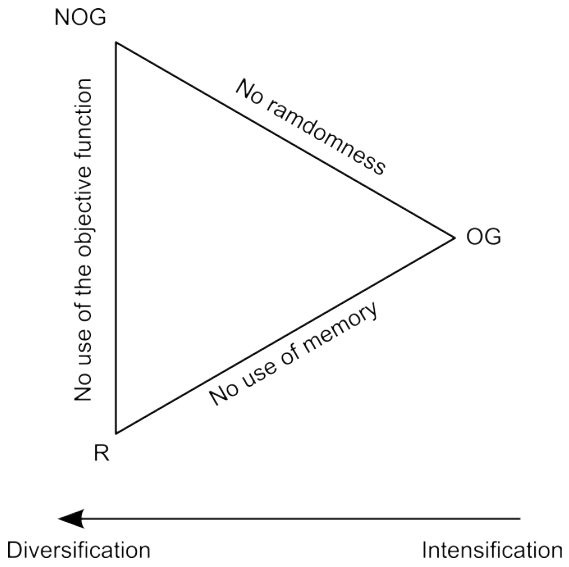

Figure 3.1: The **I&D** frame [Blum and Roli, 2003]

***AMP*, (*Adaptive Memory Programming*)**

AMP (Adaptive Memory Programming) introduced by Glover [1997] aims to define the strategic memory components in metaheuristics which guide the intensification and diversification processes. Specifically, memory in AMP, which can be defined as global or inter-individual, stands for the information collected by an algorithm on the objective function distribution. It can be represented either as a simple set of points or as more complex structures. The concept of memory has been extended in Taillard et al. [2001] to produce an unified view of metaheuristics. In this scheme, a metaheuristic can be viewed as an iterative process summarized in Algorithm 3.

Algorithm 3: AMP algorithm scheme [Taillard et al., 2001]

```
1 Initialize the memory
2 while stopping criterion is not met do
3 |   Generate a new provisional solution s using data stored in the memory
4 |   Improve s by a local search; let s' be the improved solution
5 |   Update the memory using the pieces of knowledge brought by s'
6 end
```

ALS, (*Adaptive Learning Search*)

In Dréo et al. [2007], the authors presented ALS (Adaptive Learning Search) as a framework for considering the structure of metaheuristics, based on the AMP approach. The major difference is that a learning phase is considered in ALS instead of considering only a memorization process in AMP. The reason given by the authors for this is that the memory concept is quite static and passive. Moreover, it suggests that the metaheuristic only takes into account the previous iteration, without considering the whole optimization process. Thereby, a three-term to describe the main steps is proposed by Dréo et al. [2007] for a population metaheuristic: learning, diversification and intensification, with respect to a sampling either explicit, implicit, or direct. An ALS algorithm is defined as follows.

Algorithm 4: ALS algorithm scheme [Dréo et al., 2007]

```
1 Initialize a sample
2 while until stopping criteria do
3     Sampling: either explicit, implicit or direct, Learning: the algorithm extracts
      information from the sample, Diversification: it searches for new solutions,
      Intensification: it searches to improve the existing sample, Replace the previous
      sample with the new one.
4 end
```

The **I&D**, AMP and ALS frameworks allow to describe different metaheuristics using a limited set of generic concepts. However, they are too general to be used as a framework to design metheuristics [Milano and Roli, 2004] and lack the consideration of dynamic adaptation [Crainic and Toulouse, 2003]. In fact, it is hard to describe the structure, the interactions and the relations of the algorithm to the optimization problem [Danoy et al., 2010]. Thus, some agent frameworks have been proposed to tackle these issues by benefiting from the point of multiagent view.

MAGMA, (*MultiAGent Metaheuristics Architecture*)

Milano and Roli [2004] introduced a multiagent architecture called the MultiAGent Metaheuristic Architecture(MAGMA) conceived as a conceptual and practical framework for metaheuristic algorithms. The authors tried to identify the common principles and basic components underlying metaheuristic algorithms in order to hybrid and implement them easily. In this architecture, a metaheuristic is a multiagent system composed of four levels each of which corresponds to a different level of abstraction. Fig. 3.2 depicts these different levels in MAGMA. Level 0 is composed by the agents which consist in providing a feasible solution for the upper level; it can be considered the solution level. Level 1 deals with solution improvement such as local search. The agents in this level perform a trajectory in the fitness landscape until a termination condition is met; this can be defined as the level which

deals with neighborhood structure. Level 2 agents are used as a balance between the tasks of diversification and intensification; this can be defined as the landscape level.

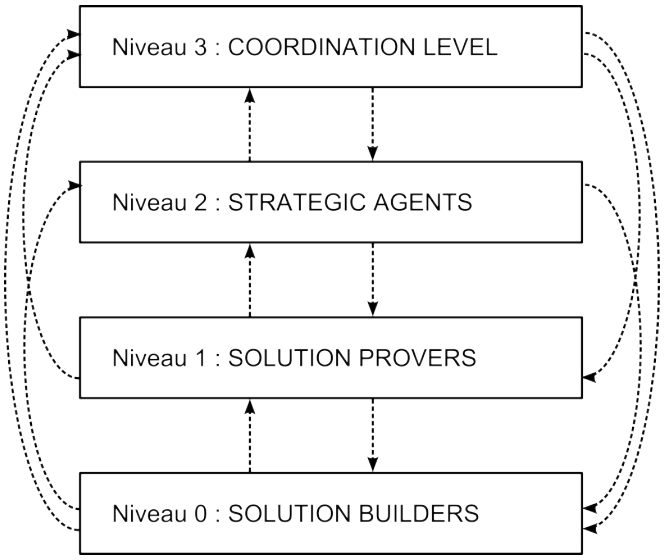

Figure 3.2: Multilevel architecture for metaheuristic algorithms [Milano and Roli, 2004]

Generic cooperative hyper-heuristic framework

In Ouelhadj and Petrovic [2009], the authors proposed an agent-based cooperative hyper-heuristic framework composed of a population of independent heuristic agents and a cooperative hyper-heuristic agent, as illustrated in Fig. 3.3. The heuristic agents perform a local search to improve their local solutions from the same or different initial solutions. They cooperate synchronously or asynchronously through the cooperative hyper-heuristic agent by exchanging the solutions of the low level heuristics. The cooperative hyper-heuristic agent, as a high level of hyper-heuristic, manages the cooperation between the heuristic agents, the overall selection of the low level heuristics, and the acceptance of their solutions. Furthermore, a variety of acceptance criteria, including AM (All moves), TS (Tabu search), IO (Improving Only), SA (Simulated Annealing) and GD (Great Deluge), has been investigated to decide whether to accept or not the selected solutions to be sent to the heuristic agents to diversify the search.

To summarize, we discuss some limitations of these agent-oriented frameworks. Although MAGMA can describe existing metaheuristics in a uniform way, it seems limited to describe the distributed approaches. Moreover, as the framework for the algorithms implementation, the concepts of self-adaptive strategy are not integrated into MAGMA. However, this type of strategies is often added in heuristic approaches to enhance the search capability. As stated in Ouelhadj and Petrovic [2009], the research into the investigation of the agented-based hyper-heuristic framework only focused on the role of cooperation between low level heuristics. Although this cooperative search in a hype-heuristic framework offers promising perspectives in cooperative hyper-heuristics, they didn't investigate the role

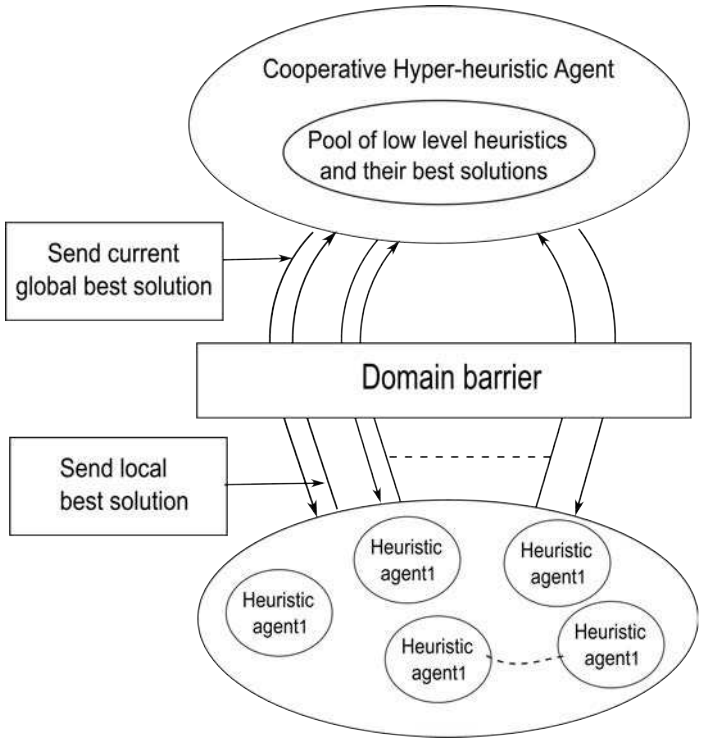

Figure 3.3: A cooperative hyper-heuristic search framework [Ouelhadj and Petrovic, 2009]

of cooperation between multiple hyper-heuristics to combine the performance of single hyper-heuristics. Hence, we can extend their cooperation mechanisms so that it increases the level of generality of framework. Moreover, learning process is not stressed in their framework. In many complex domains, however, learning is the only feasible way to train an approach to perform well.

3 Agent metaheuristic framework

Using several terms inspired by the social metaphors, some organizational models have been proposed to assist the design of systems [Ferber and Gutknecht, 1998, Ferber et al., 2004, Hannoun et al., 2000]. In Meignan et al. [2008], the authors proposed an organizational model to guide the design and analysis of metaheuristics. That is, the Agent metaheuristic framework (AMF) is introduced for the purpose of analyzing existing algorithms and facilitating the design of new metaheuristics. In this framework, a metaheuristic is viewed as an organization by using the Role-Interaction-Organization (RIO) meta-model [Gruer et al., 2002]. In this section we begin by introducing the concepts of the RIO meta-model. Then we discuss a definition of the term agent together with some characteristics and advantages of multiagent systems. Finally, we detail the Agent Metaheuristic Framework (AMF) proposed by Meignan et al. [2008].

3 .1 Role-Interaction-Organization meta-model

The RIO meta-model [A.Rodriguez, 2005, Gaud, 2007, Gruer et al., 2002, Hilaire et al., 2000, Meignan et al., 2008] takes its name from the three main concepts of meta-model: Role, Interaction and Organization. In the RIO model, an organization represents a set of roles and their interactions associated to the satisfaction of a goal or the execution of a global task. A role is an abstraction of a behavior or a status defined in an organization. It is associated to an objective to accomplish. Formally, the definition of role can be given as follows:

Definition 3.1 Role, [A.Rodriguez, 2005]

A role is the abstraction of a behavior in a certain context and confers a status within the organization. The Role gives the playing entities the right to exercise its capacities. Roles may interact the other roles defined in the same organization.

An interaction that links two roles is defined as follows:

Definition 3.2 Interaction, [A.Rodriguez, 2005]

An interaction links two roles in a way that an action in the first role produces a reaction in the second.

Finally, we give the definition of organization as follows:

Definition 3.3 Organization, [Hilaire, 2000]

An organization is defined by a set of roles, their interactions and common context. They define a specific pattern of interaction.

From a multiagent point of view, an agent can be considered as an active entity which plays roles. An agent may be associated to one or more roles and a role may be played by one or more agents. In fact, RIO provides a graphical representation of organizations. An example of the graphic representation of a RIO diagram is presented in Fig. 3.4. At an organizational level, we find two organizations. *Organization* 1 composed of three roles, noted *Role* 1, *Role* 2 and *Role* 3 respectively. *Role* 3, *Role* 4 and *Role* 5 are defined in *Organization* 2. The role playing relationship between roles and agents is dynamic. Namely, at any given time agents may request to play new roles and quit roles that they are currently performing. At the agent level the associations of roles to agents are specified. For example, *agent* 1 plays *Role* 1 and *Role* 2 in *Organization* 1.

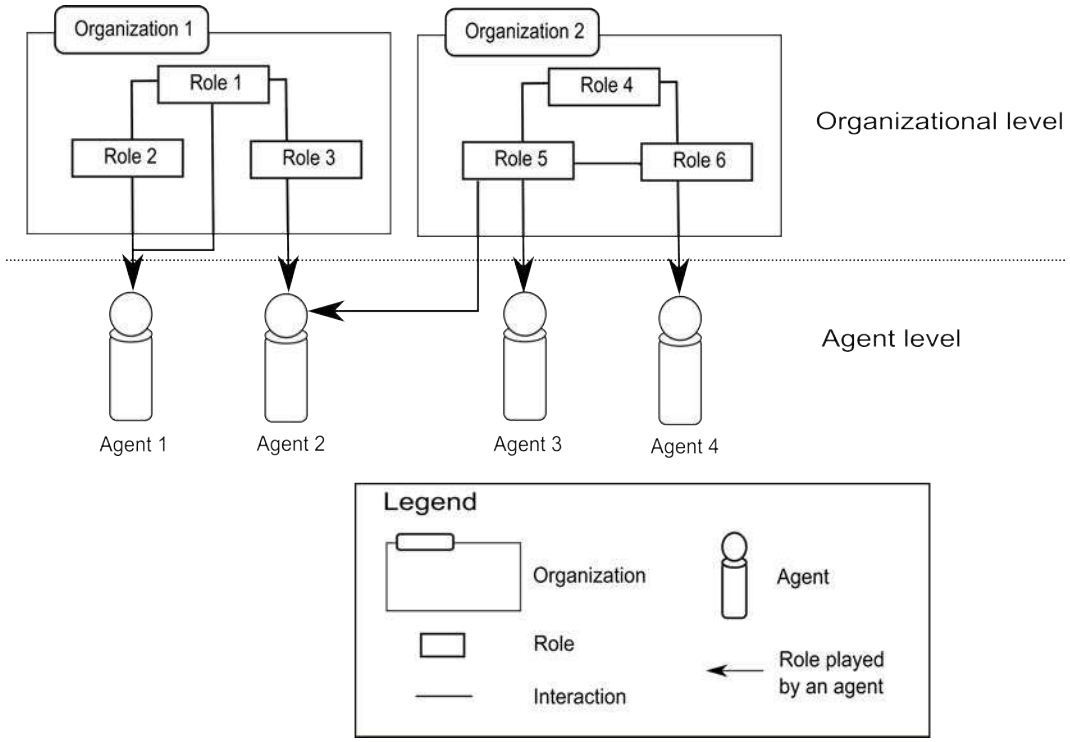

Figure 3.4: RIO model example

3 .2 Agent and multiagent systems

Using agent-based methods is not a new idea to solve problems. Despite its successful application on many problems, the definition of agent is till a topic of some debates in the theoretical artificial intelligence community. In the following, we cite some of the definitions which are used to refer to an agent.

Maes [1995] detailed that *"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."*

Hayes-Roth [1995] described that *"Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions."*

Smith et al. [1994] stated that *"Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller."*

In Jennings et al. [1998], the authors emphasized that an agent is a (software) system that enjoys the following properties:

-autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

-social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;

-reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;

-pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

In Ferber [1999], a minimal common definition of an agent is discussed

An agent is a physical or virtual entity

(a) which is capable of acting in an environment,

(b) which can communicate directly with other agents,

(c) which is driven by a set of tendencies or goals (in the form of individual objectives or of a satisfaction/survival function which it tries to optimise),

(d) which possesses resources of its own,

(e) which is capable of perceiving its environment (but to a limited extent),

(f) which has only a partial representation of this environment (and perhaps none at all),

(g) Which possesses skills and can offer services,

(h) Which may be able to reproduce itself,

(i) Whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation, and the communication its receives.

Russell and Norvig [2003] depicted a generic agent presented in Fig. 3.5. In this diagram, the authors depicted that an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. This way, a human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors, whereas a robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors.

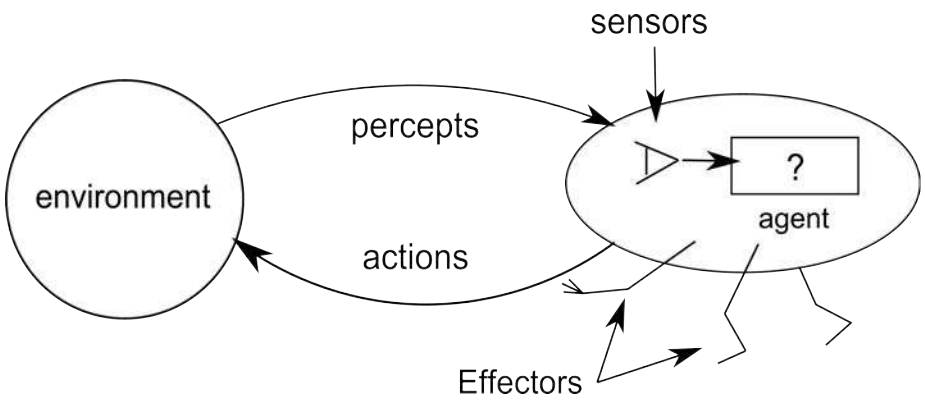

Figure 3.5: Agents interact with environment through sensor and effectors [Russell and Norvig, 2003]

Although there is no universally accepted definition of the term agent, there is a general consensus that autonomy is central of the notion of agency [Weiss, 1999]. In addition, for the purposes of our work, we consider an agent to be an entity, with states, actions, and situated in an environment. Formally, in the context of optimization, we give the definition of agent in a similar way as Milano and Roli [2004]:

Definition 3.4 Agent

An agent is an entity which can build a solution, move over a landscape, communicate with other agents by sharing information, and autonomously be active and adaptive upon the environment.

Distributed Artificial intelligence systems are usually divided into two main categories: multiagent systems (MAS) in which several agents coordinate their knowledge and activities and reason about the processes of coordinate; and distributed problem solving systems (DPS) in which the work of solving a particular problem is divided among a number of nodes that divide and share knowledge about the problem and the developing solution [Weiss, 1999]. Naturally, we can distinct that the emphasis of work on MAS is behavior coordinate, whereas DPS focus on task decomposition and solution synthesis. In this sense, a multi-agent system can be defined as

Definition 3.5 Multiagent System

A collection of autonomous agents, which are able to communicate and to coordinate with each other within an environment.

In Jennings et al. [1998], the authors summarized the main characteristics of a MAS as follows.

•*Each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;*

•*There is no global system control;*

•*Data is decentralised;*

•*Computation is asynchronous.*

In addition, all multiagent systems can be viewed as having dynamic environment since agents intentionally affect the environment in unpredictable ways [Stone and Veloso, 2000]. In this dimension, each agent is both part of the environment and modeled as a separate entity. An example is illustrated in Fig. 3.6. From this figure, we can see that there may be any number of agents, with different degrees of heterogeneity. They may interact directly (communicate) as indicated by the arrows between the agents.

Based on there characteristics, we can benefit the following advantages of MAS in our work. One comes from the parallelism that can be realized by MAS. Clearly, it is useful to make an approach more efficient, particularly in solving the large problems. Then, the

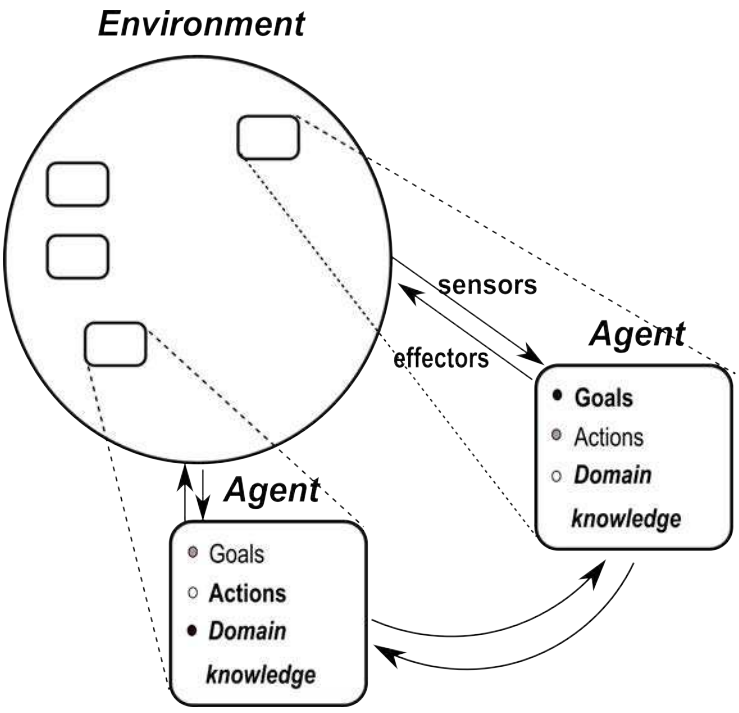

Figure 3.6: The fully general multiagent scenario [Stone and Veloso, 2000]

second advantage is scalability. Since a MAS is inherently modular, it should be easier to add new agents to the MAS than it is to add new capabilities to a monolithic system [Stone and Veloso, 2000]. Last but not least, multiagent systems provide insights about the design of approaches with cooperation and interaction, which may enlighten us on solving the problems in the face of enormous complexity.

3 .3 AMF

As mentioned earlier, the AMF presented in Meignan et al. [2008] proposed a framework based on an organizational model which can be used to describe both population-based metaheuristics and trajectory methods. This model describes a metaheuristic in terms of organization, roles and interactions. In fact, the AMF model extends the AMP scheme by adding the concepts of intensification, diversification and adaptation while keeping a high level of abstraction. In this framework, we can consider a metaheuristic as an organization composed of a set of roles which interact in order to find an optimal solution. Clearly, the goal of this organization is to efficiently solve the problem instance by providing high quality solutions in reasonable computing times. Intensification and diversification tendencies are combined to explore the search. During the search, structured information about the search space is used by subordinate procedures as heuristics in order to guide the exploration and balance these two tendencies. In addition, it is desirable to adaptively determine strategies to guide, intensify and diversify by learning from their search experiences. Four roles are defined from these observations: Intensifier, Diversifier, Guide and Strategist. The

resulting metaheuristic organizational model is described in Fig. 3.7.

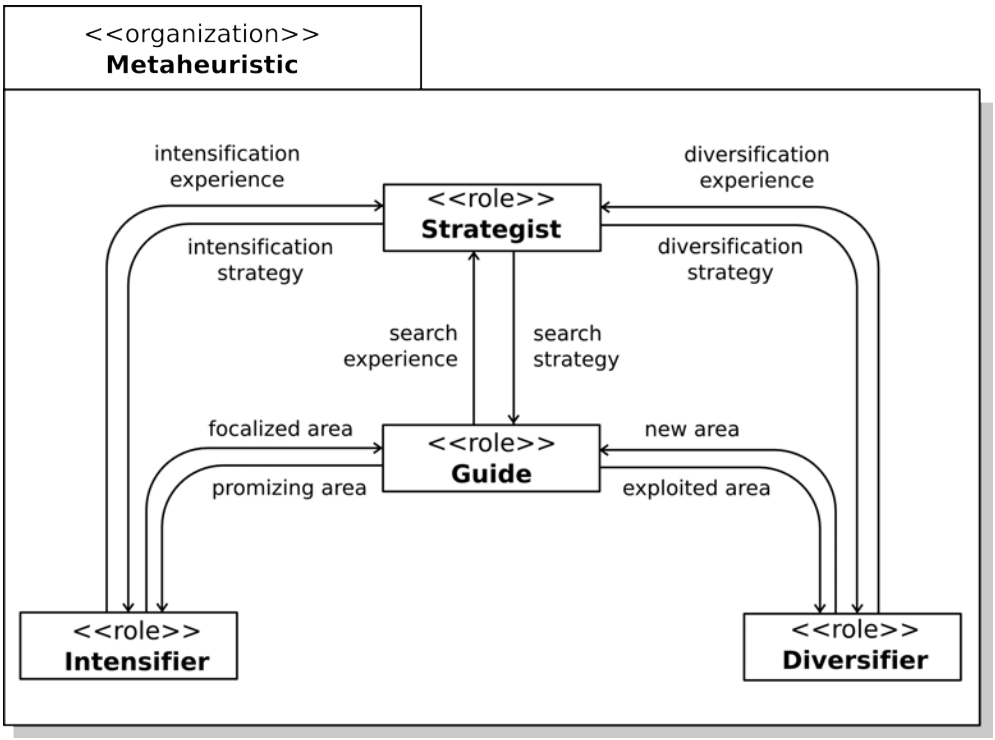

Figure 3.7: AMF organizational model of metaheuristic [Meignan et al., 2009]

To obtain a metaheuristic from the AMF organizational model, it is sometimes necessary to provide some methodological guidelines for the design of a particular metaheuristic starting from the AMF organizational model. By considering the result of design process is a multiagent system, Meignan et al. [2008] draw the following phases inspired by RIO methodology [Gruer et al., 2002]:

• AMF Roles refinement: It consists in determining the means that are required to perform the different roles described in the AMF organizational model.

• Agentification: It allows to determine the multi-agent structure of the metaheuristic.

• Metaheuristic specialization: It consists in specializing the multiagent system to treat a particular optimization problem.

More importantly, this model can be considered as an unified view of several metaheuristics. Following is an example from Meignan [2008] to illustrate how the components of metaheuristics can be expressed by a refinement of the AMF organizational model.

Island evolutionary algorithm

One of the ways of solving optimization tasks is using evolutionary algorithms [Holland, 1992], which are based on the iterative improvement of a population of solutions. It is a remarkable fact that there are numerous operators proposed and applied for specific purposes in evolutionary algorithms. Nevertheless, most algorithms emphasis on three genetic operators: selection, mutation and crossover. That is, individuals are selected and recombined

in order to generate new solutions that replace other ones by using these operators. Due to increasing demands placed on evolutionary algorithms for solving large problems, some of them drive naturally towards parallel processing. Island evolutionary algorithm is one of such algorithms [Tanese, 1989]. The resulting parallelization is that the overall population is broken into a relatively small number of subpopulations called islands. The islands evolve independently a number of generations and some individuals (normally the best) then transit between them called individuals migration. Fig. 3.8 shows an organization model of island evolutionary algorithm and its agentification. Briefly speaking, it is composed of three roles: Recombinator-Mutator, Selector and Coordinator. The Recombinator-Mutator role is a refinement of the Diversifier role since the aim of this role is to diversify the population by using the mutation and crossover operators, so as to avoid getting trapped within local optima. The Selector role selects better individuals by allowing them to pass on their genes to the next generation and prohibit the entrance of worst fit individuals into the next generations. Therefore, this role corresponds to the Intensifier role. In respect of agentification, there are three agents in this example. Each of them plays a set of roles and manages a subpopulation of solutions. The agents interact with each other when undergoing individual migration.

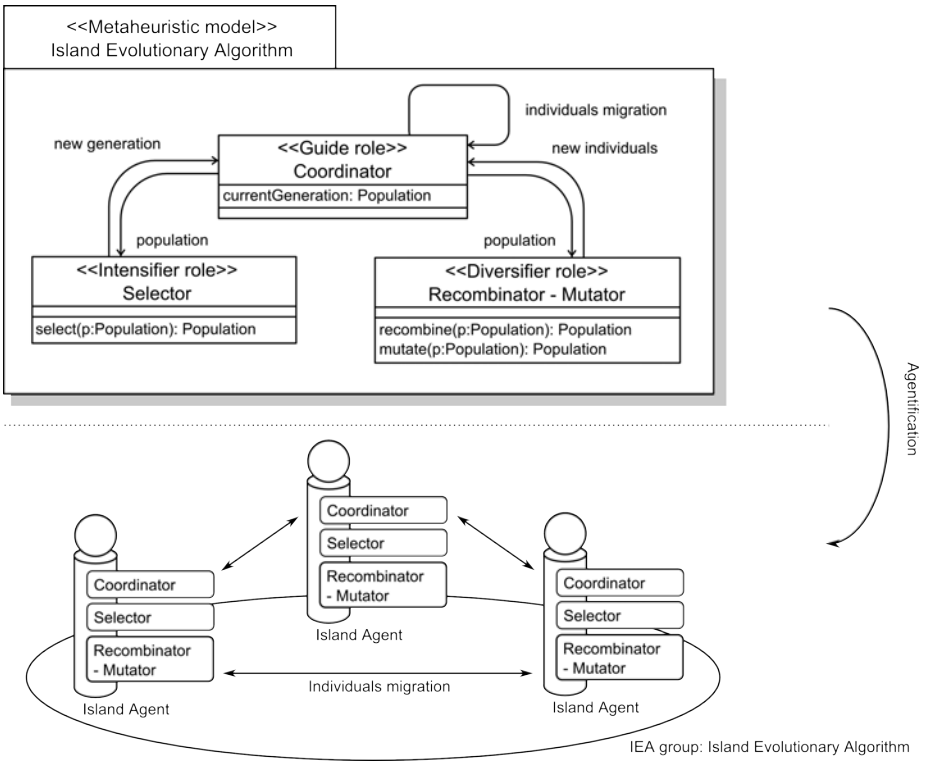

Figure 3.8: An organization model of island evolutionary algorithm and its agentification [Meignan, 2008]

4 From AMF to multi-level hyper-heuristic

The AMF encourages the designe of modular metaheuristics by the identification of common components. Although this promising direction of research leads to the successful applications, this model is sometimes too generic to support the design of metaheuristics. Moreover, it may not ensure the possibility to reuse the component to tackle different problems [Meignan et al., 2009]. As noted in the previous chapter, the hyper-heuristic methodology is proposed to address such a issue. Therefore, in our work, we focus attention on the role of cooperation between hyper-heuristics since it might make more sense to raise the level of generality. For the purpose of facilitating the design of cooperative hyper-heuristics, we propose a multi-level hyper-heuristic pattern, which is derived from the AMF. In agreement with the previous work on the AMF, we present this pattern in the following sections whereby the basis of RIO meta-model.

4 .1 Overview

A traditional hyper-heuristic framework, as represented in Fig. 3.9, is composed of two levels, a high level and a low level, with a problem domain barrier separating them. On the one hand, the low level includes a set of problem-specific heuristics called low level heuristics, which search directly on the solution space. On the other hand, high level usually operates a certain low level heuristic to be applied at a given step of the search process depending on the non-specific knowledge, such as the difference in the objective function, historical performance, etc.

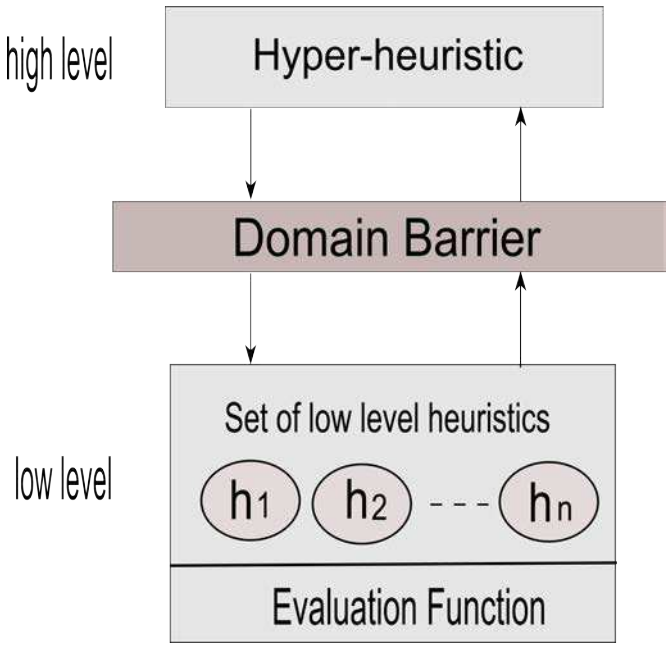

Figure 3.9: Hyper-heuristic traditional framework [Burke et al., 2003a]

From the RIO point of view, we define a cooperative hyper-heuristic as an organization.

The goal of this organization is to efficiently explore the search by combining a set of hyper-heuristics in the hope that it can find high quality solutions in reduced amount of time. In a cooperative hyper-heuristic search, where more than one hyper-heuristic is involved, the information must be exchanged in order to achieve the cooperation. The exploration is determined by intensification and diversification moves. However, to guide the exploration, finding a balance between two moves often depends structured information, such as record performances. Additionally, adaptiveness is a desired feature for search. As a result, the strategies employed to guide, intensify and diversify may be adapted whereby the search experiences. Five roles stems from this definition: Cooperation, Guide, Strategist, Diversifier and Intensifier. From these observations, the resulting organization called multi-level hyper-heuristic pattern is shown in Fig. 3.10. As can be seen from this figure, we define a three-level architecture that providers an additional level on top of a hyper-heuristic in order to make use of the cooperative search between hyper-heuristics. A top level thus refers to the Cooperation role. A high level contains the Guide and Strategist roles. The Diversifier and Intensifier roles are involved in a low level. The following subsections address three levels in the pattern more precisely.

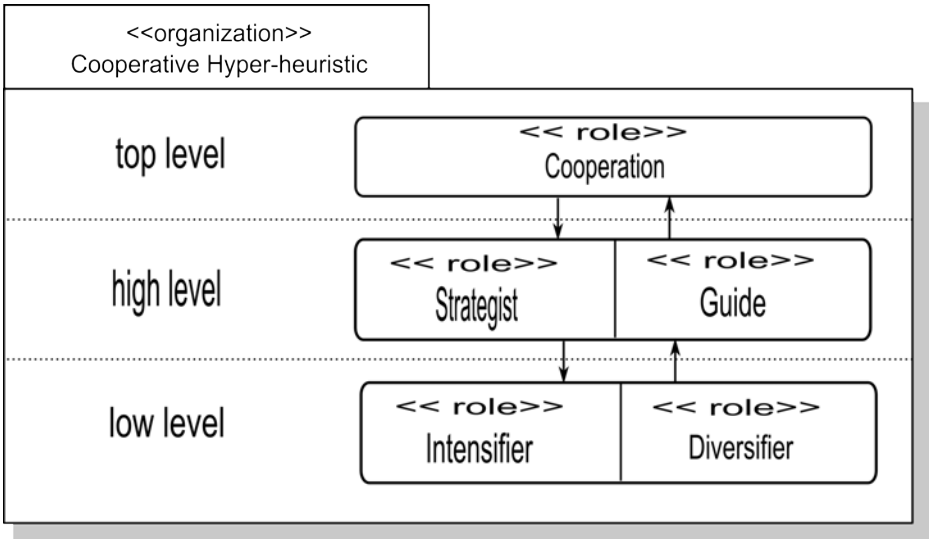

Figure 3.10: Organizational model of cooperative hyper-heuristic

4 .2 Top level

At this level, the distribution of the computation is organized on the metaphor of the coalition as proposed by Meignan et al. [2009]. Namely, the coalition is made up of several agents, which concurrently explore the search space and cooperate to improve their search abilities.

Each agent can be seen as a hyper-heuristic. The coalition structure is intended to support robustness and facilitate the distributed computation since control is decentralized and

communications between hyper-heuristics are asynchronous. Consequently, the removal or addition of any hyper-heuristic would not perturb the global functioning of the system. In our current study, we only investigate the improvement carried out by such a cooperative level of hyper-heuristics. The method is simulated on a single processor, but it is also intended to be easily implemented on multi-processor or asynchronous computer networks.

The Cooperation role allows hyper-heuristics to communicate. For example, hyper-heuristics can exchange information about their own solutions of any problem given. Alternatively, the behavior of hyper-heuristics, e.g., selection of low level heuristics, can be learned/shared between hyper-heuristics. The primary reasons behind communicating/sharing information between hyper-heuristics are to increase the level of generality and to speed up the exploration of the solution space for large and complex problems. It should be interested to notice that this level helps to design hybrid hyper-heuristics in many ways. Although all hyper-heuristics used here are homogeneous, it should be mentioned that this level also allows to achieve cooperation between heterogeneous hyper-heuristics. That is, the coalition can be composed of agents that are not identical. For instance, it is possible to form a hybrid of hyper-heuristics with a different set of low level heuristics. Alternatively, hyper-heuristics have the same set of low heuristics but different learning strategies.

Formally, we give the definition of the Cooperation role composing the top level as follows.

Definition 3.6 Cooperation Role

The Cooperation role combines the efforts of several independent agents by organizing the coalition. This role makes the information that agents must communicate available and accessible to agents. The goal of the Cooperation role is to provide a mechanism to allow the information exchanged so as to yield a more efficient global search.

4 .3 High level

A hyper-heuristic itself operates at a high level in the way of performing a search over a set of low level heuristics for solving the problems. At this level a hyper-heuristic is seen as an intelligent agent who takes decisions and learns from the past experiences. According to AMF the agent behavior is specified by roles that interact one with each other. Two roles are played by an agent for managing the low level heuristics during the search. They are respectively called Guide and Strategist roles.

How to obtain a balance between diversification and intensification has become a significant principle for successfully implementing algorithms. Here, the Guide role is responsible for such a balance by following a decision process. Specifically, the Guide role attempts to make a decision between intensification and diversification by imitating intelligent processes. That is, a kind of "memory" is used as the essential element of the Guide

role. The memory term in fact has been taken from Adaptive Memory Programming (AMP) scheme [Taillard et al., 2001]. In practice, memory can take several forms embedded into algorithms, such as a tabu list in tabu search, a population of solutions in evolutionary algorithms and the pheromone trail in ant colony algorithms. As summarized in Taillard et al. [2001], the common characteristics shared by these memory-based methods are the exploitation of a memory to construct a new solution, an improvement procedure to find an even better solution and a memory update procedure based on pieces of knowledge brought by the improved solution. Using the memory, we can identify main "activities" which may be performed by the Guide role within hyper-heuristic framework like the following:

- maintain a history of the search performed by low level heuristics;
- store the solutions found;
- avoid staying on regions which have been excessively exploited;
- offer the information about promising regions;
- provide dynamically the intensification and diversification.

To summarize, we formalize the definition of the Guide role as follows:

Definition 3.7 Guide Role

The Guide role directs, in the one hand, to diversify the search by exploring unvisited regions, in the other hand to intensify the search in order to obtain promising solutions. To this end, this role uses a memory, where the information necessary is recorded, for guiding purposes. The goal of the Guide role is to achieve a balance between the intensification and diversification.

Indeed, hyper-heuristics must address the issue of how to produce good sequences of the low level heuristics. Often, the most effective way is to incorporate a learning mechanism for the sake of assisting the selection of low level heuristics during the search process. One of the commonly used methods for learning is by using a choice function which adaptively ranks the low level heuristics [Burke et al., 2010a]. Another method is reinforcement learning due to its simplicity and effectiveness [Nareyek, 2004]. To take into consideration a cooperative search, if a hyper-heuristic trends to behave as the most efficient than another one, other hyper-heuristics must learn its behavior so as to compensate their lack of "knowledge". Therefore, the Strategist role corresponds here to the adaptation or self-adaptation mechanisms used in hyper-heuristics. The goal of this role is to improve the performance of the search process and possibly to reduce parameter setting. Obviously, adaptation can be considered as a distinguishing feature of this role. Similar to the AMF [Meignan et al., 2009], the term of adaptation is characterized by the modification or adjustment of the search strategy resulting from the observation of experiences. Thus, adaptation mechanisms use some kind of feedback to determine the nature or amplitude of the change. In our study, we will apply a reinforcement learning approach in conjunction with a cooperative learning mechanism between agents called mimetism learning [Meignan et al., 2009]. Some important activities performed by the Strategist role might be:

- to distinguish desirable behaviors;
- to use the abstract knowledge for modifying the behaviors;
- to assign credit/blame associated with the low level heuristics.

The definition of the Strategist role can be formally given below:

Definition 3.8 Strategist Role

The Strategist role combines the use of different learning mechanisms so that the search behavior improves over time. The goal of the Strategist role is to adapt the search strategies according to the problem at hand.

4 .4 Low level

Intensification and diversification are two major issues when designing a global search method [Blum and Roli, 2003]. Diversification generally refers to the ability to visit many and different regions of the search space, whereas intensification refers to the ability to obtain high quality solutions within those regions. Thus, the Intensifier and Diversifier roles respectively represent the intensification and diversification procedures or tendencies. In practice, intensification and diversification can be carried out in many ways. By analyzing most of popular metaheuristics, we can observe that the way to achieve intensification is mainly by a local descent procedure. A typical example can be taken from the tabu search [Glover and Laguna, 1998]. The main way to achieve diversification is to use perturbation techniques. For instance, crossover in the genetic algorithm [Holland, 1992] makes sure new solutions by swapping parts of existing solutions. In a hyper-heuristic, a promising way to achieve intensification and diversification is to combine several low level heuristics with strong specialization for intensification of diversification. Obviously, this is done at this level. More precisely, operators specified for intensification are used in local search procedures, whereas operators for diversification usually apply the perturbation to solution.

Formally, the definitions of the Diversifier and Intensifier roles can be respectively expressed as follows:

Definition 3.9 Intensifier Role

The Intensifier role is engaged in concentrating the search. The goal of this role is to exploit deeply in promising area of the search space.

Definition 3.10 Diversifier Role

The Diversifier role refers to the exploration of the search space. The goal of this role is to move the search to unexplored areas.

5 Summary

Due to the importance of combinatorial optimization problems for the scientific as well as the industrial world, the field of metaheuristics is a rapidly growing field of research for solving various combinatorial optimization problems. The different components and concepts are used among these metaheuristic approaches. Therefore, it was interested in proposing simple, flexible, robust and modular metaheuristics. Several frameworks of metaheuristics have been proposed in order to carry out these features and to analyze their similarities and differences. This chapter outlines some frameworks for metaheuristics, and also hyper-heuristics. In these previous works, we observe that agent-oriented frameworks seem to be a promising field of research. However, there are some limitations of these existing approaches in some regards. The interest in cooperative search frameworks has risen due to successes in combining novel search algorithms. Nevertheless, cooperative algorithms are relatively new. At the same time, most of current design of cooperative search algorithms focus on metaheuristics. Based on the AMF model, we investigate the multi-level hyper-heuristic pattern that provides an organizational description of cooperative search in context of hyper-heuristics.

In a traditional hyper-heuristic framework, there is two levels: high level and low level. The low level is in charge of building the solutions, whereas the high level operates a set of low level heuristics in a strategic way. Here, we extend the traditional framework to describe a multi-level heuristic pattern which contains three levels: top level, high level and low level. Clearly, the top level is an additional level. Multiple hyper-heuristics enable to communicate the collected information in regions of solution space through this level. At the same time, it can help us to design new cooperative algorithms with this extra level in the context of hyper-heuristics. The high level operates on heuristics rather than directly on the solutions by indirectly choosing a low level heuristic at each step. Within the pattern presented here, this level means to improve individual searches by learning. The reason behind learning is to choose the most appropriate low level heuristic from a set of heuristics during the search based on the experiences accumulated. Observing the similarities between metaheuristics, we can identify two basic elements: intensification and diversification. The intensification consists in looking further into the exploration of certain areas of the solution space, offering the promising solutions. The diversification can be considered as the restart searches when stagnation is encountered during the search e.g., no improvements after a specif number of algorithm cycles. Therefore, the low level here addresses a problem at hand by designing a set of low level heuristics for intensification and diversification.

The organizational approaches have offered the new ways for analyzing, designing and implementing MAS. The AMF is proposed as an organizational and multiagent framework to design and hybridize metaheuristics. The advantages of the AMF shed new light on the cooperative search in a hyper-heuristic framework. Enlightened by these previous works, we use an organizational model to describe a cooperative hyper-heuristic search. In a sim-

ilar way, we define several roles, each is an abstraction of a behavior in an organization. More precisely, five roles are used in our model: Cooperation, Guide, Strategist, Diversifier and Intensifier. As explained previously, the concept of role does not match to a particular agent, in a contrary, it can be played by several agents. At the same time, an agent can also play several roles. Indeed, two roles are linked by an interaction. From the RIO point of view, the interaction that links between the Guide role and the Intensifier role (or the Diversifier role) corresponds to information about the regions of solution space. For instance, the best solution found in certain areas of the solution space is an interaction between the Guide role and the Diversifier role. For the Cooperation and the Strategist roles, the interaction is the most efficient behavior of agent found.

In short, the pattern presented in this chapter serves as a new way to design the cooperative hyper-heuristics. To test our proposed pattern and to observe the advantages of cooperation in the context of hyper-heuristics, we propose a new type cooperative hyper-heuristic in the next chapter.

CHAPTER 4

A TWO-PHASE COOPERATIVE HYPER-HEURISTIC APPROACH FOR BUS DRIVER SCHEDULING

1 Introduction

To the best of our knowledge, there is very little research work on the bus driver scheduling problem by using hyper-heuristics. In this chapter, we introduce the two-phase cooperative hyper-heuristic approach (TPCH, for short) based on the multi-level hyper-heuristic pattern. The aims of this research are:

• to valid our proposed pattern carrying out the characteristics of generality, flexibility and scalability;

• to propose a novel cooperative search approach;

• to exclusively devote the bus driver scheduling problem, particularly in solving real-world instances.

We present our proposed approach in this chapter. The principles of TPCH are outlined in section 2 . Firstly, we depict a general structure of TPCH in section 3 . Section 4 details the decision process related to a TPCH agent. Then, we put the emphasis on learning mechanisms in sections 6 to 4 . Finally, section 7 gives the low level heuristics used to solve the bus driver scheduling problem.

2 Method principles

2 .1 Strategies of cooperation

Cooperative search is a category of parallel algorithms, in which several search algorithms run in parallel in order to solve the optimization problem at hand [El-Abd and Kamel, 2005]. The authors in Crainic and Toulouse [2003] distinguish three parallel strategies for metaheuristics. The first strategy aims directly to reduce the execution time of a given solution method without achieving higher quality solutions. Namely, this kind of strategy cannot improve solutions but runs faster when compared with a sequential one. In the second strategy, parallelization is obtained by partitioning the set of decision variables. It is

generally implemented in a master-slave framework without direct interactions between the search processes. Often, the resulting framework of this strategy is a master-slave structure. The last parallel strategy consists of several concurrent searches in the solution space. Although each concurrent search may implement the same heuristic method independently, they usually communicate to identify the best overall solution during the search. The parallel strategy of our approach falls into this last category. In general, there are two fundamental strategies for the design of cooperation among agents: centralized cooperative strategy and decentralized cooperative strategy. In centralized cooperative strategy there is a central agent that conducts the interchange of information between the processes and makes the decision on the explicit steps made by the individual process. In decentralized cooperative strategy, however, each search process has its own rules to decide when and how interchange the relevant information with other processes [Alba, 2005]. Theoretically, cooperative architectures are more robust and scalable when used in a decentralized manner. Additionally, communications among agents are intended here in a way such that the removal or addition of any agent would not perturb the global functioning of the system. For the above reasons, TPCH has been designed by considering a parallel hyper-heuristic approach with a decentralized strategy. That is to say that an agent which acts as a central controller does not exist. Furthermore, every agent is independent, and can communicate with others by sending individual messages.

Fig. 4.1 illustrates our cooperative architecture. As we have seen, there is not a special agent which dictates or centralized control to manage the agents. The arrow lines represent the information exchanged. The agents, on the one hand, visit search space independently. On the other hand, any agent can share information with the other concurrent agents. To summarize, the strategy of cooperation used in our approach consists in sharing the information gathered by the agents while they perform search independently on the solution space.

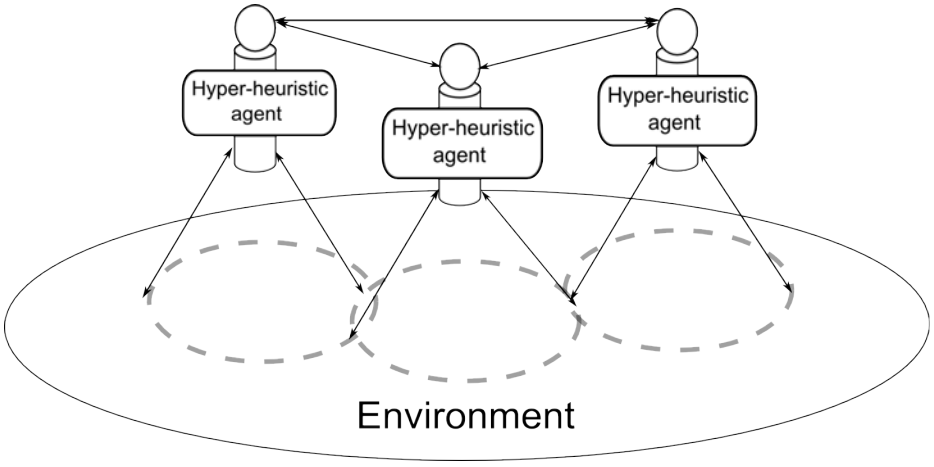

Figure 4.1: Distributed cooperative architecture

2 .2 Reinforcement learning

In TPCH, the problem of selecting the most appropriate low level heuristics is viewed as a reinforcement learning problem.

Machine learning, a branch of artificial intelligence, is the study of computer programs and algorithms that automatically improve their performance through experience. Reinforcement learning (RL) belongs to the category of machine learning algorithms. We begin to explain reinforcement learning by giving a simple example. As discussed in Dayan and Watkins [2006], one way in which animals acquire complex behaviors is by learning to obtain rewards and to avoid punishments. Consider teaching a dog to achieve a task, you do not need to tell it how to do, but you can reward/punish it if it does the right/wrong thing. By using this feedback, after many trials the dog will learn a behavior to achieve the task while avoiding any punishments. The behavior of this dog in this example can be considered as a type of learning, that is, a way of training dog by reward and punishment without needing to specify how the task is to be achieved. In fact, reinforcement learning theory is a formal computational model of such type of learning.

As discussed previously, multiagent systems are rapidly finding applications in a variety of domains. The complexity of many tasks arising in these domains makes them difficult to solve with designed agent behaviors in advance. It is common to make the agents to discover a solution using learning strategies. Recently, there has been growing interest in extending RL to the multi-agent domain [Busoniu et al., 2008, Panait and Luke, 2005]. In Fig. 4.2 we depict the agent-environment interface. In the reinforcement learning system, an agent interacts with its environment to achieve a goal. On each step of interaction the agent observes the current state, s, of the environment; the agent then choose an action, a, from a set of possible actions in that state. The action changes the state of the environment, and the value of this state transition is communicated to the agent through a reward, r. A policy π maps states to actions (or action probabilities). Agents observe their individual states and perform actions for which numerical rewards are given. Thus, the agent's goal is to maximize its long-term cumulative reward by learning an optimal policy that maps states and actions.

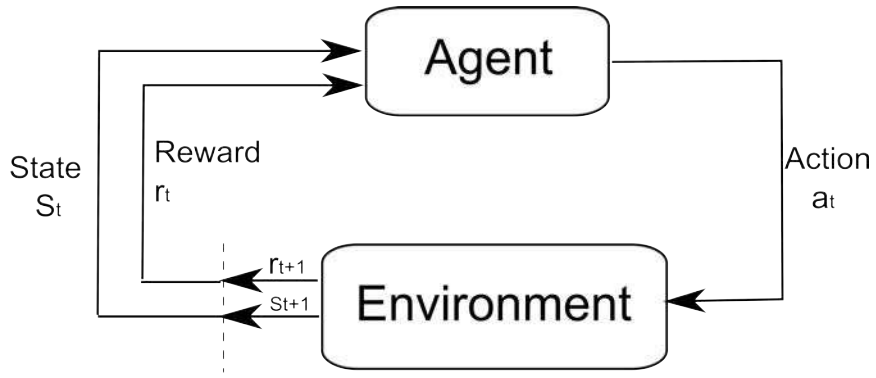

Figure 4.2: The agent-environment interaction in reinforcement learning

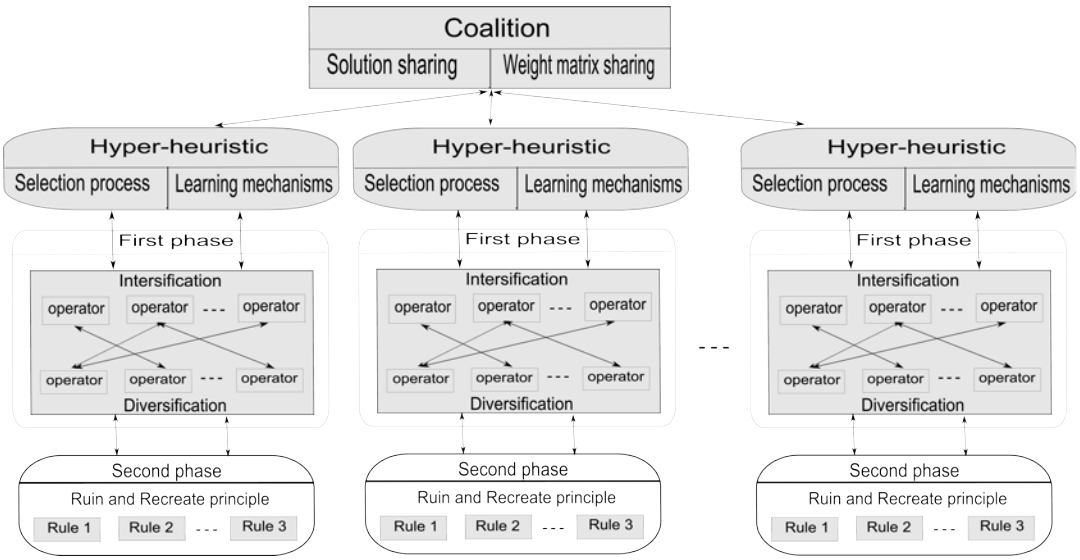

Figure 4.3: Two-phase cooperative hyper-heuristic approach

Reinforcement learning can provide a robust and intelligent way for agents to learn how to coordinate their action choices in environment. Using RL as a means of achieving coordinated behavior is attractive because of its generality and robustness.

3 General structure of TPCH

Based on the multi-level hyper-heuristic pattern, the proposed approach is illustrated in Fig. 4.3.

Briefly speaking, each agent can be viewed as having two phases: the fundamental principle of the first phase is to provide the interplay between intensification and diversification of search, whereas the second phase is related to the ruin and recreate search principle. The behavior of agents is based on four components: operators, rules, selection process and learning mechanisms. The operators are used to implement intensification or diversification tasks. Intensification operators refer to improvement process based on local search procedures, and diversification operators correspond to three types of perturbation procedures. Since, when solving a given problem, local search procedures alone cannot escape from a local optimum, the latter provides different degrees of diversification which perform either different number of perturbation moves or different types of moves used for perturbation. Moreover, a set of rules plays also the role of diversification in the ruin and recreate procedure when more diversification is needed. The order in which operators are executed is determined by the selection process. For each application of an operator to the current solution, the agent's rewards and states are stored in what we call an experience. Based on the experiences accumulated, learning mechanisms are applied in order to choose the most appropriate operators in the selection process. It is important to note that agents have no synchronization point and doesn't necessitate shared memory.

Overall, our approach is described in Algorithm 5. In this algorithm, each agent manages three problem solutions: a current solution $S_{current}$, the best solution from an agent $S_{bestfound}$ and the best solution from the entire coalition $S_{bestcoalition}$. When a solution is received (line 24), this last one is noted $S_{received}$. The attribute W is a weight matrix, r refers to the rewards and E denotes the experience memory. This memory stores o (operator index), t (execution time of operator), s (state) and fitness value for each operator application. The decision process determines the sequence of operators to apply (lines $9-10$). Then, the agent's set of solutions is updated (lines $12-26$) and the experience is stored after each application of an operator to the current solution. Based on the experiences accumulated, learning mechanisms modify the rules of the decision process. Moreover, if the obtained result remains unchanged after a given number of consecutive iterations (line 18 and line 34), it seems that the diversification is not sufficient, and it is necessary to make large 'jump' to quickly move out of unpromising regions of the solution search space. Thus, the search goes into a ruin and recreate procedure (lines $34-38$). After the application of a set of rules (line 35), $S_{current}$ is returned to the first phase (line 36) and a counter for consecutive non-improving local optima is restarted from zero (line 37). Note that the given number of consecutive iterations can be computed via a parameter $p\%$. The $p\%$ parameter determines the percentage of the maximum number of iterations allowed in solving the problem.

Some advantages of the multi-level hyper-heuristic pattern can be observed in the algorithm. First, it is possible to distinguish the realization of defined roles in the pattern. Realization of Cooperation role engages the communication among agents by sharing the information (lines $21-22$). Guide role performs the choice of operators (lines $9-10$) and updates the set of solutions (lines $12-26$). After one operator is applied, the experience is stored. Strategist role observes the experiences and modifies the decision weights using individual and mimetism learning (lines $28-32$). Realization of Intensifier and Diversifier roles corresponds to the application of low level heuristics (line 10 and line 35). Thus, the algorithm can be viewed as a particular schedule of roles. Second, all procedures in Algorithm 5, except the operators and rules application, are problem independent. This point follows the hyper-heuristic approach, and ensures the flexibility of TPCH.

4 Selection process

The performance of hyper-heuristics depends on the selection of a promising low level heuristic at each step to use its ability during the search. Here, we combine the selection operators (low level heuristics) with a reinforcement learning scheme to improve the decision making process, called first phase in our approach.

In the reinforcement learning scheme, as explained in subsection 2 .2 of this chapter, an agent perceives a state of the environment and takes an action, which causes the environment to transit into a new state. It receives a scalar signal, called reinforcement, from its

Algorithm 5: Description of the TPCH algorithm

```
/* Initialization */
Set maximum number of iterations i_max
Set threshold percentage p%
Set count value j ← 0
W ← init_weight_matrix()
E ← init_experience_memory()
for i = 1, ..., i_max do
    /* Choose and apply an operator */
    op← choose_operator(W, s)
    S_new ← apply_operator(S_current, op)
    /* Update solutions */
    S_current ← S_new
    if the S_bestfound improved by operator application then
        S_bestfound ← S_current
        r ←compute_rewards()
    end
    else
        j ← j + 1
    end
    if the S_bestcoalition improved by operator application then
        broadcast_solution(S_bestcoalition)
        broadcast_weight_matrix(W)
    end
    if new best coalition solution received from another agent then
        S_bestcoalition ← S_received
    end
    /* Learning mechanisms*/
    r ←compute_rewards(E)
    individual_learning(r)
    if weight matrix received from another agent then
        mimetism_learning(W, W_received)
    end
    /* Ruin and recreate */
    if j = i_max * p then
        S_new ← apply_rules(S_current)
        S_current ← S_new
        j ← 0
    end
end
Output S_bestcoalition
```

environment depending on the action taken. The reinforcement can be positive (reward), negative (punishment), or 0. Before we use multi-agent systems to solve problems, four elements should generally be defined. The first is to define what an agent is. The second is

to define the environment where all agents live. The third concerns the definition of state. The last is a definition of action that each agent can take to achieve its purpose. As mentioned already, a hyper-heuristic is referred to as an agent. In the context of hyper-heuristics we define the state, the action and the environment as follows. Let $H = \{h_1, h_2, \ldots, h_m\}$ be a set of the low level heuristics. Given a problem to solve, a low level heuristic h is selected to apply from H at each decision point of the search. The state s_i indicates the application of the low level heuristic $h_i, h_i \in H$. In a given state, the selection of the low level heuristic $h_j, h_j \in H$ corresponds to the possible action a_j. The environment can be viewed as a problem domain offering the rewards, which estimate the performances of selected low level heuristics. After one action is applied, a new solution and several rewards can be obtained from the environment, restarting the new state at the same time.

Let $S = \{s_0, s_1, \ldots, s_m\}$ be the set of the states, $A = \{a_1, a_2, \ldots, a_m\}$ be the set of actions. Each state-action pair $\langle s_i, a_j \rangle$ is assigned a weight $w_{i,j}$, which represents a degree of suitability of the action in the given state. Namely, a weight $w_{i,j}$ is associated to each action $a_j, a_j \in A$, for the state $s_i, s_i \in S$. The effective choice of execution of an action is performed by a *roulette wheel selection principle*. Thus, the probability $P(a_j \mid s_i)$ to apply the action a_j in the state s_i is computed using the following formula:

$$P(a_j \mid s_i) = \frac{w_{i,j}}{\sum_{k=1}^{m} w_{i,k}} \tag{4.1}$$

with:

$S : (s_i)_{i=0,\ldots,m}$; Set of states

$A : (a_j)_{j=1,\ldots,m}$; Set of actions

$W : (w_{i,j})_{i=0,\ldots,m;j=1,\ldots,m}$; Weight matrix

5 Individual learning mechanism

Initialization of the weight matrix is made with the initial weight w. The weights that correspond to undesirable actions are set to zero. Here, we set the weight $(w_{j,j})_{j=1,\ldots,m}$ to zero in order to prevent selecting successively the same actions. At the beginning of the optimization, the learning state is set to the state s_0. Then, a learning process will adjust the weights according to the past experiences of the agent. An example of a weight matrix is shown in Fig. 4.4 where the first column indicates the states; the first line indicates the actions. In this example we can easily see that there are four actions a_1, a_2, a_3 and a_4. Each state indicates the application of one action. As it can be seen from Fig. 4.4, an agent selects the action a_2 at the beginning and perform it, and then the agent senses the current state s_2.

It should be noted that a special state *local optimum* is reached when all the actions have failed to improve a given solution after the consecutive steps. Therefore, we propose a ruin

Figure 4.4: An example of a transition step using the weight matrix

and recreate procedure, called second phase in our approach, in an attempt to overcome this difficulty. In the second phase the search randomly chooses a sequence of rules and then applies each rule once.

Using reinforcement learning techniques for the learning selection in hyper-heuristics is not new, having been proposed by Nareyek [2004]. In the majority of hyper-heuristics using reinforcement learning techniques, a low level heuristic is rewarded if it improves a solution, while a negative reinforcement is applied if it fails to do so. However, the reward seems inappropriate only for the improvement of solution. As stated in Bai et al. [2007], although some heuristics cannot improve the solution directly, they are still useful in creating some intermediate situations to reach the optimal solution (or a good quality solution). In addition, the positive behaviors of action (e.g. search efficiently) may be considered when we give the rewards to actions. Here, we assume that a good solution obtained by agent benefits from three particular cases to be rewarded. Table 4.1 presents these particular cases and indicates the quantifiable rewards. It is clear that other particular cases can be easily introduced, alone or in conjunction with those described below, without modifying the agents' architecture.

Let us dicuss a question before we detail the particular cases used in our approach. In fact, for the purpose of estimating the actions in a given state, it is common to compare their performances. In this sense, the following question has to be answered. How to measure the performance of actions? To answer this question, we introduce an improvement value, which allows to measure the ability of finding a good solution quickly. Formally, the improvement value $C_{i,j}$ can be given by the following formula.

$$C_{i,j} = \frac{F}{T} \tag{4.2}$$

with:

$C_{i,j}$; Value related the action j in the state i

F; Improvement on fitness values

T; Execution time

Now let us return to the following three cases in which the rewards can be given.

Table 4.1: Particular cases of rewards

Case	Description	Value of reward
Case 1	A new best solution is found.	1
Case 2	The actions which are not applied in a given state are rewarded if the obtained solution remains unchanged.	0.3
Case 3	The fitness value improvement per unit execution time is more than the one of other actions in the same state.	0.5

Case 1 is the most important criterion to gauge the performance of action. At each iteration, the reward is given as follows: when an action results in a better solution than the best solution found we update the weights of the corresponding action with a reward of 1, otherwise no reward is given. In practice, this simple binary reward may be the most frequently used as a learning rule in the context of hyper-heuristics.

Case 2 leads to select actions which are not applied by raising their weights so as to insure the best balance as possible between exploration and exploitation. It has been mentioned already that in each state an agent must select an action. For the purpose of obtaining rewards in short term, the agent, on the one hand, selects an action with highest weight in a very natural way. On the other hand, the agent needs also to take other ones that may help it to obtain high rewards in long term and to avoid being stuck on sub-optimal policies. Several strategies have been proposed during the learning process for balancing exploration and exploitation. For instance, a strategy called GLIE is widely used to achieve a balance between exploration and exploitation by making randomly the decision between to choose the best and some other ones [Singh et al., 1998]. However, it is worth noting that this type of strategies may cause too much exploration that prevents from maximizing the rewards. Moreover, the agent cannot sufficiently exploit its knowledge in this situation. Here, we raise the possibilities to select the actions that are not applied if the obtained solution remains unchanged. In order to explain how it works, Fig. 4.5 shows an example of five actions in a state for the case 2. In this example, the agent would execute an action from a set of actions (a_1, a_2, a_3, a_4 and a_5) in state S_1, where each action has an improvement value except actions a_1 and a_4. Due to the fact that an action is not assigned an improvement value indicating that the action has not yet taken by the agent, actions a_1 and a_4 are not

yet applied in this example. If the solution obtained remains unchanged, it seems worth to discover new actions that may lead to good outcomes. For this example, as it is noted previously, we do not consider the application of actions a_1. Consequently, we reward action a_4 so that it can help the agent select this action in next step.

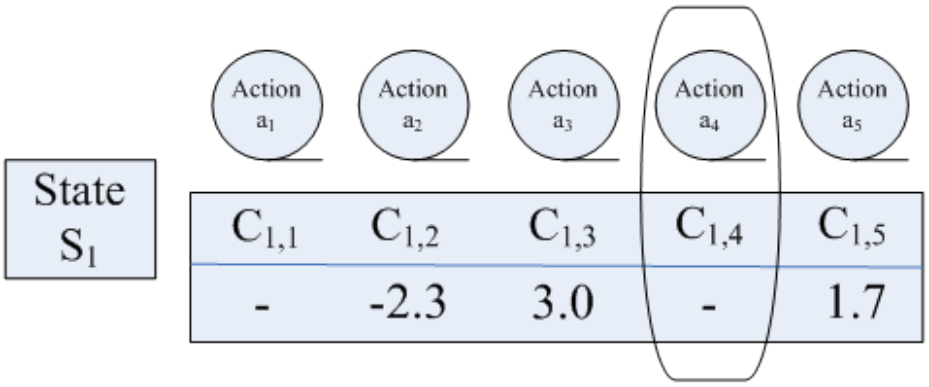

Figure 4.5: An example of the Case 2

Case 3 is used to reward well-performed actions. Once we obtain all the values C associated with the actions in a given state, a reward is given to the action that has the highest value C according to the current estimate. However, if there are at least two actions with same highest values, we randomly select one of actions which have the highest value to reward. Then, the experience memory E will restart to record these values from zero. Note that the value C is negative if an action leads to worse the fitness values. To put it more precisely, we take an example showing in Fig. 4.6. As we have seen, there are five actions in state 1, each having an improvement value except action a_1. In this example, $C_{1,3}$ is the highest value comparing with others. As a result, action 3 will be rewarded. It is clear that case 3 and case 2 are conjoint, that is, every action in a given state would have its improvement value if we raise the possibilities to select the actions that are not applied. The benefits of this case are twofold. First, favorable selections will be kept and used to obtain promising neighboring solutions. Second, this case leads to select efficient actions, and, consequently, the agent will learn faster.

It is worth underling that the values of reward reflect the importance of the performance during the search process. After the application of the action a_j in the state s_i, the weight $w_{i,j}$ for the state s_i can be calculated by

$$w_{i,j} = w_{i,j} + \sum_{k=1}^{c} r_{ij}^{(k)} \tag{4.3}$$

where r_{ij} is the rewards after the application of the action a_j in the state s_i; c is the number of particular cases.

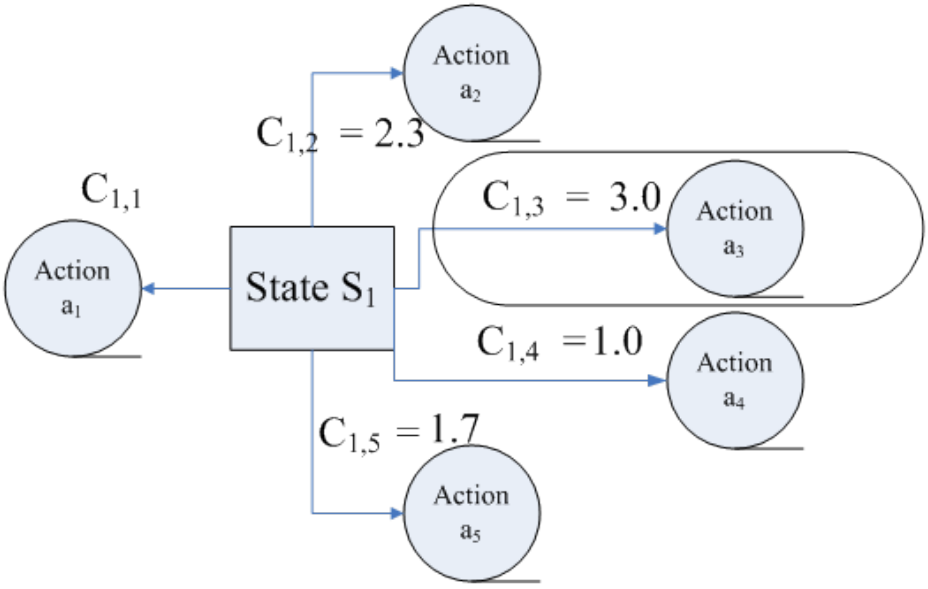

Figure 4.6: An example of the Case 3

6 Cooperative mimetism learning

In our study, each agent uses a reinforcement procedure to learn individually on the one hand. On the other hand, a cooperative mimetism learning is used to learn the behaviors of agent already enhanced by the individual learning. It may be impossible or undesirable for agents to learn all their knowledge all the time. In this sense, we first make the assumption that an agent which tends to behave as the most efficient agents can be found during the search. Under this assumption, the cooperative mimetism learning improves the search in the way that propagates the search behavior of this agent to others. Now, this rises two questions that need to be answered before accomplishing the cooperative search: $1)$ How to define the most efficient agent, $2)$ and which information to consider as the search behavior. Here, to facilitate, we consider an agent as more efficient than another one if it can improve the best coalition solution during the search. In our case, a weight matrix could be regarded as sort of the search behavior for agent, since it impacts the search paths. So far, we can apply the cooperative mimetism learning. Once the most efficient is found, the learning procedure is implemented as follows. The most efficient agent broadcasts its weight matrix to the other agents of the coalition. Then, when agents receive the weight matrix, they resume the search with a modified weight matrix. In fact, mimetism learning is firstly proposed by Yamaguchi et al. [1997] for the purpose of sharing learning results. In Meignan et al. [2009], the authors adapted this notion by giving an example of a successful cooperative algorithm design in the context of optimization. Following this meaningful work, the cooperative mimetism learning is performed using the same formula as stated in Meignan et al. [2009]. Let W_a be the weight matrix of the imitator agent A, and W_b the weight matrix of the imitated agent B. The imitation is computed as follows:

$$W_a = (1 - \rho) \cdot W_a + \rho \cdot W_b \tag{4.4}$$

with

W_a: weight matrix of the imitator agent;

W_b: weight matrix of the imitated agent;

ρ: mimetism rate.

As mentioned earlier, the cooperative search here is not based on a decomposition of problem domain. Rather, a set of independent agent executes concurrently in the search space. The cooperation focuses on the sharing information gathered by these individual agents. The information shared not only gives correct indications concerning the current status of the global search, but also enhances global behavior.

7 Low level heuristics

In this section, we introduce both the specialized operators and rules for the bus driver scheduling problem.

7 .1 Specialized operators

Starting from an initial solution generated by applying a fast construction operator called *Generation operator*, we apply a set of specialized operators to achieve a balance between diversification and intensification during the search process. Here, three diversification operators and six intensification operators are used by the agents. The set of diversification operators is composed of *Crossover operator*, *Shake operator* and *Perturbation operator*. Six intensification operators are as follows: *Cut and Add operator*, *Removal and Add operator version 1*, *Removal and Add operator version 2*, *Swap operator version 1*, *Swap operator version 2* and *Exchange operator*. The operators are depicted in Fig. 4.7 and we explain these operators in more detail below.

Generation operator: Initial solutions are obtained by this operator using a run-cutting method. Namely, the pieces of work of the vehicle blocks are successively covered by the created duties until the cover is complete. Indeed, the run-cutting method inspired by manual schedule procedures can be implemented in different ways. Here, *Generation operator* creates an initial feasible schedule by covering progressively uncovered pieces of work from a pool of vehicle blocks. During this process, a piece of work is covered by an assigned duty (or driver) if all the constraints are satisfied, and uncovered otherwise. The algorithm is repeated until all pieces of work have been covered.

Cut and Add operator: The neighborhood structure used in this operator is inspired by a heuristic called improvement heuristic in Lan et al. [2007]. If two solutions share at least one duty, these two solutions are called neighboring solutions. Formally, a solution

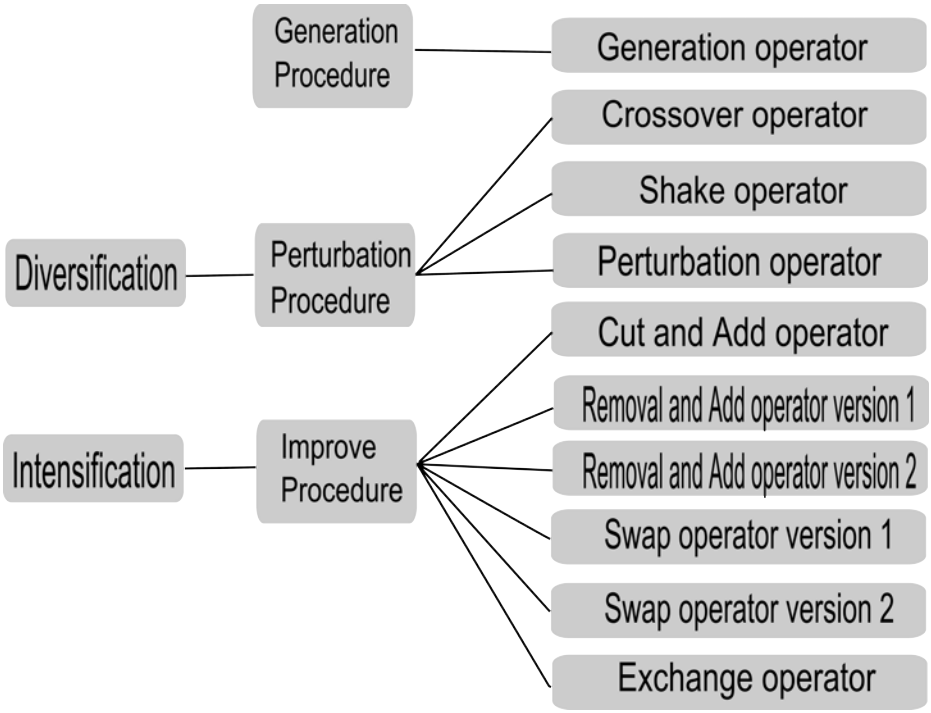

Figure 4.7: Specialized operators for the bus driver scheduling problem

$S = \{D_1, D_2, \ldots, D_l\}$ is a subset of selected duties. Each duty is an ordered set of tasks, $D_i = \{t_{i1}, t_{i2}, \ldots, t_{ik_i}\}, t_{ij} \in M$. Given two feasible solutions S and S', if $S \cap S' \neq \emptyset$, then S and S' are neighboring solutions, otherwise, they are disjointed solutions. The operator is performed as follows: A duty is selected from a given feasible solution, then we create a new solution by removing its pieces of work which can be added to other duties satisfying feasibility. Fig. 4.8 illustrates an example of *Cut and Add operator* performed on a feasible solution S. In this example, a feasible solution $S = \{D_1, D_2, D_3, D_4\}$ corresponds to a set of four duties, where each duty is a set of pieces of work. The piece of work 6 is taken out of the duty D_2, and the piece of work 14 is from the duty D_3. We attempt to add these two pieces into the other duties if feasibility is satisfied. Subsequently, these removed pieces are covered by D_4. Note that this operator is employed to reduce extra time and idle time, thereby reducing the total costs.

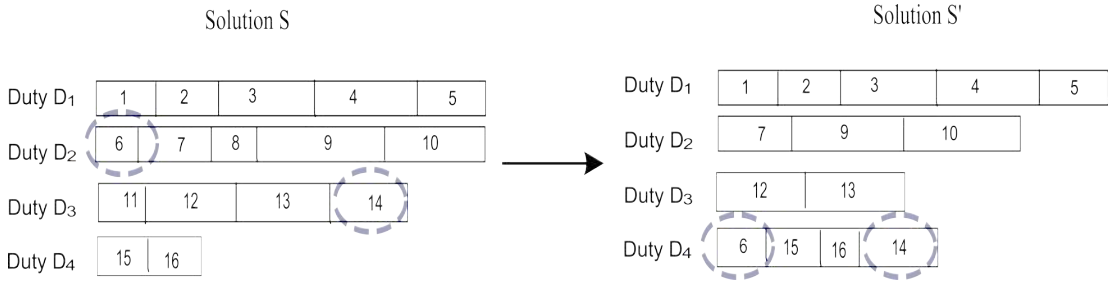

Figure 4.8: Cut and Add operator

Removal and Add operator version 1: This operator which follows the similar process

of the neighborhood search introduced by Lan et al. [2007] can be applied via a two-step procedure. Firstly, a number of duties that have the less working time are removed from a given feasible solution. The solution will thus become infeasible because there are some uncovered pieces of work. Next, this partial solution is made feasible in such way that we try to add these pieces of work into other duties if feasible, otherwise we construct new duties to cover the remaining pieces of work. To explain it more clearly, a simple example is shown in Fig. 4.9. Duty D_4 with pieces of work 15 and 16 in a feasible solution S can be found in this example. The solution S is transformed into a new solution S' by removing D_4, and adding all its pieces to the other duties. This operator is used in order to reduce the number of duties.

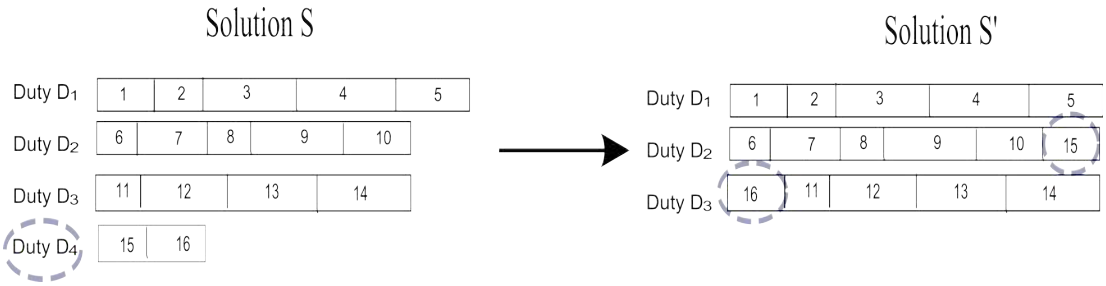

Figure 4.9: Remove and Add operator version 1

Removal and Add operator version 2: Behaves the same as *Removal and Add operator version 1*, except the selection of duties for removal. This operator evaluates the number of pieces of work for all the duties. Then a duty with the fewest number of pieces is selected to removal until the removal number of duties is satisfied. Next, we try to add their pieces to other duties as the same procedure as in *Removal and Add operator version 1*.

Swap operator version 1: This operator is based on the local search introduced by De Leone et al. [2011] which swaps two pieces of work in two duties. An example of description for this operator can be found in Fig. 4.10. Suppose that duty D_i covers pieces of work $\{t_{i1}, t_{i2}, t_{i3}, t_{i4}, t_{i5}\}$ and duty D_j covers pieces of work $\{t_{j6}, t_{j7}, t_{j8}, t_{j9}, t_{j10}\}$, D_i, $D_j \in S$. The decomposition of two duties can be applied to obtain four subsequences of piece of work. *Swap operator* is then performed to transform into two new duties D'_i and D'_j by swapping. As shown in this example, the new two duties have less extra time than before, therefore the total costs can be reduced. Note that this operator consists in finding the solutions with better costs, however, the number of duties cannot be reduced since the number of duties remain unchanged.

Swap operator version 2: The operator is as follow. A duty can be decomposed in partial consecutive duties, each having one or sequence of piece of work, respectively. Next, a similar decomposition can be applied to another duty. Then, a set of interchanges is performed among these partial duties while feasibility is still satisfied. This operator can be performed as shown in Fig. 4.11. In more detail, assuming that duty D_i which covers pieces of work $\{t_{i1}, t_{i2}, t_{i3}, t_{i4}, t_{i5}\}$ is decomposed as following partial parts: a sequence of

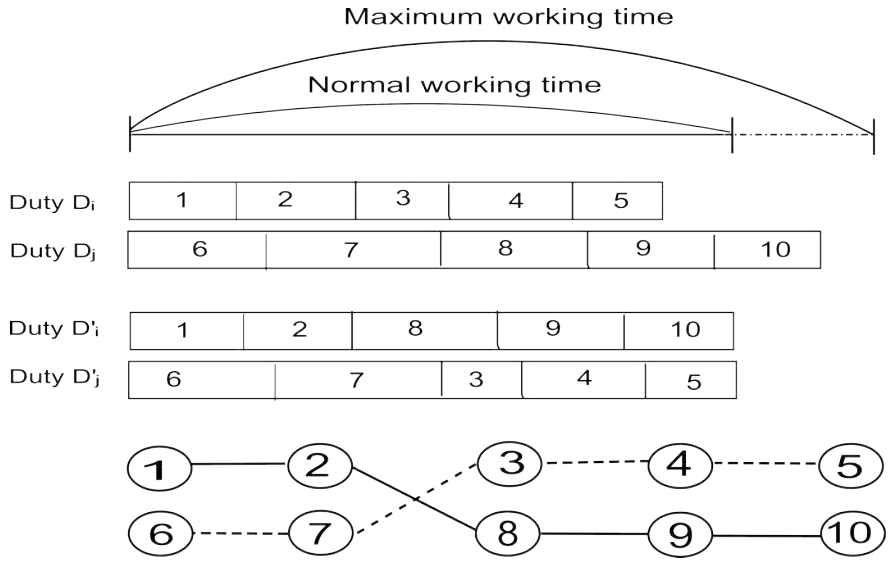

Figure 4.10: Swap operator version 1

pieces of work $\{t_{i1}, t_{i2}\}$, piece of work $\{t_{i3}\}$, piece of work $\{t_{i4}\}$ and piece of work $\{t_{i5}\}$. A similar decomposition can be applied to duty D_j to obtain the partial consecutive duties. Then, the two new duties D'_i and D'_j can be obtained after applying a set of interchanges. In this resulting example, we can observe that the costs for extra time are reduced. Note that this operator has the same purpose as Swap operator version 1, that is, it provides the possibility of constructing duties with lower cost.

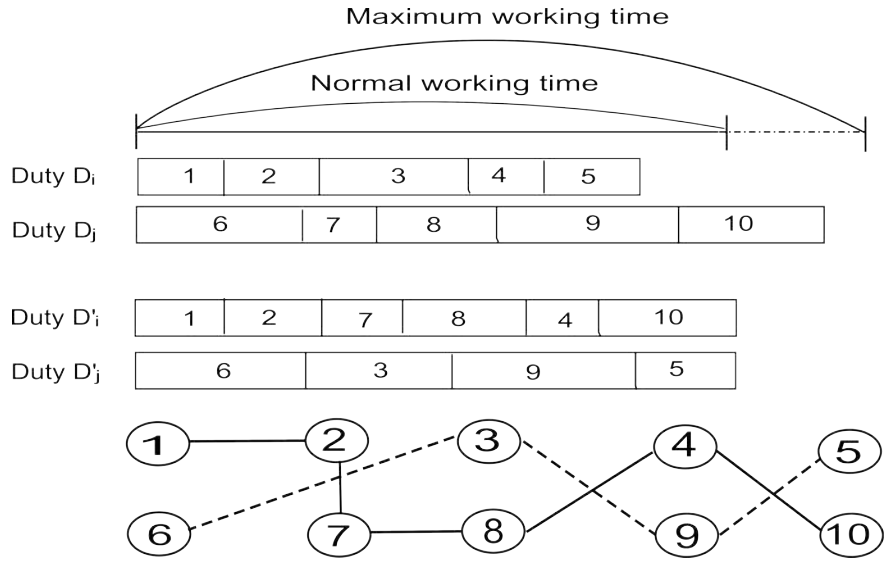

Figure 4.11: Swap operator version 2

Exchange operator: This operator consists in taking one task out from two duties and exchanges them while feasibility is still satisfied. Similar operator is used in Chew et al. [2001]. In Fig. 4.12 we present an example in which the pieces of work 3 and 8 are exchanged between two duties D_i, D_j. Thus, we obtain the two new duties D'_i and D'_j. Again, this operator is used in order to yield duties with lower cost.

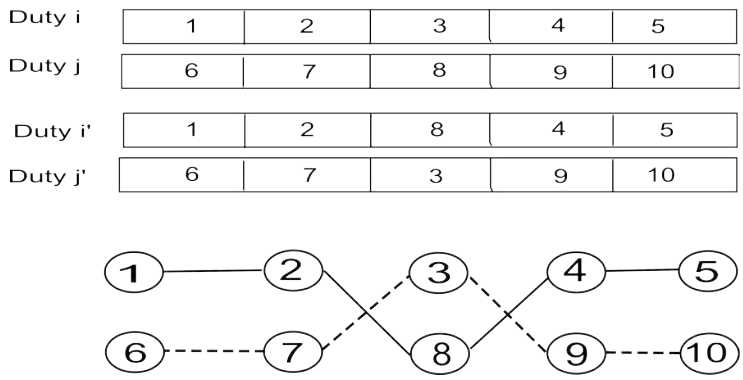

Figure 4.12: Exchange operator

Crossover operator: The solutions used for this operator are the current solution and the best found solution in an agent. This operator creates an offspring solution by inserting a duty from the fist solution in the other one. To obtain a valid solution without duplication of pieces of work, each pieces of work in the inserted duty is removed from other duties. More in detail, an example in Fig. 4.13 describes how to perform this operator. Given two feasible solutions S_a and S_b, each solution corresponds to a set of duties respectively. Then, *Crossover operator* is carried out by inserting Duty D_3 from S_b to S_a. Of course, some overlapping pieces are produced in this step. Therefore, in the next step all the pieces coved by D'_3 are removed from other duties D_1, D_3 and D_4 in S_a. This operator aims at giving a perturbation by choosing a solution in the neighborhood of the current best solution in order to produce a solution that maintains some good features of the current one.

Shake operator: This operator offers a new starting point which is not 'too far' from current one in order not to deteriorate too much the solution. It consists in exchanging tasks among a number of randomly selected duties if feasibility is satisfied.

Perturbation operator: The objective of this operator is to perturb the solution in order to provide a good starting point for the search. Behaves the same as *Shake operator*, but it performs in a way such that a duty tries to exchange its tasks with a neighboring one in a given solution. It is clear that this perturbation type has a strong diversification effect.

7 .2 Ruin and recreate procedure

This subsection describes the way to ruin and to recreate the current solution, continuing search from this new solution may allow to escape from a local optimum and to find better solutions. As mentioned before, we achieve the ruin and recreate principle in a special way by using a set of rules for the purpose of a strong perturbation when more diversification is needed. The following rules are used in this study:

Rule 1: A given feasible solution is destroyed by removing randomly a given number of duties. As a result, the feasible solution is transformed into a partial solution. Then, *Swap operator* is applied to this partial solution. Finally, the solution is rebuilt by satisfying all

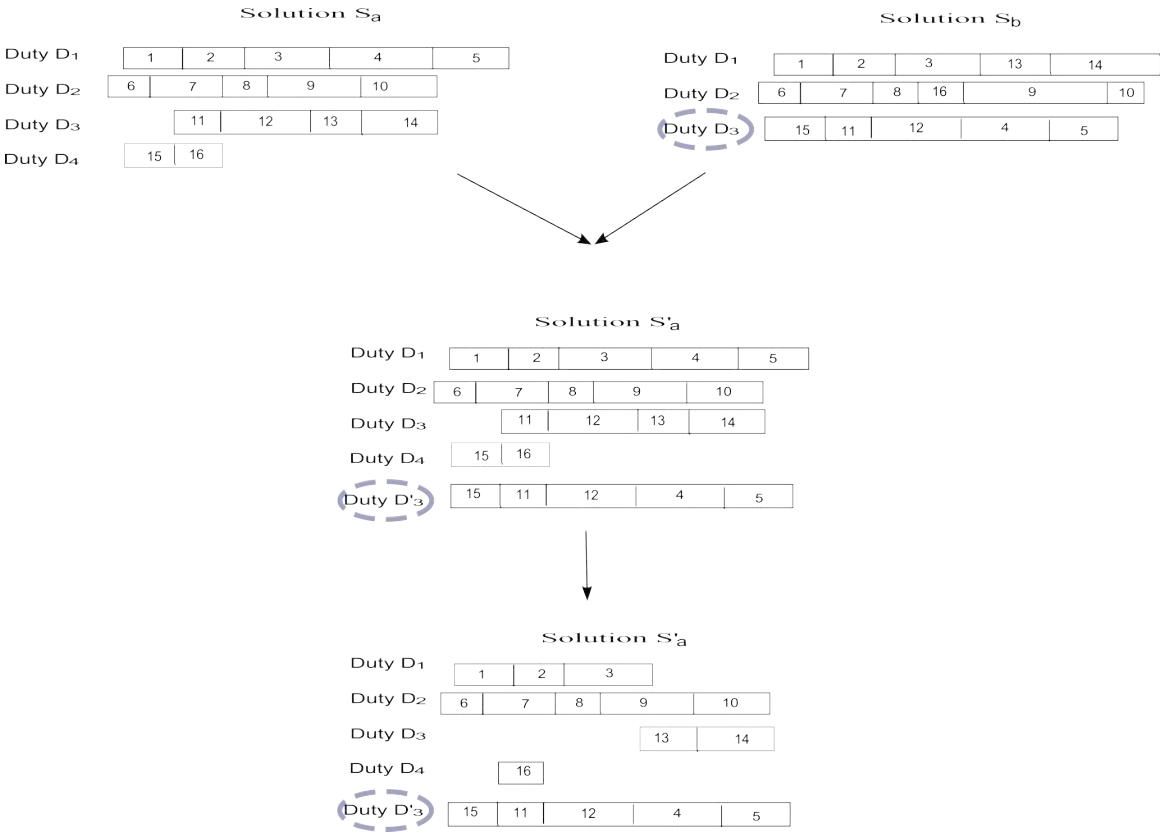

Figure 4.13: Crossover operator

the constraints until the solution is complete.

Rule 2: A given feasible solution is destroyed by removing randomly a given number of duties. As a result, the feasible solution is transformed into a partial solution. Then, *Exchange operator* is applied to this partial solution. Finally, the solution is rebuilt by satisfying all the constraints until the solution is complete.

Rule 3: A given feasible solution is destroyed by removing randomly a given number of duties. As a result, the feasible solution is transformed into a partial solution. Next, the behavior is same as *Cut and Add operator*, but removes the pieces of work from the duties with the total working time which is more than normal working time. Then, we try to add these pieces of work to the duties with the short total work time. Finally, the solution is rebuilt by satisfying all the constraints until the solution is complete.

Rule 4: A given feasible solution is destroyed by removing randomly a given number of duties. As a result, the feasible solution is transformed into a partial solution. Next, the behavior is same as *Cut and Add operator*, but removes the pieces of work from the duties with the total working time which is more than normal working time. Then, we try to add these pieces of work to a duty with the shortest total work time. Finally, the solution is rebuilt by satisfying all the constraints until the solution is complete.

8 Summary

Although we can find various methods dealing with bus driver scheduling in the literature, these algorithms are often specifically tailored to particular instances. They are not generally applicable to other similar problems (or even instances of the same problems). Hyper-heuristics represent a novel search methodology that addresses this issue. As far as we know, no works have been devoted to hyper-heuristics for the bus driver scheduling problem. This chapter presented a novel type hyper-heuristic approach, namely TPCH, which is based on a proposed pattern introduced in the previous chapter. Overall, TPCH is a parallel computing algorithm with the purpose of accelerating and broadening the search. It also benefits from this parallel scheme to hybrid multiple hyper-heuristics. The cooperative search consists in sharing the gathered information among concurrently executing hyper-heuristics. In fact, the cooperative search can be broadly done in two ways: centralized and decentralized manner. The centralized manner used is usually controlled by a central entity. Contrary to centralized one, there is no single agent with a global view of group activities when it is done in a distributed manner. Clearly, in this decentralized view each agent decides on its own actions based on its local knowledge and any other information it may obtain from the other agents. Our work focuses on the latter case, meaning that the agents are not controlled by any other agent, and they can communicate and interact directly with any other agent to achieve a goal. With regard to an individual hyper-heuristic, one of the main challenges is to be as general as possible on how to manage a set of low level heuristics with minimum parameter tuning. TPCH combines reinforcement learning and mimetism learning, in such an intelligent way, to adaptively choose promising the low level heuristics during the search process. The specialized operators in the low level can provide the intensification and diversification. Moreover, a set of rules allows to reach the regions of unfeasible solutions, instead of always maintaining solutions in the feasible region, and then to repair the feasibility of solutions. This extension enriches the possibilities of achieving a large 'jump' out of unpromising regions of the solution search space. In the next chapter, we will present the experimental design and analyses results. The comparative experiments will be performed on both the benchmark problems and the real world cases.

CHAPTER 5

COMPUTATIONAL EXPERIMENTS

1 Introduction

In this chapter, we study the influence of the role of the main algorithmic components, a comparative evaluation of the approach against some of heuristics used in the literature and the application to real-world problems as well. We first present the parameter setting of algorithm and the test instances used in our experiments. Then, we perform some evaluations and analysis of the main characteristics of the approach. Finally, we evaluate the approach on publicly available instances and compare its performance to a number of other heuristics.

All the tests are performed in a personal computer with Intel Core2(TM) 2 Duo CPU T5870 with a 2.00 GHz processor, 1.99 GHz with 2.00 Gb of RAM memory on Microsoft Windows 2002. The whole implementation was developed in the Java language using multithreads. In addition, the percentage deviation, denoted "%" in the following sections, over the best known solution to measure the quality of a solution is calculated as following:

$$Deviation = \frac{Result_{Average} - Result_{Bestknown}}{Result_{Bestknown}}\% \tag{5.1}$$

2 Parameter setting

The parameter setting is given in Table 5.1. These parameters were determined experimentally over a set of combinations, choosing the one that yield the best average output. Moreover, these parameters are kept constant for all the tests.

Table 5.1: Parameter setting of TPCH

Parameter	Description	Value
w	Initial weight value	1.0
p	Threshold percentage	0.03
ρ	Mimetism rate	0.3

Table 5.2: Size and the best known results of test case 1

Problem	No. of tasks	Best known results		
		No. of drivers	Total cost	CPU(sec.)
1	25	12	2371	0.10
2	50	20	2600	4.63
3	100	40	7395	1.88
4	250	81	8854	70.90
5	500	145	9716	6567.80

3 Test instances

We use the three test cases in our experiments, each has a set of instances. The test cases are described as follows.

Test case 1 provided by Mauri and Lorena [2007] is a set of artificial benchmark instances [1] based on a public transit company in Brazil. More details, this case contains five different instances. For each instance, Table 5.2 lists its number of tasks and the best known results so far. In accordance with Mauri and Lorena [2007] we use the same duty rules and cost functions as described in section 2.2. Note that it is a single depot case, where the normal working time is 8 hours and the maximum working time 10 hours.

Test case 2 is another set of generated benchmark instances [2]. A detailed description, characteristics, and the way of generating these data instances could be found by Huisman et al. [2005]. Briefly, the instances have been classified in two types (A and B) according to the travel speed. There are six sets of instances in each type, with four lines (A to B, A to C, A to D and B to C) in three sets and with five lines (adding a line from C to E in the above four lines). Hence, there are 5 relief locations in this case. Moreover, the sets differ in the number of trips and contain $80, 100, 160, 200, 320$ and 400 trips, respectively. Thus, there are twelve sets altogether. Each set consists of ten data instances. While Huisman et al. [2005] have analyzed five different duty types, the duty scheduling rules associated in these instances here are relatively simple and the same as in De Leone et al. [2011]. Namely, the feasible duty schedules we have generated are sets of pieces of work with no particular properties. The working time and maximum working time correspond 6.5 hours and 12 hours, respectively.

To our knowledge, there is not much real-life instances on which we can test our approach. Test case 3 involves a real-world instance, described in detail below, that was made available to us. In Chen and Niu [2012], the authors provide a subset of dataset from a bus service company in China. More details, the instance contains 168 trips (pieces of work) for one line of bus in a day. The maximum working time is 8 hours. The idle time between

[1] The instances are available, and can be downloaded, at `www.lac.inpe.br/~lorena/instancias.html`

[2] The instances are available, and can be downloaded, at `www.few.eur.nl/few/people/huisman/instances.htm`

two consecutive trips for one duty is considered as the costs. Note that the bus routes are circular (ring around the city) in this real-world instance. In other words, it has only one relief location.

4 Experiments with different strategies of learning and selection

Learning function and action selection are the core elements in reinforcement learning mechanism in the sense that it determines behavior of agent. Here, we evaluate the approach against some popular forms of reinforcement learning introduced by Sutton and Barto [1998]. To be precise, we consider four configurations using two strategies of learning and two strategies of selection.

4 .1 Strategies of learning

Q-learning [Watkins, 1989] is a type of learning function that does not need a model of its environment and can be used on-line policy. It works by estimating the value of state-action pair $Q(i, j)$, which can be viewed as the weight w_{ij} in our context. The key formula to update this value is as follows:

$$Q(i,j) \leftarrow Q(i,j) + \alpha[r_i + \gamma \max_a Q(j,a) - Q(i,j)] \tag{5.2}$$

where $Q(i, j)$ is the value of the state-action pair (s_i, a_j); α and γ are the learning rate and discount factor, respectively; r_i is the rewards as the result of executing the action a_j in the state s_i.

Algorithm 6 presents Q-learning. In this algorithm, it can be seen that the agent maintains a value of state and action representing a prediction of the worth of taking a particular action in a particular state.

The other learning function is SARSA [Sutton and Barto, 1998], which is very similar to the Q-learning except for the update rule of Q value being followed:

$$Q(i,j) \leftarrow Q(i,j) + \alpha[r_i + \gamma Q(j,k) - Q(i,j)] \tag{5.3}$$

where $Q(i, j)$ is the value of the state-action pair (s_i, a_j); α and γ are the learning rate and discount factor, respectively; r_i is the reward value as the result of tacking action a_j in state s_i; $Q(j, k)$ denotes the Q value of next state-action pair (s_j, a_k).

SARSA learning is presented in Algorithm 7. As we have seen, there are two action selection steps needed, for determining the next state-action pair along with the first. The major difference between SARSA learning and Q-learning is that the maximum reward for

Algorithm 6: Q-learning

/∗ Input ∗/
States s∈ S
Actions a ∈ A
/∗ Initialization ∗/
Set learning parameters α, γ
Initialize $Q(i, j)$
for *step k = 0, 1, 2,...* **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observe reward r, state s_j
    $Q(i, j) \leftarrow Q(i, j) + \alpha[r_i + \gamma \max_a Q(j, a) - Q(i, j)$
    $s_i \leftarrow s_j$
end

the next state is not necessarily used for updating the Q-values. Instead, action a_k is selected by using the same selection policy to obtain a new value of state-action pair $Q(j, k)$.

Algorithm 7: SARSA learning

/∗ Input ∗/
States s∈ S
Actions a ∈ A
/∗ Initialization ∗/
Set learning parameters α, γ
Initialize $Q(i, j)$
for *step k = 0, 1, 2,...* **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observe reward r, state s_j
    $Q(i, j) \leftarrow Q(i, j) + \alpha[r_i + \gamma Q(j, k) - Q(i, j)]$
    $s_i \leftarrow s_j$, $a_j \leftarrow a_k$
end

4 .2 Strategies of selection

We consider firstly ϵ-greedy [Sutton and Barto, 1998] as a selection strategy. In ϵ-greedy action selection, at each step the agent chooses a random exploratory action with probability ϵ, $0 < \epsilon < 1$, while with probability $1 - \epsilon$ the agent chooses the greedy action, that it takes whatever action seems best at the present moment, even when that decision might lead to bad long term consequences.

Softmax [Sutton and Barto, 1998] is the other selection strategy. Softmax policy is also based on the mentioned values $Q(i, j)$, favoring the actions with the highest values. On one hand, it ensures the selection of the best actions according to their $Q(i, j)$ value. On the other hand, a temperature parameter is used to decrease over time in order to obtain a good

balance between exploration and exploitation. In the state s_i, the action a_j is selected with the probability:

$$P(a_j \mid s_i) = \frac{e^{\frac{Q(i,j)}{\tau}}}{\sum_{k=1}^{m} e^{\frac{Q(i,k)}{\tau}}} \tag{5.4}$$

where m is the number of actions, and τ is the temperature parameter.

Table 5.3: Parameter setting of other reinforcement learning mechanisms

Parameter	Description	Value
α	Learning rate	0.5
γ	discount factor	0.5
τ	Temperature parameter	0.3
ϵ	Possibility	0.1

By considering the above strategies, we can combine four reinforcement learning mechanisms. The parameter setting of these mechanisms is given in Table 5.3. It should be pointed out that the rewards are given during the learning process in these mechanisms according to three particular cases described in Chapter 4. These mechanisms have been compared using the instances in the test case 1. In order to evaluate these performances, we consider that the mimetism condition (Algorithm 5, line 29) is never reached. For each instance, we have tested 10 times per instance using 10 agents in the coalition and 1000 iterations by agent. The results for each mechanism are presented in Table 5.4. The first column gives the instance number. Then the average deviations (denoted by $\%$) to the best known values and the computation times in seconds are reported. The results indicate that the mechanism used in TPCH performs better on average than other mechanisms in terms of quality of solution. Considering computation time, we observe that the differences are not noticeable, that is, all the mechanisms required approximately the same amount of CPU time. All these conclusions can be clearly seen in Fig. 5.1 and 5.2, showing an overall comparison of performance obtained by using the different learning mechanisms on the instances in the test case 1.

Table 5.4: Comparison of TPCH with different learning mechanisms on the instances in the test case 1

Problem	TPCH without mimetism			Q-learning with ϵ-greedy			Q-learning with Softmax			SARSA with ϵ-greedy			SARSA with Softmax		
	Drivers[a] (%)	Cost[b] (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)
1	-3.33	-5.60	3.46	-0.83	-2.24	3.41	-1.66	-3.62	3.33	-0.83	-2.31	3.40	-2.50	-4.43	3.26
2	0.00	0.00	3.65	0.00	0.00	3.71	0.00	0.00	3.40	0.00	0.00	3.49	0.00	0.00	3.69
3	-3.25	-2.12	10.67	-2.75	-1.73	10.07	-1.50	-1.86	9.36	-1.25	-0.39	9.38	-1.75	-0.72	14.44
4	0.00	0.37	64.73	0.00	-0.08	67.72	-0.37	-0.21	70.18	0.00	-0.01	57.93	0.00	0.20	90.87
5	0.00	2.37	246.81	0.00	3.09	303.73	0.00	2.85	289.58	0.00	3.01	311.23	0.27	2.52	299.79
Average	-1.32	-1.00	65.86	-0.72	-0.19	77.73	-0.71	-0.57	75.17	-0.42	0.06	77.09	-0.80	-0.47	82.41

[a] denotes the percentage deviation over the best known result in terms of driver number
[b] denotes the percentage deviation over the best known solution in terms of total cost

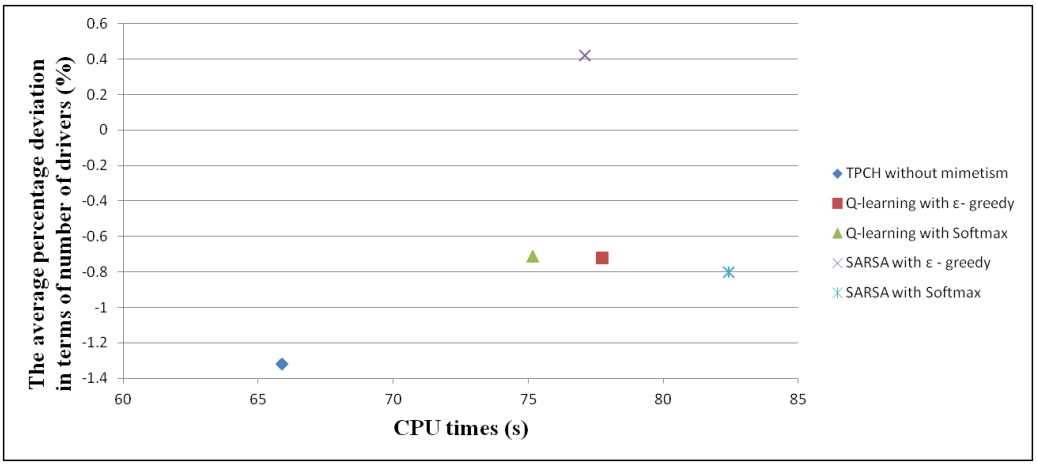

Figure 5.1: The average number of drivers obtained by using the different learning mechanisms on the instances in the test case 1

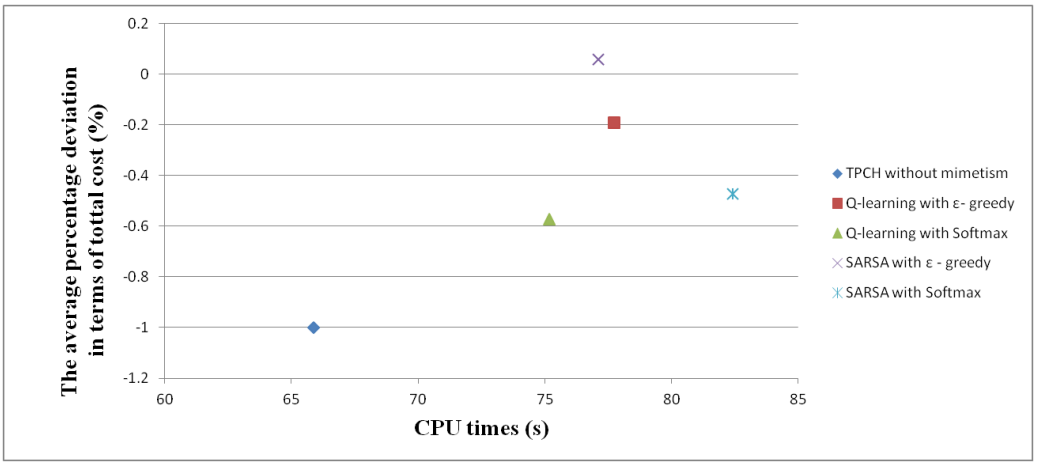

Figure 5.2: The average total cost obtained by using the different learning mechanisms on the instances in the test case 1

5 Performance with regard to coalition size

Starting with the parameter settings given in the previous section, but with the coalition size set to 10, it might be interested to know the relative coalition size for agents. In this section, we evaluate the performance of the different coalition size. The tests are done using the number 4 instance (250 tasks) in the test case 1.

See Fig. 5.3 for an example depicting the average total cost over 10 runs according to the computation time carried out for four coalition sizes from 50 to 1 agents. It can be seen that the computation time increases as the number of agents grows to obtain the same total cost at the beginning, but the coalition with more agents seems to have the advantage if the computation time still increases. We thus note from this figure that the

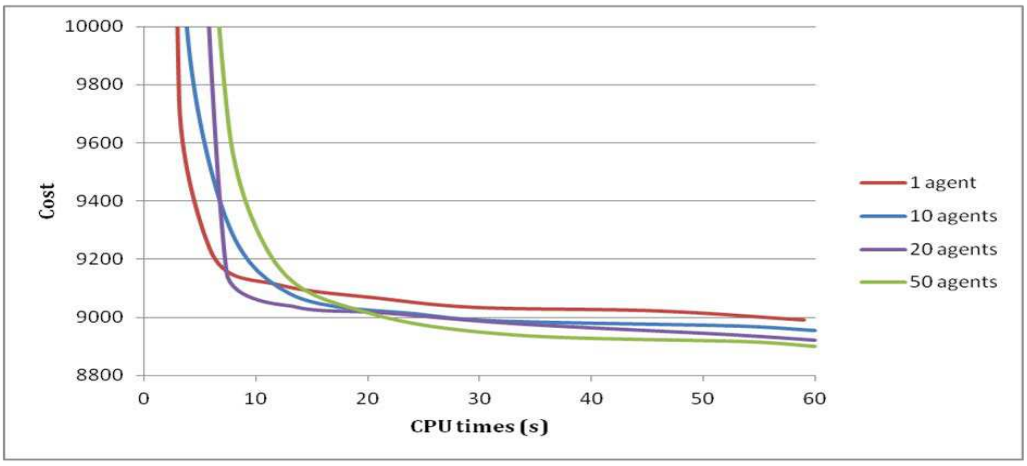

Figure 5.3: Average total cost according to the computation time

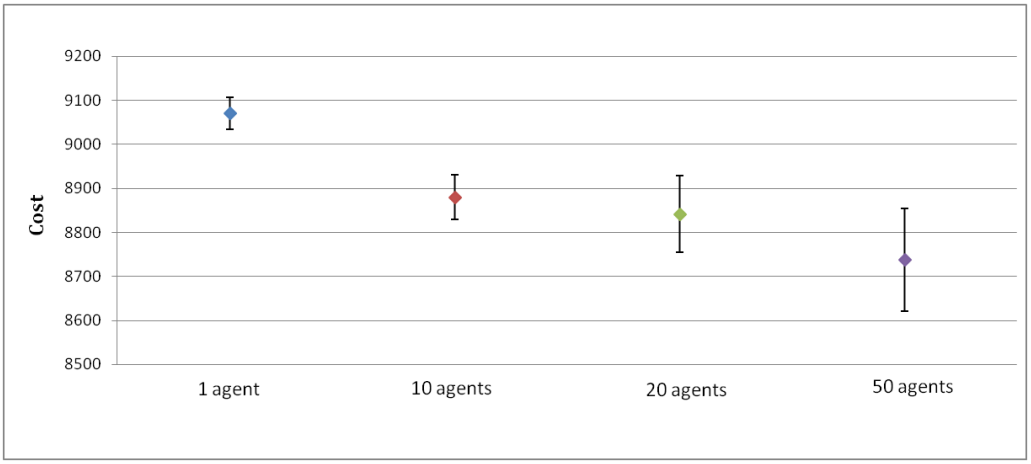

Figure 5.4: Performances of TPCH according to coalition size

size of 50 seems to be the best-performing case. However, the coalitions which are more than 10 number of agents become gradually with the computation time increased to an asymptotic point where more agents do not help much. A key point to notice is that in the case of one agent, i.e. without cooperation, its curve always lies above the curve of other cases after 20 seconds. To validate our observations, we present statistical analysis by calculating confidence intervals. Error bars are 95% confidence intervals for the mean. They are computed on the basis of standard deviations over 10 runs (2500 iterations per agent) as follows. Assuming that the sample mean $\widehat{\mu}$ and standard deviation $\widehat{\sigma}$ for $K = 10$ replicates are computed as

$$\widehat{\mu} = \frac{1}{K} \sum_{k=1}^{K} y_k, \tag{5.5}$$

where y_k, $k = 1, \ldots, 10$, are the mean fitness values in 95% confidence intervals, and

$$\widehat{\sigma} = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}(y_k - \widehat{\mu})}, \tag{5.6}$$

a confidence interval that would cover the true mean approximately 95% of the time is

$$\widehat{\sigma} \pm t_{K-1,.025}\frac{\widehat{\sigma}}{K}, \tag{5.7}$$

where $t_{K-1,.025}$ is the classic Student's t-statistic with $\alpha = .025$ and $K - 1$ degrees of freedom.

Moreover, we use the same parameters setting in statistic analysis proposed by Créput and Koukam [2008]. The confidence level is $\alpha = .025$ instead of $\alpha = .05$ because the confidence interval is symmetric about $\widehat{\sigma}$, so the total probability of including the true mean σ is .95, but the probability of missing σ on either side of $\widehat{\sigma}$ is .025. For $K = 10$ runs, $t_{K-1,.025} = 2.262$.

The results in 95% confidence intervals of Fig. 5.4 clearly show that a single agent is unable to compete with the multi-agent coalitions. Also, the case of 50 agents wins in all cases, but 10 agents is considered to be more adequate from the point of view of both solution times and achieved objectives.

6 Performance with regard to second phase and learning mechanisms

In this section, we now turn to an analysis of the main algorithmic components' influence for a fixed coalition size of 10 agents and 5000 iterations per agent when performing statistic analysis. Here again, all the tests are done using the number 4 instance (250 tasks) in the test case 1. We carried out these tests applying our approach without mimetism learning, with random selection, with simple reinforcement learning and without the second phase, respectively. As mentioned in the previous section, the condition to broadcast the weight matrix (Algorithm 5, line 29) is always considered to be false in order to remove mimetism. To apply random selection, we consider that both reinforcement mechanism (Algorithm 5, line 27 and 28) and mimetism learning are never applied. In the case of the simple reinforcement learning, the reward is given simply to the first particular case (see section 4.2). Namely, when an action results in a better solution than the best solution found, we update the weight of this action with a reward of 2. In the case of the approach without second phase, the parameter $p\%$ is set to a value greater than 1.

Fig. 5.5 plots the progress using the average total cost over 10 runs according to the number of iterations performed. As we see from this figure, for all configurations, the total

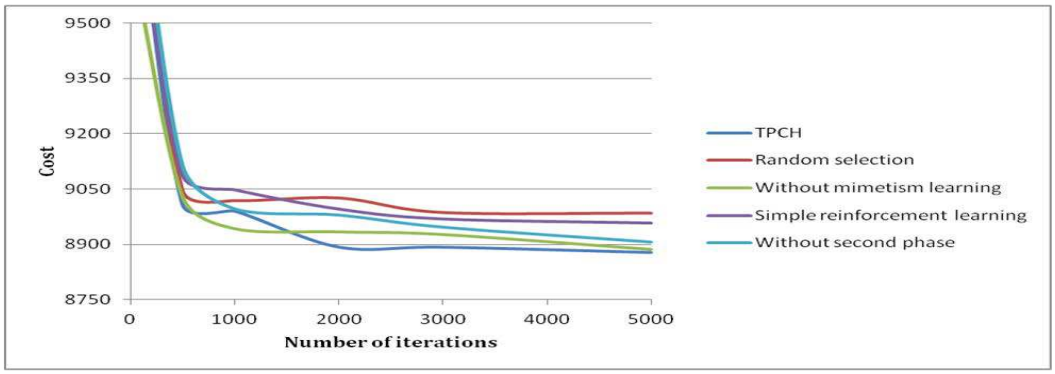

Figure 5.5: Average total cost according to the number of iterations

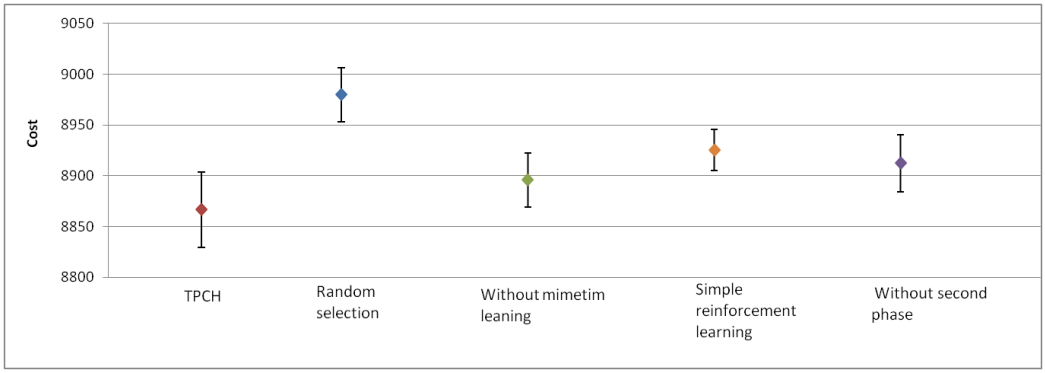

Figure 5.6: Performances of TPCH removing algorithmic components

cost decreases and we achieve better results as the number of iterations increases. However, it can be seen that with the same number of iterations carried out, the solutions tend to obtain toward overlapping target value at the beginning but the different quality solutions are yield as the number of iterations increases. The results of statistical analysis, with the same parameters setting in the previous section but 100 runs, given in Fig. 5.6 indicate statistically these differences.

With these evaluations shown by Fig. 5.5 and 5.6, we can conclude that the addition of second phase and the addition learning mechanisms show a good performance in terms of quality of the solutions found. With regard to the difference between approach with and without mimetism, the analysis showed that the use of mimetism learning improves the quality of solutions found by agents. Clearly, selection operators have the greatest influence on the algorithm performance. Thus, removing the learning mechanisms yield the relative poor quality solutions. Since little rewards given, the approach with the simple reinforcement learning is not sufficient to outperform TPCH. Finally, it seems that the approach without the second phase does not easily get stuck at a local optimum. As expected, this experimentation illustrates the positive impact of learning mechanisms used in our approach on the behavior of the agents.

7 Comparative study

Here, we perform a comparative evaluation of TPCH against some heuristics used in the literature for the bus driver scheduling problem. For each test case, we use the parameter setting presented in Table 5.1 as mentioned previously.

In Table 5.5, we give the computational results of different approaches on the test case 1. The deviations and the computation time required for TPCH correspond to the average values on 10 runs and a coalition size of 10 agents, each performs 5000 iterations. The average computation times in seconds are reported in columns headed by CPU in Table 5.5. All the other approaches proposed by Mauri and Lorena [2007] obtained the solutions on Intel Celeron processor of 2.0 GHz and 256 Mb of RAM memory. Comparing with the best known solutions, we can see that the reduce to 2.03% on average in terms of driver number and on average 2.66% saving on the total costs. The results indicate that our approach improve some best know solutions available so far for the test case 1. Furthermore, the average run time required is 280.20 seconds. Although we cannot compare directly the CPU time required, because the approaches were implemented using different computers and the authors give the best solution found over 5 runs, our approach is able to furnish the relatively good results from the point of view of quality of solutions achieved with the reasonable solution times.

Table 5.5: Computational results against the best known results for the test case 1

Problem	TPCH[a]			PTA/LP[b]			PTA/LP∗[c]			SA[d]			SA_20[e]		
	Drivers[f] (%)	Cost[g] (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)
1	-4.16	-6.26	6.21	0.00	0.00	0.10	0.00	0.00	0.30	0.00	0.00	1.90	0.00	0.00	35.48
2	-0.50	-0.96	12.93	0.00	0.00	4.73	0.00	0.00	41.64	0.00	2.07	42.87	0.00	0.00	949.04
3	-5.25	-5.31	46.09	0.00	0.00	1.57	0.00	0.00	4.98	0.00	0.00	7.60	0.00	0.00	173.26
4	-0.24	-0.63	375.46	0.00	0.00	71.34	1.23	4.40	229.68	4.93	15.17	199.86	2.46	11.04	3749.02
5	0.00	-0.14	960.32	0.00	0.00	6596.66	4.13	28.94	6717.30	5.51	32.37	7061.52	5.51	29.12	143565.20
Average	-2.03	-2.66	280.20	0.00	0.00	1334.88	1.07	6.67	1398.78	2.09	9.92	1462.75	1.59	8.03	29694.36

[a] Two-phase cooperative hyper-heuristic approach
[b] Population training algorithm/Linear programming with more iterations
[c] Population training algorithm/Linear programming with few iterations
[d] Simulated annealing
[e] Simulated annealing with 20 executions
[f] denotes the percentage deviation over the best known result in terms of driver number
[g] denotes the percentage deviation over the best known solution in terms of total cost

Compared with GRASP (Greedy Randomized Adaptive Search Procedure) in De Leone et al. [2011], Table 5.6 reports an overview of the results obtained for the the test case 2. All the numerical tests reported by De Leone et al. [2011] were carried out on a Pentium 4 with CPU 3.20 GHz and with 1.00 Gb RAM. In our experiment each test has been run 3 times for all the size of datasets. The coalition size is fixed for 10 agents and 1000 iterations by agent. We report the average number of drivers, and also the average computation time in seconds (denoted as CPU). Although our proposed approach seems to improve the best solutions for the sum of drivers, it should be emphasized that the comparisons between these results obtained cannot be made directly, because we cannot compare the total cost due to the two models defined differently. Regardless it is worth noting that our solutions obtained are faster in general, especially for larger cases.

Table 5.6: Comparative results in terms of driver number for the test case 2

Instance		TPCH[a]		GRASP[b]		
Problem	Type	Number of drivers	CPU (sec.)	Number of drivers	Time[c] (sec.)	Total Time[d] (sec.)
6	80A	17.3	5.3	20.3	17.0	33.7
7	100A	20.9	9.6	23.7	15.0	43.8
8	160A	27.5	29.1	30.7	53.6	117.7
9	200A	34.5	40.1	37.6	71.1	143.2
10	320A	48.4	121.4	53.6	157.2	366.9
11	400A	61.2	131.7	65.7	204.0	458.9
12	80B	19.8	5.8	22.2	14.9	29.8
13	100B	23.9	10.6	26.7	18.2	41.9
14	160B	33.1	32.2	37.1	45.1	104.0
15	200B	41.1	37.1	45.8	63.0	141.1
16	320B	59.9	105.1	67.1	127.9	325.6
17	400B	75.2	132.2	83.4	267.6	455.9
Average		38.5	55.0	42.83	87.8	188.54

[a] Two-phase cooperative hyper-heuristic approach
[b] Greedy Randomized Adaptive Search Procedure
[c] The mean time needed to find the better solution
[d] The mean time needed to run a total of 1000 iterations

We perform 10 runs per instance for the test case 3. All the runs are done with a coalition size of 10 agents and 5000 iterations by agent. Table 5.7 summarized that the average results on 10 runs for our TPCH solution are compared with those obtained by Chen and Niu [2012] using Tabu search. As well, the average computation in seconds is shown in this table, but CPU time required for the best know solution unfortunately is not reported by Chen and Niu [2012]. Our computational results show that there is on average 1.36% more expensive in terms of total cost but a reduction of 2.72 on the number of drivers when compared with the best known solution. As stated in Chen and Niu [2012], however, the driver number 11 provided by the bus company is used to generate an initial solution for

Tabu search. Then, its objective is only to minimize the total cost. On the contrary, we give priority to minimize the number of drivers since this is considered as the first objective for the most bus companies. Moreover, the best of these 10 runs can be considered as a new best solution since it saves 6.43% to the total cost compared with the best known solution, and gives the total number of 10 drivers (see Appendix).

Table 5.7: Computational results against the best known results for the test case 3

Instance		Best known		TPCH[a]			TS[b]		
Problem	No. of tasks	No. of drivers	Total cost	Drivers[c] (%)	Cost[d] (%)	CPU (sec.)	Drivers (%)	Cost (%)	CPU (sec.)
18	168	11	3512	-2.72	1.36	42.17	0.00	0.00	-

[a] Two-phase cooperative hyper-heuristic approach
[b] Tabu search
[c] denotes the relative percentage deviation over the best known result in terms of number of drivers
[d] denotes the relative percentage deviation over the best known solution in terms of total cost

8 A real-world case of a bus company in France

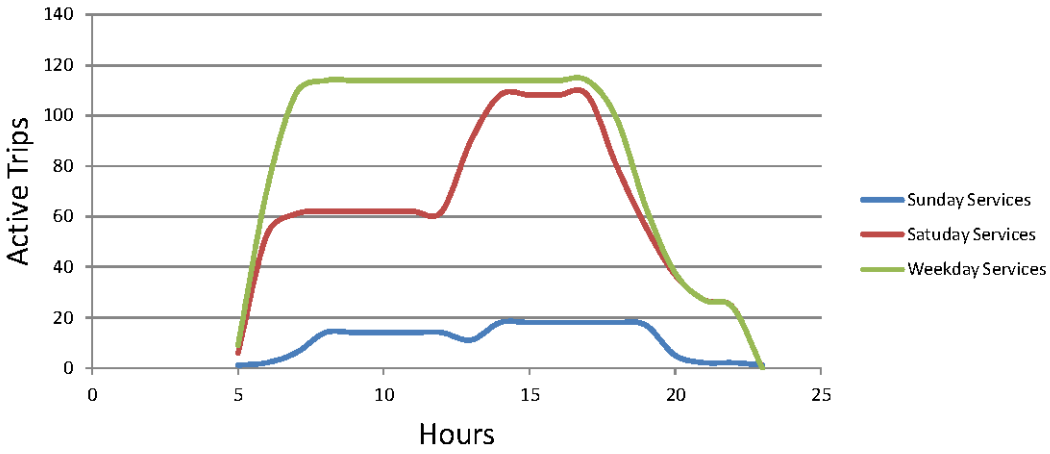

Figure 5.7: Distribution of trips along the Workday, Saturday and Sunday

In this section we apply our approach to a real-world large scale driver scheduling problem. This case involves a city bus service in Belfort in the east of France. All the data sets have been provided by the Optymo company. This company covers the whole bus service in Belfort, with over 300 drivers. For this real-life set of data all the information about the tasks was available, such as the start/finish time and start/finish location of each task, together with the bus lines.

Typically, in each city the frequency of service trips during a weekday is more than a weekend. A measure of the number of active trips along the Week day, Saturday, Sunday is shown in Fig. 5.7. This figure is constructed by the number of active trips at different time, which can reflect the different service levels satisfied.

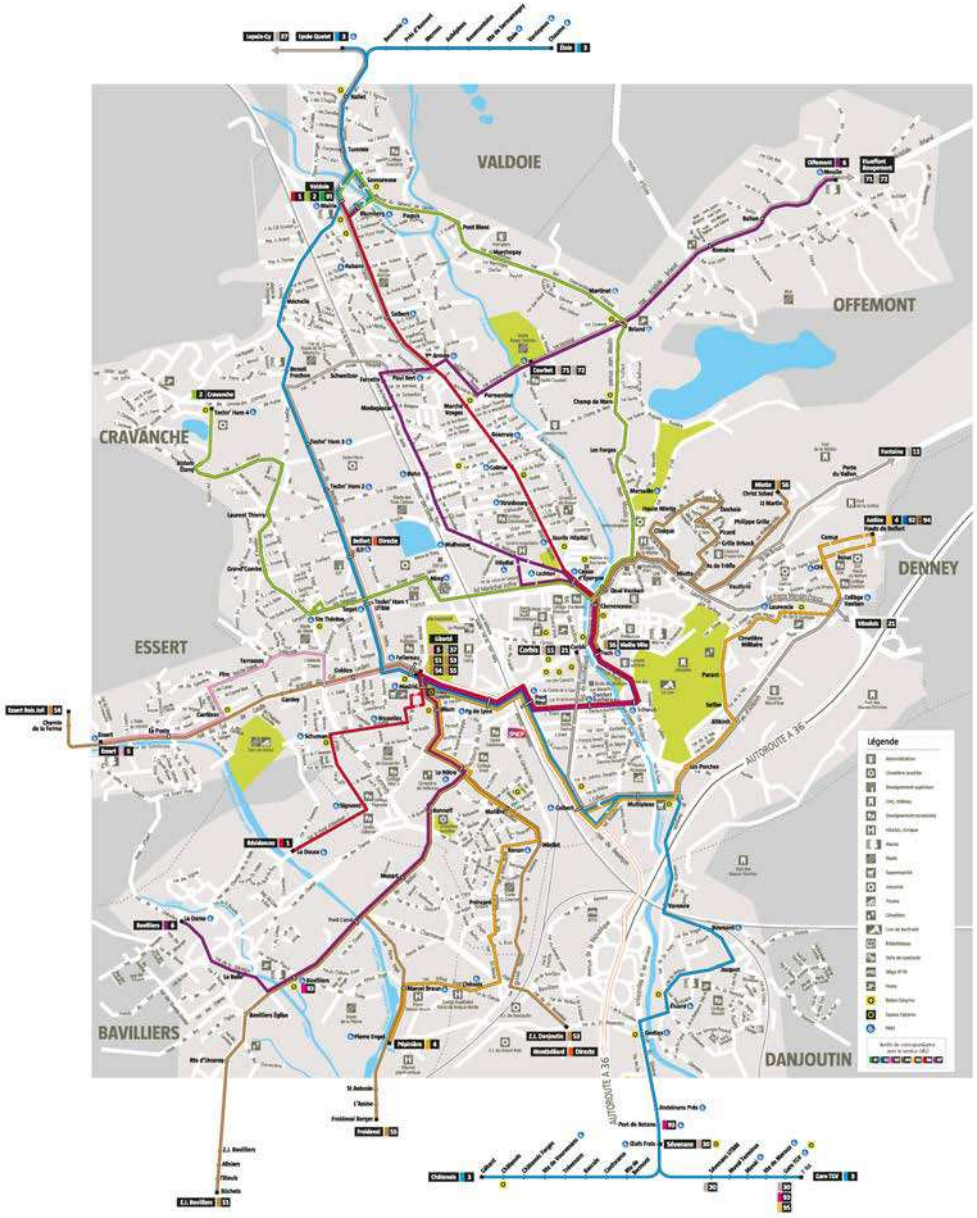

Figure 5.8: The bus services network in Belfort

Fig. 5.8 shows the schematic diagram of existing bus service network. In 2011, Optymo served more than 8.4 million passengers.[3] Therefore, good schedules enable not only to reduce operation costs, but also to achieve driver and passenger satisfaction. Table 5.8

[3] http://revue-transport-public.com/component/content/article/687-urbain/1740-nouvelle-etape-de-developpement-a-belfort

summarizes the dataset on which our approach was applied. More precisely, Optymo operates 6 lines covering the whole city bus service in Belfort. In fact, the number of tasks depends on the day of the week (weekday, Saturday and Sunday) and the season of the year (summer and winter schedules). Experiments to be reported are conducted on their winter schedule of 2012 including all the lines. The number of weekday services contains 1581 tasks with 18 relief locations, Saturday services includes 1176 tasks with 19 relief locations, and 207 tasks with 10 relief locations for Sunday services. Additional restriction from this operational environment impose that each driver entitles at least 30 minutes of break time. Moreover, the working time (589 minutes) and maximum working time (600 minutes) must be considered.

Table 5.8: General description of instances for the Optymo

Bus line	Weekday		Saturday		Sunday	
	No. of tasks	No. of relief locations	No. of tasks	No. of relief locations	No. of tasks	No. of relief locations
Line 1	329	3	243	3	93	3
Line 2	166	3	124	3	14	3
Line 3	342	3	225	3	36	3
Line 4	328	3	246	3	28	3
Line 5	93	3	93	3	12	3
Line 6	323	3	245	3	24	30

Here, the quality of the solutions is evaluated on the basis of the percent improvement to the number of drivers obtained by Optymo schedulers. Table 5.9 compares our average solutions in the ten runs with the solutions used in Optymo. It can be seen that the average improvements represent a reduction of 1.38% on the solutions produced by the Optymo planners. It is worth to notice that the tests performed on the real-world instances have demonstrated that our approach can obtain good results while consuming an acceptable amount of CPU time. The savings achieved may not seem large but in a typical operational environment, any savings will prove valuable since schedules are generally changed few times in one year. Moreover, our approach produce more efficient schedules, because manual solutions generally take at least a few days to prepare, and then the schedulers adjust them daily in order to improve some of their features [Dias et al., 2002].

Table 5.9: Computational results for the real-world instances provided by the Optymo

Instance			TPCH	
Problem	No. of tasks	Best known No. of drivers	Drivers (%)	CPU (sec.)
19	207	9	0.00	111.84
20	1176	56	2.32	1548.93
21	1581	77	-6.49	4653.93
Average			-1.38	2104.90

9 Summary

In this chapter, we analyzed the characteristics of TPCH in different dimensions. Firstly, learning is especially interesting in combination with hyper-heuristics to perform better choices during the search process. Therefore, we studied several popular learning mechanisms embedded TPCH to compare the performances. Observing from the results of the experiments, it can be found that the proposed reinforcement leaning mechanism performed better than others. Also, we studied the influence of the coalition size and the role of the main algorithmic components. The experimental results showed that we can obtain the satisfactory results when an overly large coalition size is employed, since a better chance of exploring the search space and discovering possible good solutions might occur in this situation. However, it is inevitable to suffer from an undesirable rise in computation time. Therefore, we determined an appropriate coalition size according to the experiments. In the last set of experiments we tested the influences of the main algorithmic components. The results can be the evidence in support of the fact that all of them play important roles in TPCH.

We performed a comparative evaluation of TPCH against some approaches found in the literature on a variety of artificial benchmark datasets. Most results reported in the

previous section indicate that our approach can produce better solutions when compared with other ones. Later in this chapter, the problem of the Optymo in Belfort city was analyzed. The solution obtained using our approach were presented and compared with the actual schedules in Optymo. The existing schedules of the problem are improved by 1.38% on average in terms of driver number. From the results on the test benchmark problems and especially from the results achieved by solving the Optymo operation problem, we can conclude that our approach is successful in solving the bus driver scheduling problem.

Conclusions

At the end of this thesis, we draw the conclusions and outline suggested further research directions. As it was stated earlier, the aim of scope of this thesis is to tackle the bus driver scheduling problem by proposing novel approaches. This chapter sums up not only the major developments from the investigation presented in this thesis, but also the attempts to explore the other new way of tracking bus driver scheduling problems, such as column generation heuristic. Therefore, before giving the achievements in this research and future work, we introduce briefly an abstract of a column generation heuristic since this approach is still in its infancy and needs more improvement.

The organization of this chapter is as follows. Section 1 gives a brief description of our heuristic approach within the context of column generation. In section 2, we present a general conclusion for the major achievements and contributions in this thesis. Section 3 suggests some potential future directions.

1 A column generation heuristic

Despite the successful application of column generation on crew scheduling and similar problems, attempts to improve column generation are still active due to the fact that column generation is known for its poor convergence. In our work, we seek to address the bus driver scheduling problem by using a column generation heuristic [Li et al., 2013]. A main distinctive feature of this work when compared with the existing approaches is to deal with the subproblem of column generation. That is, in contrast to the conventional column generation procedure, the subproblem here is solved using a heuristic search procedure inspired by some ideas from hyper-heuristics. We have tested the proposed approach by carrying out computational experiments on some instances. However, the improvements are needed to find the good results.

2 Achivements in this research

Heuristic and metaheuristic approaches tend to be knowledge rich, requiring substantial expertise in both the problem domain and appropriate heuristic techniques [Aickelin and Dowsland, 2000, Cowling et al., 2001b]. In this sense, most current methods are likely hard to adapt to new situation demands due to the fact that they are heavily dependent on specific

problem knowledge. As already introduced, hyper-heuristic methodologies are motivated by the goal of raising the level of generality. In other words, hyper-heuristics consist in removing the disadvantage of metaheuristic approaches and having a reusable, robust and fast-to-implement approach applicable to a wide range of problems and instances [Han et al., 2002]. Hence, there is considerable interest in the use of hyper-heuristics to solve the bus driver scheduling problem. Along this thesis, we have defined the multi-level hyper-heuristic pattern, and then proposed a novel approach based on this pattern for solving the bus driver scheduling problem.

Let us now briefly review the path that we have followed in this work.

Chapter 1 presented the bus driver scheduling problem and its importance for public transit providers. Then, the mathematical formulations are described. In this chapter, we also reviewed the three major types of research methods: heuristic methods, mathematical programming methods and metaheuristic methods. Moreover, we introduced some previous researches to present both success and limitations for bus driver scheduling.

In chapter 2 we reviewed the literature for related work on hyper-heuristics and identified reasons why hyper-heuristics offers promising perspectives for supporting heuristic development. In addition, we gave the main hyper-heuristic categories according to Burke et al. [2013] in which the emergence of hyper-heuristics has been deeply investigated.

Chapter 3 examined some recent metaheuristic and hyper-heuristic frameworks. Obviously, different frameworks analyze and design algorithms from different points of view. We summarized some common points and main advantage of these frameworks. However, we also discussed their limitations, particular in the context of cooperative hyper-heuristics. Based on the observations, we presented in this chapter a multi-level hyper-heuristic pattern as a blueprint to implement the cooperative hyper-heuristics. An important characteristic of this pattern is that it combines multiple hyper-heuristics such that the group intelligence can be more than the sum of the individual hyper-heuristics' performances. This pattern is concerned with the following aspects for the design and analysis of cooperative hyper-heuristics:

• Facilitating the design: The pattern consists of three abstract layers, each having its proper utility. We described main goals of each layer. The resulting architecture can be considered as a guideline of the design.

• Towards a multi-agent organizational theory: As an organizational model, all the critical roles are identified and defined. Therefore, it may allow the research into designing new and better approaches when using more powerful techniques to play these roles, and thereby offer a means for achieving further advances through the use of agent-based method.

• Identifying algorithmic components: In this pattern, we try to identify the algorithmic components underlying hyper-heuristics, such as the memory and learning mechanisms. These components may be useful for the implementation of algorithms.

Chapter 4 presented the usage of the proposed pattern by introducing a particular hyper-

heuristic approach, called TPCH. In this chapter, we detail how the hyper-heuristics take advantages of learning and cooperation mechanisms. Briefly speaking, TPCH adopts a decentralized approach which composes a set of agents. An individual which encapsulate a single solution are organized in a group called a coalition. In comparison to simple evolutionary algorithms, these agents have additional capacities of decision, learning and cooperation. In addition, the learning and cooperation capacities of agents favor the adaptation to various problem instances. The specialization of TPCH for the bus driver scheduling problem necessitates the definition of diversification and intensification heuristics. We combined here the different types of low level heuristics to achieve the intensification and diversification. To the best of our knowledge, there is not much research work on incorporating these types of low level heuristics simultaneously in a hyper-heuristic approach. Specifically, four types of low level heuristics are covered:

-Mutational heuristic (perturbation): performs a random modification of the solution.

-Neighborhood search heuristic (hill climbing): starts from a solution, and then searches a better one by incrementally changing a single element of this solution.

-Ruin and recreate heuristic (large neighborhood search): allows to partly destroy the solution and rebuilds it afterwards.

-Crossover heuristic (recombination): obtains a new solution in such a way that it combines solution components from two solutions.

We showed also, in chapter 5, that the proposed hyper-heuristic approach can successfully address the bus driver scheduling problem both in terms of solution quality and computational effort, particularly in regard to the large real-world instances.

3 Perspectives and future research

In this section, we detail the future research directions as follows.

• As like many other hyper-heuristics, TPCH has often involved one or more tuning parameters to find good parameter settings for a given problem. Therefore, a major limitation of TPCH is that many trial runs with different parameters are needed. Additionally, an interesting approach might use other leaning mechanisms for each agent to perform more efficiently. Finally, no theoretical analysis that guarantees the convergence of TPCH has been presented. Some of these limitations can be seen as rewarding paths for future work.

• Our efforts are focused in finding good quality and implementable solutions in reasonable computational time. Even though time consumed generally is satisfied when applying to the bus driver scheduling problem, a faster algorithm is still hoped to be accomplished. This may be achieved and be exploited by using the massive parallelism within multiprocessor architectures. These issues are currently under investigation in our research team and will be the subject of further research.

• A major challenge within column generation procedure is to solve its subproblem.

Our future research might also investigate an effective strategy which can overcome this challenge. It is reasonable to believe that future work on the hybridization column generation methods and heuristics might be highly promising.

• Finally, we would like to make some suggestions for future work in the field of crew scheduling. In the work reported here we focused on the bus driver scheduling problem. It might be thought that other crew scheduling problems, e.g., train crew scheduling, could be applied. This raises interesting research questions about how to solve variants of the similar problems with minimal changes to the approaches.

Résumé étendu en français

1 Contexte

Tous les fournisseurs de transport en commun à travers le monde sont confrontés à un même problème : l'affection des conducteurs aux véhicules ordonnances (l'habillage). Trois aspects principaux doivent être considérés pour ce problème : toutes les tâches doivent être entièrement affectées, toutes les fonctions doivent être réalisables et le nombre de conducteurs doit être minime. De par sa nature combinatoire, l'affection des conducteurs est un problème très complexe. Beaucoup d'efforts de recherche sur ce sujet ont été entrepris depuis l'année 1960. Cependant, il reste l'un des problèmes les plus difficiles dans le processus de planification des transports en commun [Zhao, 2006]. Ainsi, la recherche dans ce domaine toujours continue, et de nouvelles approches sont constamment proposées afin de résoudre ce problème.

Au cours des dernières années, la taille des données à prendre en compte pour le problème de l'habillage s'est vue considérablement augmenter en raison de deux facteurs : l'expansion rapide des lignes de bus dans les villes et l'augmentation du nombre de passagers dans les transports publics. Par conséquent, de nombreuses entreprises aujourd'hui n'ont pas besoin de rechercher l'optimalité, ni même des solutions proches de l'optimalité. Elles sont plus intéressées par d'assez bonnes solutions obtenues dans un délai raisonnable. Les heuristiques ont joué un rôle important dans cette situation. Même si nous pouvons lister différentes heuristiques pour faire face aux problèmes de l'habillage dans la littérature, ces méthodes sont adaptées à des conditions particulières. Celles-ci peuvent influer sur l'expression de la fonction objective et sur les contraintes de service, telles que le contrat union, la réglementation de l'entreprise, etc. Souvent, comme l'algorithme leur est dédié, il est très difficile de l'adapter et de l'appliquer à d'autres problèmes, voire à d'autres instances du même problème. Les metaheuristiques ont permis de faire face à cet inconvénient, mais celles employées dans la plupart des études restent trop dépendantes du problème original. La recherche sur les hyper-heuristiques est une tentative pour surmonter les dépendances des métaheuristiques : l'objectif est d'élever le niveau de généralité pour résoudre une série de problèmes. Dans cette thèse, nous concentrons notre attention sur une approche hyper-heuristique, qui présente notamment des avantages potentiels en termes de flexibilité, de modularité et de robustesse sur les heuristiques ou métaheuristiques existantes.

Un problème classique lors de la conception de metaheuristiques est la difficulté de

parvenir à un équilibre entre intensification et diversification. L'utilisation de modèles d'organisation encourage la conception de métaheuristiques par l'identification des composants communs [Meignan et al., 2008]. Par conséquent, nous nous sommes intéressés à l'utilisation de notions organisationnels destinés à soutenir la conception d'approches hyper-heuristiques.

Comme certains problèmes similaires, le problème de l'habillage peut également être résolu par programmation mathématique, et en particulier par programmation linéaire. Selon une revue de la littérature, la génération de colonnes semble être l'une des approches les plus efficaces pour ce type de problème [Desrochers and Soumis, 1989, Ernst et al., 2004, Lübbecke and Desrosiers, 2005]. Comme nous le savons, l'utilisation de méthodes mathématiques peut aider à obtenir des solutions optimales, mais la complexité de calcul croît de façon exponentielle avec la taille du problème. Au cours des dernières années, l'intérêt en combinaison de méthodes exactes et d'heuristiques a considérablement augmenté chez les chercheurs en optimisation combinatoire. Dans notre travail, nous tentons également d'explorer l'utilité et le potentiel de ces possibilités par la recherche d'une approche heuristique dans le cadre de la génération de colonnes. Il convient de souligner que cette approche en est encore à ses balbutiements et mériterait des recherches plus poussées.

2 Objectifs de nos travaux

L'objectif de cette thèse est de présenter une approche basée sur les agents, visant à résoudre le problème de l'habillage plus rapidement et efficacement. La démarche que nous adoptons pour atteindre cet objectif est la suivante. Tout d'abord, nous décrivons un modèle construit sur des concepts organisationnels afin de faciliter la conception d'hyper-heuristiques coopératives. Deuxièmement, nous développons une approche efficace et adaptative pour résoudre le problème de l'habillage en se basant sur le modèle proposé.

Un modèle hyper-heuristique multi-niveau basé sur le framework AMF *(Agent Metaheuristic Framework)*

Comme les hyper-heuristiques peuvent montrer des performances différentes au cours de la recherche, il est logique de tester si elles peuvent coopérer d'une telle façon de sorte qu'elles puissent échanger des informations utiles afin d'améliorer la capacité de l'exploration dans l'espace de recherche. Cependant, la conception de mécanismes de coopération et la détermination d'informations utiles à l'échange entre hyper-heuristiques reste un défi. Suivant la direction donnée par Meignan et al. [2008], un modèle d'organisation nommé modèle hyper-heuristique multi-niveau, dérivé du framework *AMF*, est proposé pour faciliter la conception d'hyper-heuristiques coopératives. En réalité, Meignan et al. [2008] ont proposé le framework *AMF* pour l'analyse des algorithmes existants, et ils encouragent la conception de nouvelles métaheuristiques. Une métaheuristique est définie

comme une organisation composée d'un ensemble de rôles qui interagissent afin de trouver une solution optimale. De cette manière, un modèle organisationnel de métaheuristiques peut être utilisé pour décrire à la fois des métaheuristiques basées sur la population et les méthodes de trajectoire. A notre connaissance, l'utilisation d'un modèle organisationnel dans le but d'analyser et de concevoir des hyper-heuristiques est une approche rare. Par conséquent, nous décrivons un modèle organisationnel dans un contexte d'hyperheuristiques coopératives. Le motif résultant vise à soutenir la conception d'une hyper-heuristique coopérative avec les caractéristiques souhaitées de souplesse, d'évolutivité et de généralité.

TPCH : Une approche hyper-heuristique multiagent

Pour valider le modèle proposé, nous développons une approche hyper-heuristique multi-agent nommée *TPCH* (*Two-phase cooperative hyper-heuristic approach*). A travers cette approche, nous présentons comment les hyper-heuristiques, que l'on considère comme des agents, peuvent être organisés d'après la métaphore de la coalition. Plus précisément, *TPCH* est un algorithme de calcul parallèle qui vise à accélérer et l'élargissement de la recherche. Le schéma de parallélisassion est mis en œuvre de manière à ce que la suppression ou l'ajout d'un agent ne perturbe pas le fonctionnement global du système. Par conséquent, *TPCH* a été conçu en considérant une hyper-heuristique parallèle avec une stratégie décentralisée, où plusieurs agents parcourent l'espace de recherche de manière indépendante. En outre, l'approche ne repose pas sur une quelconque information globale. De plus, chaque agent peut communiquer avec les autres en utilisant les informations qu'il reçoit d'autres agents. Cette communication peut se faire par l'échange d'informations sur les solutions trouvées et les comportements de recherche d'apprentissage. Dans l'ensemble, la coalition se compose de plusieurs agents qui explorent simultanément l'espace de recherche et qui coopèrent afin d'améliorer leurs capacités de recherche.

Les agents, qui combinent un ensemble d'heuristiques de bas niveau, appliquent certaines règles fondées sur des principes de l'intelligence artificielle lors de la recherche. Plus précisément, un agent améliore une solution candidate en choisissant et en appliquant de manière itérative des heuristiques, à partir d'un ensemble d'heuristiques de bas niveau, lorsqu'il s'agit de résoudre un problème donné. Chaque agent utilise individuellement un apprentissage par renforcement afin d'adapter la sélection d'une méthode (une heuristique) en fonction de l'efficacité de plusieurs méthodes potentielles. En pratique, cela est réalisé en mettant à jour une matrice de poids. Dans le même temps, l'apprentissage par mimétisme permet de partager les connaissances individuelles acquises par renforcement. Ces deux mécanismes d'apprentissage sont utilisés conjointement, ce qui permet à l'agent de sélectionner la meilleure heuristique de bas niveau à appliquer pendant le processus de recherche.

Une telle approche permet non seulement d'accélérer la convergence vers la meilleure solution, mais elle est également applicable uniformément sur divers ensembles d'instances

de problèmes, sans efforts excessifs. Pour illustrer cela, une application en est effectuée sur une variété de jeux de données, y compris sur des problèmes d'ordonnancement réels.

3 Structure de la thèse

Après un bref exposé de nos propositions, nous présentons l'organisation de cette thèse. Suite à l'analyse précédente, nous l'avons divisée en deux parties principales.

La première partie (chapitres 1 et 2) présente l'état de l'art concernant le problème de l'habillage et des hyper-heuristiques. Nous présentons d'abord le contexte opérationnel dans lequel le problème est résolu et deux modèles formels d'ordonnancement de chauffeurs utilisés dans notre travail. Ensuite, nous donnons un aperçu des travaux connexes sur les méthodes de résolution de problèmes, exactes et heuristiques (chapitre 1). Enfin, nous passons en revue la littérature sur les hyper-heuristiques (chapitre 2).

La deuxième partie (chapitres 3, 4 et 5) présente *TPCH*, qui est une approche hyper-heuristique parallèle basée sur l'apprentissage par renforcement. Brièvement, nous commençons par introduire le framework *AMF*. Sur la base de ce cadre, nous décrivons une organisation comme un modèle pour concevoir des approches qui visent à tirer parti des synergies entre les hyper-heuristiques (chapitre 3). *TPCH* est construite à partir de ce modèle. Nous donnons les détails de cette approche, et de son application au problème de l'habillage (chapitre 4). Enfin, nous étudions l'influence du rôle des principaux composants algorithmiques dans l'approche proposée, et appliquons celle-ci sur une variété de jeux de données de référence artificiels et de cas réels (chapitre 5).

Cette thèse se compose de cinq chapitres, et est organisée de la manière suivante :

Le chapitre 1 décrit le domaine du problème. Ce chapitre résume d'abord le contexte des problèmes d'ordonnancement de chauffeurs d'autobus. Ensuite, nous présentons les formulations des problèmes utilisés dans les approches que nous proposons. Enfin, nous listons les techniques d'optimisation qui sont appliquées aux problèmes de l'habillage.

Le chapitre 2 fait une synthèse de la littérature sur les hyper-heuristiques, et présente les façons dont le terme « hyper-heuristique » a été interprété et appliqué. Ce chapitre traite également des tendances prometteuses de la recherche actuelle sur ce sujet.

Le chapitre 3 reprend les premiers pas vers la conception d'une coopération entre les hyper-heuristiques dont nous introduisons les concepts organisationnels. Ensuite, la relation entre la théorie de l'organisation et l'hyper-heuristique coopérative est étudiée. Enfin, nous avons étendu les fonctionnalités et amélioré les performances du framework traditionnel des hyper-heuristiques. Le framework proposé est à plusieurs phases et plusieurs niveaux afin de tirer parti de la synergie de l'exécution de plusieurs hyper-heuristiques, suivant une structure de coalition dont le but est de soutenir et de faciliter une distribution du calcul robuste par un contrôle décentralisé.

Le chapitre 4 présente notre approche pour résoudre le problème de l'habillage. Les principes et les principaux composants en sont décrits dans ce chapitre. En effet, cette approche est construite à partir du framework donné dans le chapitre 3. Par conséquent, nous pouvons considérer notre démarche comme une étude de cas pour présenter comment les concepts de la théorie des organisations et des systèmes multi-agents peuvent contribuer à la conception de la recherche coopérative dans le contexte d'hyper-heuristiques.

Le chapitre 5 poursuit l'étude de l'approche proposée par le test sur différentes instances de problème. Les résultats expérimentaux montrent que notre approche surpasse certaines des heuristiques de la littérature dédiées au problème.

Appendix

A New Best Solution to the instance provided by Chen and Niu [2012]

The new best solution to the real problem provided by Chen and Niu [2012] is given as follows:

63-76-80-86-93-100-111-116-124-132-140-149-154-159-2-4-8-14-38-43-49-71-23,
61-6-11-68-82-89-121-129-136-143-158-164-19-33-44-54,
88-94-99-118-125-133-155-161-165-168-7-13-27-37-42-50-55-65,
72-126-137-144-151-166-5-21-29-36-48-53-62-79-85-91,
74-101-142-150-9-16-24-31-67-84-92-107-112-117-156-160,
97-148-163-1-12-18-25-32-39-45-57-70-77-105-115-122-130-167,
30-40-59-73-81-87-95-106-119-127-135-145-152-157-162-10-17-47,
52-58-69-75-96-102-109-114-123-131-138-146-20-26-34-41-46,
60-66-83-90-98-103-108-113-134-141-3-15-22-28-35-51,
56-64-78-104-110-120-128-139-147-153

Bibliography

Aickelin, U., Burke, E., and Li, J. (2009). An evolutionary squeaky wheel optimization approach to personnel scheduling. *IEEE Transactions on Evolutionary Computation*, 13(2):433–443.

Aickelin, U. and Dowsland, K. A. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3(3):139–153.

Alba, E. (2005). *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience.

A.Rodriguez, S. (2005). *From Analysis to Design of Holonic Multi-Agent Systems: A Framework, Methodological Guidelines and Applications*. PhD thesis, Université de Franche-Comté.

Bader-El-Den, M. and Poli, R. (2008). Generating SAT local-search heuristics using a GP hyper-heuristic framework. In Monmarché, N., Talbi, E.-G., Collet, P., Schoenauer, M., and Lutton, E., editors, *Artificial Evolution*, number 4926 in Lecture Notes in Computer Science, pages 37–49. Springer Berlin Heidelberg.

Bai, R. (2005). *An Investigation of Novel Approaches for Optimizing Retail Shelf Space Allocation*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham.

Bai, R., Burke, E. K., Gendreau, M., Kendall, G., and McCollum, B. (2007). Memory length in hyper-heuristics: An empirical study. In *IEEE Symposium on Computational Intelligence in Scheduling, 2007. SCIS '07*, pages 173–178. IEEE.

Bai, R. and Kendall, G. (2005). An investigation of automated planograms using a simulated annealing based hyper-heuristic. In Ibaraki, T., Nonobe, K., and Yagiura, M., editors, *Metaheuristics: Progress as Real Problem Solvers*, number 32 in Operations Research/Computer Science Interfaces Series, pages 87–108. Springer US.

Blais, J.-Y. and Rousseau, J.-M. (1988). Overview of HASTUS current and future versions. In Daduna, D. J. R. and Wren, A., editors, *Computer-Aided Transit Scheduling*, number 308 in Lecture Note in Economics Mathematical Systems, pages 175–187. Springer Berlin Heidelberg.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308.

Burke, E., Hart, E., Kendall, G., Newall, J., Hart, E., and Ross, P. (2003a). *Handbook of metaheuristics, Kluwer Academic, chap Hyper-Heuristics: An Emerging Direction in Modern Search Technology*. Springer.

Burke, E., Kendall, G., Landa Silva, D., O'Brien, R., and Soubeiga, E. (2005). An ant algorithm hyper-heuristic for the project presentation scheduling problem. In *The 2005 IEEE Congress on Evolutionary Computation, 2005*, volume 3, pages 2263–2270 Vol. 3.

Burke, E., Kendall, G., and Soubeiga, E. (2003b). A Tabu-Search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9:451–470.

Burke, E., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192.

Burke, E. K., Curtois, T., Hyde, M. R., Kendall, G., Ochoa, G., Petrovic, S., and Vazquez- Rodriguez, J. (2009a). Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *In Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA 09)*, pages 790–797.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010a). In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146, pages 449–468. Springer US, Boston, MA.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010b). A classification of hyper-heuristic approaches. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, number 146 in International Series in Operations Research & Management Science, pages 449–468. Springer US.

Burke, E. K., Hyde, M. R., and Kendall, G. (2006). Evolving bin packing heuristics with genetic programming. In Runarsson, T. P., Beyer, H.-G., Burke, E., Merelo-Guervós, J. J., Whitley, L. D., and Yao, X., editors, *Parallel Problem Solving from Nature - PPSN IX*, number 4193 in Lecture Notes in Computer Science, pages 860–869. Springer Berlin Heidelberg.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., and Woodward, J. R. (2009b). Exploring hyper-heuristic methodologies with genetic programming. In Mumford, C. L. and Jain, L. C., editors, *Computational Intelligence*, number 1 in Intelligent Systems Reference Library, pages 177–201. Springer Berlin Heidelberg.

Burke, E. K., Kendall, G., Mısır, M., and Özcan, E. (2012). Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1):73–90.

Busoniu, L., Babuska, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172.

Calégari, P., Coray, G., Hertz, A., Kobler, D., and Kuonen, P. (1999). A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5(2):145–158.

Cavique, L., Rego, C., and Themido, I. (1999). Subgraph ejection chains and tabu search for the crew scheduling problem. *The Journal of the Operational Research Society*, 50(6):608.

Chaib-Draa, B., Moulin, B., Mandiau, R., and Millot, P. (1992). Trends in distributed artificial intelligence. *Artificial Intelligence Review*, 6(1):35–66.

Chen, M. and Niu, H. (2012). Research on the scheduling problem of urban bus crew based on impartiality. *Procedia - Social and Behavioral Sciences*, 43:503–511.

Chen, P.-C., Kendall, G., and Berghe, G. (2007). An ant based hyper-heuristic for the travelling tournament problem. In *IEEE Symposium on Computational Intelligence in Scheduling, 2007. SCIS '07*, pages 19–26.

Chew, K.-L., Pang, J., Liu, Q., Ou, J., and Teo, C.-P. (2001). An optimization based approach to the train operator scheduling problem at singapore MRT. *Annals of Operations Research*, 108(1-4):111–122.

Cotta, C., Sevaux, M., and Sörensen, K. (2008). *Adaptive and multilevel metaheuristics*. Springer, Berlin.

Cowling, P., Kendall, G., and Soubeiga, E. (2001a). A hyperheuristic approach to scheduling a sales summit. In *Lecture Notes In Computer Science*, volume 2079, pages 176–190. Springer-Verlag.

Cowling, P., Kendall, G., and Soubeiga, E. (2001b). A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference, MIC 2001*, pages 127–131.

Cowling, P. and Soubeiga, E. (2000). Neighborhood structures for personnel scheduling: A summit meeting scheduling problem. In *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*. American Association for Artificial Intelligence.

Crainic, T. G. and Toulouse, M. (2003). Parallel strategies for meta-heuristics. In Glover, F. and Kochenberger, G. A., editors, *Handbook of Metaheuristics*, number 57 in International Series in Operations Research & Management Science, pages 475–513. Springer US.

Créput, J.-C. and Koukam, A. (2008). The memetic self-organizing map approach to the vehicle routing problem. *Soft Computing*, 12(11):1125–1141.

Daduna, J. R., Branco, I., and Paixão, J. M. P. (1995). *Computer-aided transit scheduling: proceedings of the Sixth International Workshop on Computer-aided Scheduling of Public Transport*. Springer.

Daduna, J. R. and Mojsilovic, M. (1988). Computer-aided vehicle and duty scheduling using the HOT programme system. In Daduna, D. J. R. and Wren, A., editors, *Computer-Aided Transit Scheduling*, number 308 in Lecture Note in Economics Mathematical Systems, pages 133–146. Springer Berlin Heidelberg.

Danoy, G., Bouvry, P., and Boissier, O. (2010). A multi-agent organizational framework for coevolutionary optimization. In Jensen, K., Donatelli, S., and Koutny, M., editors, *Transactions on Petri Nets and Other Models of Concurrency IV*, number 6550 in Lecture Notes in Computer Science, pages 199–224. Springer Berlin Heidelberg.

Dayan, P. and Watkins, C. J. (2006). Reinforcement learning: A computational perspective. In *Encyclopedia of Cognitive Science*. John Wiley & Sons, Ltd.

De Leone, R., Festa, P., and Marchitto, E. (2011). A bus driver scheduling problem: a new mathematical model and a GRASP approximate solution. *Journal of Heuristics*, 17(4):441–466.

Denzinger, J., Fuchs, M., and Fuchs, M. (1997). High performance ATP systems by combining several AI methods. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 102–107. Morgan Kaufmann.

Desrochers, M. and Rousseau, J.-M. (1992). *Computer-aided transit scheduling: proceedings of the Fifth International Workshop on Computer-aided Scheduling of Public Transport*. Springer.

Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13.

Dias, T. G., de Sousa, J. P., and Cunha, J. F. (2002). Genetic algorithms for the bus driver scheduling problem: A case study. *The Journal of the Operational Research Society*, 53(3):324–335.

Dréo, J., Aumasson, J.-P., Tfaili, W., and Siarry, P. (2007). Adaptive learning search, a new tool to help comprehending metaheuristics. *International Journal on Artificial Intelligence Tools*, 16:483–505.

El-Abd, M. and Kamel, M. (2005). A taxonomy of cooperative search algorithms. In Blesa, M. J., Blum, C., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics*, number 3636 in Lecture Notes in Computer Science, pages 32–41. Springer Berlin Heidelberg.

Ernst, A., Jiang, H., Krishnamoorthy, M., and Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27.

Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Professional.

Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 1998 International Conference on Multi Agent Systems*, pages 128–135.

Ferber, J., Gutknecht, O., and Michel, F. (2004). From agents to organizations: An organizational view of multi-agent systems. In Giorgini, P., Müller, J. P., and Odell, J., editors, *Agent-Oriented Software Engineering IV*, number 2935 in Lecture Notes in Computer Science, pages 214–230. Springer Berlin Heidelberg.

Fisher, H. and Thompson, G. (1963). *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*. Prentice-Hall international series in management. Prentice-Hall, Englewood Cliffs, N.J. :. Bibliography: p. 380-387.

Fogel, D., Fogel, L., and Atmar, J. (1991). Meta-evolutionary programming. In *Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers*, pages 540 –545 vol.1.

Fores, S. (1996). *Column Generation Approaches to Bus Driver Scheduling*. PhD thesis, University of Leeds.

Fores, S., Proll, L., and Wren, A. (2001). Experiences with a flexible driver scheduler. In Voß, S. and Daduna, J. R., editors, *Computer-Aided Scheduling of Public Transport*, number 505 in Lecture Notes in Economics and Mathematical Systems, pages 137–152. Springer Berlin Heidelberg.

Freling, R. (1997). *Models and Techniques for Integrating Vehicle and Crew scheduling*. PhD thesis, Erasmus University, Rotterdam, Nederland.

Garrido, P., Castro, C., and Monfroy, E. (2009). Towards a flexible and adaptable hyperheuristic approach for vrps. In Arabnia, H. R., de la Fuente, D., and Olivas, J. A., editors, *Proceedings of the 2009 International Conference on Artificial Intelligence, (ICAI 2009), July 13-16, 2009, Las Vegas Nevada, USA, 2 Volumes*, pages 311–317. CSREA Press.

Garrido, P. and Riff, M. (2010). DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16(6):795–834.

Gaud, N. (2007). *Systèmes multi-agents holoniques: de l'analyse à l'implantation. Méta-modèle, méthodologie, et simulation multi-niveaux*. PhD thesis, Université de Franche-Comté et Université de Technologie de Belfort-Montbéliard.

Glover, F. (1997). Tabu search and adaptive memory programming — advances, applications and challenges. In Barr, R. S., Helgason, R. V., and Kennington, J. L., editors, *Interfaces in Computer Science and Operations Research*, number 7 in Operations Research/Computer Science Interfaces Series, pages 1–75. Springer US.

Glover, F. and Laguna, M. (1998). *Tabu search*. Kluwer, Boston, Mass.

Gratch, J. and Chien, S. (1993). Learning search control knowledge for the deep space network scheduling problem. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign.

Gruer, P., Hilaire, V., Koukam, A., and Cetnarowicz, K. (2002). A formal framework for multi-agent systems analysis and design. *Expert Systems with Applications*, 23(4):349–355.

Han, L., Kendall, G., and Cowling, P. (2002). An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In *Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution And Learning, (SEAL'02), Orchid Country Club, Singapore, 18-22 Nov 2002*, pages 267–271.

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000). MOISE: an organizational model for multi-agent systems. In Monard, M. C. and Sichman, J. S., editors, *Advances in Artificial Intelligence*, number 1952 in Lecture Notes in Computer Science, pages 156–165. Springer Berlin Heidelberg.

Hayes-Roth, B. (1995). An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1–2):329–365.

Hickman, M. D., Mirchandani, P. B., and Voss, S. (2008). *Computer-aided systems in public transport*. Springer.

Hilaire, V. (2000). *Vers une approche de spécification, de prototypage et de vérification de systèmes multi-agents*. PhD thesis, Université de Franche-Comté.

Hilaire, V., Koukam, A., Gruer, P., and Müller, J.-P. (2000). Formal specification and prototyping of multi-agent systems. In Omicini, A., Tolksdorf, R., and Zambonelli, F., editors, *Engineering Societies in the Agents World*, number 1972 in Lecture Notes in Computer Science, pages 114–127. Springer Berlin Heidelberg.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press, Cambridge, Mass.

Huisman, D., Freling, R., and Wagelmans, A. P. M. (2005). Multiple-Depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502.

Ieumwananonthachai, A. and Wah, B. W. (1995). TEACHER – an automated system for learning knowledge-lean heuristics. Technical report, Center for Reliable and High Performance Computing, Coordinated Science Laboratory, Univ. of Illinois.

Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

Joslin, D. E. and Clements, D. P. (1999). "Squeaky Wheel" optimization. *Journal of Articial Intelligence Research*, 10(1):353–373.

Kazemi, A., Zarandi, M. H. F., and Husseini, S. M. M. (2009). A multi-agent system to solve the production–distribution planning problem for a supply chain: a genetic algorithm approach. *The International Journal of Advanced Manufacturing Technology*, 44(1-2):180–193.

Keller, R. and Poli, R. (2007). Linear genetic programming of parsimonious metaheuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4508–4515.

Keller, R. E. and Poli, R. (2008). Cost-benefit investigation of a genetic-programming hyperheuristic. In Monmarché, N., Talbi, E.-G., Collet, P., Schoenauer, M., and Lutton, E., editors, *Artificial Evolution*, number 4926 in Lecture Notes in Computer Science, pages 13–24. Springer Berlin Heidelberg.

Kendall, G. and Hussin, N. (2005). A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology. In *Practice and Theory of Automated Timetabling V*, pages 270–293.

Kendall, G. and Mohamad, M. (2004). Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proceedings of the 12th IEEE International Conference on Networks (ICON 2004)*, volume 2, pages 769–773 vol.2.

Kiraz, B., Ş Etaner-Uyar, A., and Özcan, E. (2013). Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Koza, J. R. (1999). *Genetic programming III: darwinian invention and problem solving*. Morgan Kaufmann, San Francisco.

Kwan, A., Kwan, R. S. K., Parker, M. E., and Wren, A. (1996). Producing train driver shifts by computer. In *Computer in Railways V, Vol.1: Railway systems managment*, pages 421–435. Computational Mechanics Publications.

Kwan, R. and Kwan, A. (2007). Effective search space control for large and/or complex driver scheduling problems. *Annals of Operations Research*, 155(1):417–435.

Lan, G., DePuy, G. W., and Whitehouse, G. E. (2007). An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, 176(3):1387–1403.

Leitão, P., Barbosa, J., and Trentesaux, D. (2012). Bio-inspired multi-agent systems for reconfigurable manufacturing systems. *Engineering Applications of Artificial Intelligence*, 25(5):934–944.

Leung, J. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC.

Li, J. (2005). A self-adjusting algorithm for driver scheduling. *Journal of Heuristics*, 11(4):351–367.

Li, J. and Kwan, R. S. (2003). A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research*, 147(2):334–344.

Li, S., Hassani, A. H. E., Créput, J., and Koukam, A. (2013). Heuristique de génération de colonnes pour l'habillage dans les systèmes de transport en commun. In *Actes du 14ème congrés national de la société française de recherche opérationnelle, ROADEF'2013*.

Lo, H. K. (2009). Proceedings of the 11th international conference on advanced systems for public transport. Hong Kong Society of Transportation Studies Limited.

López-Camacho, E., Terashima-Marín, H., and Ross, P. (2011). A hyper-heuristic for solving one and two-dimensional bin packing problems. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO'11)*, pages 257–258, New York, NY, USA. ACM.

Lourenco, H. R., Paixao, J. P., and Portugal, R. (2001). Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331 –343.

Lübbecke, M. E. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53(6):1007–1023.

Maes, P. (1995). Artificial life meets entertainment: lifelike autonomous agents. *Communications of the ACM*, 38(11):108–114.

Manington, B. and Wren, A. (1975). A general computer method for bus crew scheduling. In *Proceedings of International Workshop on Urban Passenger Vehicle and Crew Scheduling*. Chicago.

Mauri, G. R. and Lorena, L. A. N. (2007). A new hybrid heuristic for driver scheduling. *International Journal of Hybrid Intelligent Systems*, 4(1):39–47.

Meignan, D. (2008). *Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques application aux problèmes d'optimisation de réseaux de transports*. PhD thesis, Université de Technologie de Belfort-Montbéliard.

Meignan, D., Créput, J.-C., and Koukam, A. (2008). An organizational view of metaheuristics. In Jennings, N. R., Rogers, A., Petcu, A., and Ramchurn, S. D., editors, *Proceedings of the First International Workshop on Optimisation in Multi-Agent Systems, AAMAS'08*, pages 77–85.

Meignan, D., Koukam, A., and Créput, J. (2009). Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16:859–879.

Meilton, M. (2001). Selecting and implementing a computer aided scheduling system for a large bus company. In Voss, S. and Daduna, J. R., editors, *Computer-Aided Scheduling of Public Transport*, volume 505, pages 203–214. Springer Berlin Heidelberg, Berlin, Heidelberg.

Milano, M. and Roli, A. (2004). MAGMA: a multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(2):925–941.

Minton, S. (1996). Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1-2):7–43.

Misir, M., Verbeeck, K., De Causmaecker, P., and Berghe, G. V. (2010). Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8. IEEE.

Mitra, G. and Welsh, A. P. G. (1981). A computer based crew scheduling system using amathematical programming approach. In *Proceedings of the International Conference on Computer Scheduling of Public Transport*, pages 281–296. Amsterdam The Netherlands: North Holland.

Nareyek, A. (2004). Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer Decision-Making*, number 86 in Applied Optimization, pages 523–544. Springer US.

Ochoa, G., Qu, R., and Burke, E. K. (2009a). Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation (GECCO '09)*, pages 341–348, New York, NY, USA. ACM.

Ochoa, G., Vázquez-rodríguez, J. A., Petrovic, S., and Burke, E. (2009b). Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 1873–1880.

Ouelhadj, D. and Petrovic, S. (2008). A cooperative distributed Hyper-Heuristic framework for scheduling. In *IEEE International Conference on Systems, Man and Cybernetics, 2008. SMC 2008*, pages 2560–2565. IEEE.

Ouelhadj, D. and Petrovic, S. (2009). A cooperative hyper-heuristic search framework. *Journal of Heuristics*, 16:835–857.

Özcan, E., Bilgin, B., and Korkmaz, E. E. (2008). A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.*, 12(1):3–23.

Özcan, E., Misir, M., and Burke, E. K. (2009). A self-organising hyper-heuristi framework. In *In Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA 09)*, pages 790–797.

Özcan, E., Misir, M., Ochoa, G., and Burke, E. K. (2010). A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing*, 1(1):39–59.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Parker, M. E. and Smith, B. M. (1981). Two approaches to computer crew scheduling computer scheduling of public transport. In *Proceedings of the International Conference on Computer Scheduling of Public Transport*, pages 193–222. Amsterdam The Netherlands: North Holland.

Rechenberg, I. (1971). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin.

Rechenberg, I. and Schwefel, H. P. (1974). *Adaptive Mechanismen in der Biologischen Evolution und ihr Einfluss auf die Evolutionsgeschwindigkeit: Arbeitsbericht*.

Ross, P. (2005). Hyper-heuristics. In Burke, E. K. and Kendall, G., editors, *Search Methodologies*, pages 529–556. Springer US.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Ryan, D. M. and Foster, B. A. (1981). An integer programming approach to scheduling. In *Proceedings of the International Conference on Computer Scheduling of Public Transport*, pages 269–280. Amsterdam The Netherlands: North Holland.

Shen, Y. and Kwan, R. S. (2001). Tabu search for driver scheduling. In *Computer-Aided Scheduling of Public Transport*, volume 505, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg.

Shepardson, F. (1981). *Modeling the bus crew scheduling problem*. Amsterdam The Netherlands: North Holland.

Sim, K., Hart, E., and Paechter, B. (2012). A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492, pages 348–357. Springer Berlin Heidelberg, Berlin, Heidelberg.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (1998). Convergence results for single-step on-policy reinforcement-learning algorithms. In *MACHINE LEARNING*, pages 287–308.

Smith, D. C., Cypher, A., and Spohrer, J. (1994). KidSim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67.

Soubeiga, E. (2003). *Development and Application of Hyper-heuristics to Personnel Scheduling*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham.

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Taillard, Gambardella, L. M., Gendreau, M., and Potvin, J. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal Of Operational Research*, 135(1):1–16.

Tanese, R. (1989). Distributed genetic algorithms. In *International Conference on Genetic Algorithms*, pages 434–439.

Vázquez-Rodríguez, J. and Petrovic, S. (2010). A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793.

Voss, S. and Daduna, J. R. (2001). *Computer-aided scheduling of public transport*. Springer.

Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England.

Weiss, G., editor (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Wren, A. (1981). Proceedings of the international conference on computer scheduling of public transport. Amsterdam The Netherlands: North Holland.

Wren, A. and Chamberlain, M. (1988). The development of micro-BUSMAN: scheduling on micro-computers. In Daduna, D. J. R. and Wren, A., editors, *Computer-Aided Transit Scheduling*, number 308 in Lecture Note in Economics Mathematical Systems, pages 160–174. Springer Berlin Heidelberg.

Wren, A. and Rousseau, J. (1995). *Bus Driver Scheduling - An Overview*. Berlin, Germany: Springer-Verlag.

Wren, A. and Smith, B. M. (1988). Experiences with a crew schduling system based on set covering. From the book Computer-aided transit scheduling.

Yamaguchi, T., Tanaka, Y., and Yachida, M. (1997). Speed up reinforcement learning between two agents with adaptive mimetism. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97)*, volume 2, pages 594–600 vol.2. IEEE.

Zhao, B., Guo, C., and Cao, Y. (2005). A multiagent-based particle swarm optimization approach for optimal reactive power dispatch. *IEEE Transactions on Power Systems*, 20(2):1070–1078.

Zhao, L. (2006). A heuristic method for analyzing driver scheduling problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 36(3):521–531.

Résumé :

La conception d'un système de transport en commun doit prendre en compte différentes dimensions pour résoudre deux problèmes importants d'optimisation : l'ordonnancement des véhicules (le graphicage) et l'affectation des conducteurs (l'habillage). Dans nos travaux, nous nous sommes focalisés sur le problème de l'habillage. L'objectif est de minimiser le nombre de conducteurs en respectant toutes les contraintes sociales et économiques. Par sa nature combinatoire, l'habillage est considéré comme une tâche complexe du processus de conception de réseaux de transport en commun.

Nous avons proposé une approche fondée sur les hyper-heuristiques dont le principal avantage réside dans leur faculté d'adaptation à différents problèmes. Nous nous sommes intéressés plus particulièrement à une approche coopérative, capable de prendre en compte les changements au cours du processus de résolution. Nous avons étendu les fonctionnalités et amélioré les performances du framework traditionnel des hyper-heuristiques. L'algorithme proposé comporte une combinaison de plusieurs phases et plusieurs niveaux. La métaphore de la coalition est utilisée pour permettre la coopération entre hyper-heuristiques. Elle est destinée à favoriser la diversification des solutions et amplifier la capacité de recherche selon un contrôle décentralisé où chaque hyper-heuristique possède une certaine autonomie. Il est ainsi possible d'envisager différents modes de coopération entre les hyper-heuristiques : partage de solutions, apprentissage par mimétisme ou encore mise en concurrence de différentes stratégies de recherche. L'expérimentation a été réalisée aussi bien sur des instances réelles que sur des benchmarks. Elle a donné de bons résultats tant sur la déviation que sur le temps d'exécution.

Mots-clés : Hyper-heuristique, Habillage de transport en commun, Modèle organisationnel, Système multi-agents

Abstract:

The design of public transport system must take into account different dimensions to solve two main problems of optimization: the vehicles scheduling and driver scheduling. In our work, we focused on bus driver scheduling. Its objective is to minimize the number of drivers in accordance with social and environmental constraints. By its combinatorial nature, bus driver scheduling is considered a complex task in the design process of network transport.

We have proposed an approach based on hyper-heuristics whose main advantage lies in their ability to adapt to different problems. We are particularly interested in a cooperative approach, which is able to take into account changes in the resolution process. We have extended the functionality and improved performance of the traditional framework of hyper- heuristics by proposing a pattern based on an organizational model. The proposed algorithm consists of a combination of several phases and several levels. The metaphor of the coalition is used to make cooperate several hyper-heuristics. The coalition is intended to favor diversified solutions and expand search capacity with decentralized control where each hyper-heuristic has certain autonomy. It is thus possible to consider different ways of cooperation between the hyper-heuristics: sharing solutions, learning by mimetism or carrying out different competitive search strategies. The experiment was carried out both on real-world instances and benchmarks. It gave good results on both quality of solution and execution time.

Keywords: Hyper-heuristic, Bus driver scheduling, Organizational model, Multi-agent system

■ École doctorale SPIM - Université de Technologie Belfort-Montbéliard
F - 90010 Belfort Cedex ■ tél. +33 (0)3 84 58 31 39
■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr