A Multi-Agent based Optimization Method for Combinatorial Optimization Problems

Inès Sghir

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

Thèse de Doctorat

Inès SGHIR

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
Docteur de l'Institut Supérieur de Gestion de l'Université de Tunis

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique
Unité de recherche : Laboratoire d'Etude et de Recherche en Informatique d'Angers (LERIA)
Groupe de Recherche d'Informatique et d'Intelligence Artificielle (GR2IA)

Soutenue le 29 Avril 2016

A Multi-Agent based Optimization Method for Combinatorial Optimization Problems

JURY

Rapporteurs :	**M. Patrick SIARRY**, Professeur, Université Paris-Est Créteil
	M^{me} Wided LEJOUAD CHAARI, Maître de Conférences HDR, Université La Manouba
Examinateur :	**M. David LESAINT**, Professeur, Université d'Angers
Directeurs de thèse :	**M. Jin-Kao HAO**, Professeur, Université d'Angers
	M. Khaled GHÉDIRA, Professeur, Université de Tunis

Acknowledgements

I would like to express my special appreciation and thanks to my advisors: Professor Jin-Kao Hao and Professor Khaled Ghédira. You have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. This thesis would not have been possible without your help, support and patience.

Furthermore, I would like to thank Doctor Inès Ben Jaafar for the useful comments, remarks and engagement through the learning process of this thesis.

Profuse thanks go to Professor Patrick Siarry (Université Paris-Est Créteil) and to Doctor Wided Lejouad Chaari (Université La Manouba) for their participation to my thesis committee. I am glad they accepted to be members of the jury of this thesis. I appreciate particularly the precise and efficient reviewing of the complete thesis they performed to improve this thesis.

I would like to extend my gratitude to Professor David Lesaint for honouring my jury by presiding it.

I thank also the LERIA Laboratory of the Université d'Angers for providing me the material, the means and the expertise. Besides, I would like to extend my gratitude to its staff who welcomed me nicely.

I would like to dedicate my thesis and to express my deepest appreciation to my mother and father for all the sacrifices that you've made on my behalf. Your prayer for me was what sustained me thus far.

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

Context of work

This thesis deals with combinatorial optimization problems (COP) and their resolution strategies. Heuristic methods, which have been used for COP, aim to produce high quality solutions in reasonable computing time for hard problems. Recently, research studies have integrated techniques of diversification, in order to escape bad quality local optima.

An example of these methods is memetic algorithm which is an hybrid algorithm combining local search and evolutionary algorithms. These hybrid approaches aim at moving the optimization process from one local optimum to another. However, they do not have an efficient way to know when activating the suitable techniques according to the search process. Therefore, we can assume that existing methods do not integrate an intelligent mechanism to control the resolution process.

These methods use the centralisation model to solve the combinatorial optimization problems. Other types of solution methods are based on distributed model. In fact, the decomposition of an entire system into smaller subsystems and optimizing them in a distributed way to reach the system level optimum is one of the emerging approaches to deal with the growing complexity encountered in hard optimization problems. The multi-agent system is one of these emerging approaches.

In this thesis, we present a new generic approach to solve the limitation of heuristics. The proposed approach can be considered as an hyper heuristic method because it uses the multi-agent system to select the appropriate operators of meta-heuristic methods using learning techniques.

Objectives

The main objective is to develop a generic approach that makes the search strategy more intelligent and informative. For this purpose, we adopt some ideas from multi-agent systems. In a multi-agent system, the rational behavior of agents is very important to achieve the best possible local goal. The agents work collectively to achieve the best possible global or system objective. The proposed approach aims at solving hard combinatorial optimization problems. The result is a Multi-Agent based Optimization Method for Combinatorial Optimization Problems (MAOM-COP).

This method explores the multi-agent system as an intelligent tool to activate the search operator needed by the optimization process in a distributed environment. The distributed search combines some distinguishing characteristics of several well-established metaheuristics including variable neighborhood search, tabu search, and evolutionary algorithms. The intelligent selection is made by learning techniques.

Contributions

The main contributions of this thesis are summarized as follows:

A Multi-Agent based Optimization Method for Combinatorial Optimization Problems:

We elaborate a multi-agent based and distributed method for Combinatorial Optimization Problems. MAOM-COP is composed of decision-maker agent, intensification agents and diversification agents. The crossover agents and the perturbation agent are designed for the purpose of diversification. The tabu search agents are responsible for intensification. With the help of a learning mechanism, MAOM-COP dynamically decides the most suitable agent to activate according to the state of search process.

Under the coordination of the decision-maker agent, the other agents fulfill dedicated search tasks of intensification and diversification. The performance of the proposed approach is assessed on the following classical combinatorial optimization problems: the Quadratic Assignment Problem (QAP), the Graph Coloring Problem (GCP), the Winner Determination Problem (WDP) and the Multidimensional Knapsack Problem (MKP). For each of these problems, we tested the proposed approach and we compared it with the current state of the art approaches using the corresponding benchmark instances.

The results showed that our MAOM-COP algorithms are very competitive in terms of solution quality with the current best performing algorithms from the literature. These contributions led to two papers describing the application of MAOM-COP to the Quadratic Assignment Problem (QAP) (Sghir & al., 2015a) and the Graph Coloring Problem (GCP) (Sghir & al., 2015b). Other applications of the proposed approach will be submitted.

A Recombination-Based Tabu Search Algorithm for the WDP:

We propose a dedicated tabu search algorithm (TSX_WDP) for the Winner Determination Problem (WDP) in combinatorial auctions. TSX_WDP integrates two complementary neighborhoods designed respectively for the purpose of intensification and diversification. To escape deep local optima, TSX_WDP employs a backbone-based recombination operator to generate new starting points for tabu search and to displace the search into unexplored promising regions. The recombination operator operates on elite solutions previously found which are recorded in a global archive. The performance of the proposed algorithm is assessed on a set of

500 well-known WDP benchmark instances. Comparisons with five state of the art algorithms demonstrated the effectiveness of our approach. The proposed algorithm was presented in (Sghir & al., 2013).

Thesis plan:

This thesis is organized as follows:

Chapter 1 provides the necessary background for this work and the relevant literature review. In the first section of this chapter, we introduce the combinatorial optimization problems. Then, we review the most popular heuristic and metaheuristic approaches proposed in the literature for hard combinatorial optimization problems. In the second section, we present the multi-agent system and we provide a review of multi-agent based optimization approaches.

Chapter 2 presents the proposed approach, named a Multi-Agent based Optimization Method for Combinatorial Optimization Problems.

In the third chapter, we apply the proposed method to the quadratic assignment problem. We give an overview of current state of the art QAP approaches. We present the characteristic of each agent of the method. Then, we test it using the QAP benchmarks and we compare it with the best performing approaches from the literature.

The fourth chapter solves the graph coloring problem by exploring the proposed method. We start from an overview of the state of the art GCP approaches. We present the components of MAOM-GCP. Then, we show experimental results obtained by our algorithm for popular GCP instances, and we compare these results with those obtained by the current best-performing GCP algorithms from the literature.

In the fifth chapter, we propose the application of MAOM-COP to the winner determination problem. We introduce the problem and the state of the art research solving it. We describe the tasks of each agent composing this method to be adapted to the WDP. Experimental results of the proposed algorithm show that it can realize good quality results when they are compared with the best performing approaches for the WDP. Another elaborated algorithm is presented in the appendix of this thesis. This algorithm is named as a Recombination-Based Tabu Search Algorithm for the WDP (TSX_WDP). It includes several techniques of diversification which improve the tabu search.

In the sixth chapter, we study the multidimensional knapsack problem. We review the approaches solving this problem. We describe MAOM-MKP which is the application of the proposed approach to MKP. Then, we evaluate its performance by comparing it with the best approaches of the literature for this problem.

In the last chapter, we summarize our contributions in this thesis and we underline some possible future research topics.

1

State of the art

Combinatorial optimization problems (COPs) have been widely used in a number of application areas, such as transportation, production planning, design and data fitting, automatic control systems, signal processing, communications and networks, product and shape design, truss topology design, electronic circuit design, data analysis and modelling, statistics and financial engineering, etc. The resolution of these problems can be very complex because the number of candidate solutions can grow exponentially with the size of the problem. Heuristic and metaheuristic methods are often used to generate high quality solutions in reasonable computing time. Recent studies integrate other techniques to develop an intelligent optimization process. Multi-agent system is an efficient technique of artificial intelligence. Recently, due to their characteristics, multi-agent system has been applied to solve optimization problems.

This chapter describes the necessary background for our contributions. In the first section, we make a review of the most popular heuristic and metaheuristic approaches. In the second section, we present the multi-agent system and we provide a review of multi-agent based optimization approaches.

Contents

1.1 Combinatorial optimization problems and heuristic methods

In this section, we define combinatorial optimization problems. Furthermore, we give a brief overview of popular heuristic and metaheuristic approaches for such problems.

1.1.1 Combinatorial optimization problems

A mathematical optimization problem uses a function which is either maximized or minimized relative to a given set of alternatives. This is called the objective function and the set of alternatives is called the feasible region (or constraint region). An optimization problem aims to find the best solution among the feasible solutions.
Papadimitriou and Steiglitz (Papadimitriou & Steiglitz, 1998) proposed the following definitions for an optimization problem and for an instance of an optimization problem:

- *An optimization problem is a set I of instances of an optimization problem.*
- *An instance of an optimization problem is a pair (S, f), where S is a set of feasible solutions, and f an objective function or mapping $f : S \to R^1$. Given a minimization problem, the objective is to find an optimal solution $s \in S$ (also called global optimum) for which $f(s) < f(s'), \forall s' \in S$.*

The optimization problems can be divided into two categories: problems with continuous variables and problems with discrete variables, named combinatorial optimization problems. In the continuous problems, solutions are a set of real numbers. In the combinatorial optimization problems (COP), solutions are objects (e.g., integer, permutation, graph) from a finite or possibly countable infinite set.

1.1.2 Heuristic methods for combinatorial optimization problems

In this section, we present an overview of some heuristic and metaheuristic methods including greedy algorithms, neighborhood search algorithms, and evolutionary algorithms.

1.1.2.1 Greedy algorithms

Greedy algorithms generate feasible solutions from scratch. In each step, a value is assigned to a decision variable. The choice of this value depends on the decisions and their values made in the previous steps. This choice can influence the quality of final solution. We find different problems that have been solved with greedy heuristics, like GRASP solved time constrained vehicle scheduling problem (Atkinson, 1998), GRASP applied to quadratic assignment problem (Fleurent & Glover, 1999), and the nearest neighbor heuristic for the travelling salesman problem (Reinelt, 1994).

1.1.2.2 Local search algorithms

Local search corresponds to move from a solution to another one in its neighborhood according to some well-defined rules. A local search algorithm begins from an initial solution $x_1 \in X$. Then, at each step n, a new solution x_0 is generated in the neighborhood $V(x_n)$ of the current solution x.

Formally, $V(x) \subseteq X$ is called the neighborhood of x where $x \in X$. For instance, if X is a set of binary vectors and $x \in X$, a neighborhood $V(x)$ of x can be defined as bit-flip mapping for these binary vectors. This bit-flip neighborhood can be presented as the set of all solutions $x \in X$ realized from x by flipping a single coordinate from 0 to 1 or reversely.

Formally, the set of all the possible solutions generated with a single bit-flip move is given as: $V(x) = \{x' \in X \mid x \oplus bit - flip(u), \forall u \in \{1, ..., n\}\}$ where $\oplus$ is used to denote the move operator which presents the transition from the current solution x to the new neighboring solution x_0.

1.1.2.2.a Descent/ascent local search

Descent/ascent local search is the simplest form of local search (Algorithm 1.1). For a maximization problem, it is called ascent or hill climbing algorithm. In each iteration of this algorithm, a better solution is chosen among the neighbors of the current solution. For the next generation, the new solution is the new starting point. These operations are repeated until no better solution exists in the neighborhood. In the literature, there is two main types of descent algorithms, first improvement and best improvement. In the first improvement, the first better solution found in the neighborhood is selected. The best improvement explores the entire neighborhood, to find the best neighboring solution. Descent search can easily be trapped into local optima.

Algorithm 1.1. Descent algorithm

Require: Initial solution s
Ensure: Improved solution s_{best}

```
s_best ← s
while local optimum is not reached do
    Select a neighboring solution s' ∈ N(s_best)
    if f(s') < f(s_best) then
        s_best ← s'
    end if
end while
```

1.1.2.2.b Simulated Annealing (SA)

Kirkpatrick et al. (Kirkpatrick & al., 1983) proposed the simulated annealing algorithm (Algorithm 1.2). It is a randomized local search algorithm that has an explicit strategy to escape local minima. Iteratively, the current solution is modified by randomly selecting a move from the neighborhood. If this solution improves the current solution, it is directly accepted as a new current solution. Otherwise, it will be accepted, but with a certain probability based on the control parameters which are the temperature and the cost increase (for minimization). When the temperature is high and the cost increase is low, the move may be accepted with certain probability. According to predefined cooling schedule, the temperature is progressively decreased. When the temperature is adequately low, the method stops at a local optimum by allowing only improving moves. A great number of different simulated annealing algorithms have been proposed in the literature to solve optimization problems like the multidimensional knapsack problem (Drexl, 1988), the graph coloring problem (Lim & Wang, 2004), etc.

Algorithm 1.2. Simulated Annealing algorithm

Require: Initial solution s
Ensure: Improved solution s_{best}
n=0
$s_{best} \leftarrow s$
while stopping condition not reached **do**
    Select a random neighboring solution $s' \in N(s_{best})$
    $\triangle f = f(s) - f(s')$
    if $\triangle f \geq 0$ or $exp(\triangle f/T(n)) > random[0,1]$ **then**
        $s \leftarrow s'$
    end if
    if $f(s) < f(s_{best})$ **then**
        $s_{best} \leftarrow s$
    end if
    $n = n + 1$
end while

1.1.2.2.c Tabu search (TS)

The tabu search algorithm (Algorithm 1.3) was proposed by Glover (Glover, 1986). Tabu search (TS) is a neighborhood search method which employs flexible memory to avoid being trapped at local optimum. The visited solutions, which are maintained in this memory, are declared tabu to restrict the search space and avoid cyclic behavior. A short term or a long term memory can be used, in order to improve the exploration quality. When a tabu move can result in a solution better than any visited so far, this solution can be accepted. This is the aspiration criterion. In (Taillard, 1991), Taillard et al. proposed the robust tabu search (Ro-TS) for the quadratic assignment problem. Carlton and Barnes (Carlton & Barnes, 1996) applied reactive TS to solve the travelling salesman problem. Hertz et al. (Hertz & de Werra, 1987) used tabu search to solve the graph coloring problem, etc.

Algorithm 1.3. Tabu search algorithm

Require: Initial solution s
Ensure: Improved solution s_{best}
$T \leftarrow \varnothing$ { T is the tabu list}
$s_{best} \leftarrow s$
while stopping condition not reached **do**
  Find the best neighbor $s' \in N(s)$, such that ($m \oplus s = s'$ and ($m \notin T$ or aspired(m)=true))
  $s \leftarrow s'$
  Update the tabu list T by adding m
  if $f(s) < f(s_{best})$ **then**
    $s_{best} \leftarrow s$
  end if
end while

1.1.2.2.d Variable Neighborhood Search (VNS)

The Variable Neighborhood Search algorithm (VNS) (Algorithm 1.4) was proposed by Mladenovic and Hansen (Mladenovic & al., 2010). VNS is a local search algorithm which exploits the idea of neighborhood change. VNS is extended into many forms of algorithms like Variable Neighbor Descent, Reduced VNS, Basic VNS, Skewed VNS, etc. Different algorithms have been proposed using VNS like VNS for the graph coloring problem (Avanthay & al., 2003), VNS for the multidimensional knapsack problem (Puchinger & Raidl, 2005), VNS methodology for the vertex separation problem on generic graphs (Duarte & al., 2012), etc.

Algorithm 1.4. Variable Neighborhood Search algorithm

Ensure: Improved solution s_{best}
$s \leftarrow initial\_solution\_generation$, choose $\{N_k\}, k = 1, ..., k_{max}$
$s_{best} \leftarrow s$
repeat
  $k = 1$
  repeat
    $s' \leftarrow Random\_solution(N_k))$
    $s'' \leftarrow Local\_search(s')$
    if $f(s'') < f(s'))$ **then**
      $s \leftarrow s''$
    else
      $k \leftarrow k + 1$
    end if
    if $f(s'') < f(s_{best}))$ **then**
      $s_{best} \leftarrow s''$
    end if
  until $k = k_{max}$
until stopping condition is not reached

1.1.2.2.e Iterated Local Search (ILS)

Iterated Local Search (ILS) algorithm (Algorithm 1.5) has been introduced by Baum (Baum, 1987). Starting from a local optimum, ILS performs some perturbation moves to transform it to an intermediate solution. Then, the perturbed solution is used as an initial solution to apply local search procedure, in order to obtain a new local optimum.

If the perturbation is too weak, the search will often return to visited solutions. If the perturbation is too strong, ILS becomes a simple random restart algorithm. Consequently, the perturbation strategy should either be randomized or adaptive. ILS is applied to several optimization problems like the flow shop problem (Stützle, 1998), the quadratic assignment problem (Misevicius & al., 2006), the graph coloring problem (Chiarandini & Stützl, 2002), the graph bipartitioning problem (Martin & Otto, 1996), etc.

Algorithm 1.5. Iterated local search algorithm

Ensure: Improved solution s''

$s_0 \leftarrow initial\_solution\_generation$
$s \leftarrow local\_Search(s_0)$
while stopping condition not reached **do**
    $s' \leftarrow solution\_Perturbation(s, history)$
    $s'' \leftarrow local\_Search(s')$
end while

1.1.2.3 Population based algorithms

Among population based algorithms, we present the genetic algorithm and the ant colony optimization algorithm.

1.1.2.3.a Genetic Algorithms (GA)

Genetic algorithm (Algorithm 1.6) was developed by Holland (Holland, 1975). The genetic procedure is based on different operators such as mutation and recombination. The reproductive success is formulated with a fitness function.

The genetic algorithm performs the following basic process: a set of solutions, called population, are maintained. Some solutions are selected from this population to be recombined to form new solutions. The new solutions can be mutated to create other solutions. The population is updated with new solutions generated. The process is repeated until a given stop condition is satisfied.

We cite some examples of genetic algorithms solving the quadratic assignment problem (Tate & Smith, 1995; Misevicius, 2004), the vehicle routing problem (Braysy, 2011), the multidimensional knapsack problem (Chu & Beasley, 1998), etc.

Algorithm 1.6. Genetic algorithm

$t = 0$
$P(0) \leftarrow initial\_Population$
$evaluate\_Population(P(0))$
while stopping condition not reached **do**
  $P' \leftarrow Select\_Parents(P(t))$
  $P' \leftarrow Recombine(P')$
  $P' \leftarrow Mutate(P')$
  $evaluate\_Population(P')$
  $P(t+1) \leftarrow UpDate\_population(P(t), P')$
  $t = t + 1$
end while

Algorithm 1.7. Memetic algorithm

Require: |P| (size of population P)
Ensure: S^* (best solution found is recorded)
$P \leftarrow Generation(|P|)$
$evaluate\_Population(P)$
$S^* \leftarrow Best_solution(P)$
$f^* \leftarrow f(S^*)$
while stopping condition not reached **do**
  $P' \leftarrow Select\_Parents(P(t))$ (two or more parents are selected from the population)
  $s_0 \leftarrow Recombine(P')$ (one or more offspring can be generated from the parents)
  $s \leftarrow local\_Search(s_0)$ (the offspring can be improved by local search)
  $P(t+1) \leftarrow UpDate\_population(P(t), s)$ (population is evaluated and updated with the new solutions)
  $(S^*, f^*) \leftarrow UpDate\_Best\_solution(S^*, f^*, P)$ (the best solution is retained)
  $t = t + 1$
end while

1.1.2.3.b Ant Colony Optimization (ACO)

Ant colony optimization (ACO) was proposed by Dorigo et al. (Dorigo & al., 1991). In each iteration, a number of artificial ants is used in a greedy way, to generate solutions. Then, each ant chooses the amount of pheromone to be included into the current partial solution. When a complete solution is generated, the procedure is repeated with updated pheromone levels, until reaching a stopping criterion.
Many works have been developed using ACO like the graph coloring problem (Dorigo & al., 1991), the job shop scheduling problem (Colorni & al., 1994), the vehicle routing problem (Bell & McMullen, 2004) and the multidimensional knapsack problem (Changdar & al., 2013).

1.1.2.4 Hybridizing metaheuristics with (meta-) heuristics

Recently, researchers have combined metaheuristics to solve optimization problems. Generally, it consists on merging local search methods and population based methods. Population based methods can determine the promising regions of the search space. Local search methods determine quickly the best solutions. One of the successful hybrid algorithms is memetic algorithm (Algorithm 1.7), which is proposed by Moscato (Moscato, 1989).

1.2 Multi-agent based optimization approaches

Multi-agent system (MAS) technology becomes a popular paradigm used for the conceptualization, design, analysis and implementation of many approaches and solutions. In this section, we present multi-agent systems, then, we focus on studies using this technique for optimization.

1.2.1 Multi-agent system

An agent is a physical or virtual entity that: (Ferber, 1999)

— can act in an environment;
— can communicate with other agents;
— is moved by a set of tendencies which can be an individual objective or a satisfaction function;
— has its appropriate resources;
— can perceive its environment with limit;
— has skills and provides services;
— can reproduce;
— has behaviors to satisfy objectives using its resources and its skills based on its perceptions, its representations and its communications with other agents.

A multi-agent system (MAS) is a system composed of agents which interact, most of the time, according to cooperation and competition modes. In fact, in a MAS, each agent has a partial view point to solve a problem because it has limited information about this problem. Each agent is only responsible for its knowledge, its actions and its communications with other agents. Nevertheless, it has no global view of the whole system. Therefore, a MAS is a distributed system where the tasks to be realized and the skills to make are distributed by agents. The agents interact in a MAS according to the following types of interaction including the cooperation to solve a common purpose, the coordination and the negotiation.

In a MAS, we distinguish three types of agents (Ferber, 1999):

— Cognitive agent: it has a capacity of reasoning and knowledge to execute its tasks and to manage the interactions with other agents.
— Reactive agent: it does not have a representation of his environment, but it acts with a behavior of stimulus answer and it reacts in a present state of its environment. This type of agent does not consider the past and does not plan the future.

— Hybrid agent: it has reactive and cognitive components to improve its capacity of reasoning. Analogically with the human interactions in a social organization, in a MAS, agents have to communicate because a single agent is an isolated, deaf and mute individual.

The MAS is applied to several domains like optimization and decision problems, modeling and simulation. Furthermore, it can be used in distributed applications such as management of industrial systems, control of the aerial traffic, telecommunication networks, e-commerce, robotics, image segmentation, etc.

We will present a review of multi-agent based optimization algorithms and their applications. These studies are divided into two types of frameworks based on agents functionality. Some frameworks use agents which explore the same search space, but with various strategies of resolution. They can be called as strategy based agents frameworks. In other frameworks, each agent handles a part of search space. It consists in decomposing the global problem to different sub-problems. This decomposition can concern the variables, the constraints and the objective functions. This category of frameworks can be called as sub-problems based agents frameworks.

1.2.2 Strategy based agents frameworks

In strategy based agents frameworks, agents are responsible for actions and behaviors. These agents explore learning or improving certain functionality. The asynchronous team (A-Team) (Talukdar & al., 1993, 1996) is the first conceptual framework that uses autonomous and cooperated agents to solve optimization problems. A-Team is based on features from a number of systems like insect societies, cellular communities, genetic algorithms, simulated annealing, local search, and brainstorming. A-Team is composed of a set of interconnected memories to create a strongly cyclic network. Each memory, which saves solutions produced by each agent, is dedicated to one problem, in order to form a population. In this network, each agent is in a closed loop. All agents work in parallel way and no one of them waits for results from another agent. This cooperation between agents is called asynchronous cooperation.

These agents are divided into two types: construction agents which add solutions to population and destruction agents which delete solutions from population. Each agent defines three components: an operator or an algorithm that generates solutions, a selector that selects which solutions are maintained, and a scheduler which organizes behaviors (when solutions have to be selected and with what resources). The intelligence of the agents is realized by their selector and their schedulers. Their skills are resident in their operators.
The A-Team has been applied to several optimization problems like travelling salesman (De Souza, 1993), control of electric networks (Talukdar & Ramesh, 1994; Avila-Abascal & Talukdar, 1996), job-shop-scheduling (Chen & al., 1993), train-scheduling (Tsen, 1995), and steel and paper mill scheduling (Rachlin & al., 1996; Lee & al., 1995).

In (Milano & Roli, 2004), Milano et al. proposed the Multi-AGent Metaheuristic Architecture (MAGMA). This approach is a conceptual framework that combines hybrid metaheuristics in a multi-agent system. MAGMA is divided into different levels of abstraction. At each level, there are one or more agents. LEVEL-0 contains feasible solutions for the upper level. The agent, in this level, initializes the search process. LEVEL-1 is composed of several agents that improve solutions received from LEVEL-0 using local search algorithms. LEVEL-2 agents guide the search toward promising regions and provide mechanisms for escaping local optima by exploring evolutionary techniques. LEVEL-3 Agents are agents responsible for coordination. These agents decide which information to communicate between the agents of LEVEL-1 and LEVEL-2. They know the strategy of all other level agents. In this work, several metaheuristics are used like GRASP, ILS, MA and ACO. MAGMA can decompose the metaheuristics components into a group of agents and makes communication with these agents to exchange information, but this communication is not dynamic. Each level depends on other levels.

Jędrzejowicz and Wierzbowska (Jędrzejowicz & Wierzbowska, 2006) elaborated the JADE-A-Team (JABAT) which is based on the A-Team architecture. JABAT is composed of two types of Optimization agents (called OptiAgent) and SolutionManagers agents. Each OptiAgent implements improvement algorithms (simulated annealing, tabu search, genetic algorithm, local search heuristics). SolutionManagers agents have the common memory which contains solutions generated by OptiAgents. SolutionManagers agents are responsible for updating the common memory and sending individuals to OptiAgents. All agents act in parallel way. JABAT has no intelligence communication between agents.

In (Barboucha & Jędrzejowicz, 2007), Barbucha et al. applied the JABAT framework to the vehicle routing problem (VRP). They integrate four instances of OptiAgents which consist in four local improvement procedures: OA 2-Opt agent is an implementation of the 2-opt local search algorithm for VRP which operates on a single route, OA StringCross agent exchanges two strings (routes) of customers by crossing two edges of two different routes. OA 2-Lambda agent executes the local search algorithms based on λ-interchange local optimization algorithm. This agent solves only the instances in which the customers are uniformly arranged on the plane. OA 2-LambdaC agent explores the same algorithm of OA 2-Lambda agent, but it concentrates on instances in which the customers are clustered.

In addition, JABAT framework has been implemented for multi-mode resource-constrained project scheduling problem with minimal and maximal time lags problem (MRCPSP-GPR)(Jędrzejowicz & Ratajczak-Ropel, 2013). They created five instances of OptiAgents whose two agents (optiLSAm and optiTSAe) explore two local search algorithms with different neighborhood structures, one agent called optiTSAe applies tabu Search algorithm, one agent named optiCA executes crossover algorithm, and one agent named optiPRA employs path relinking algorithm.

In (Barbucha, 2013), JABAT is applied to the capacitated vehicle routing problem. In this work, they used four instances of OptiAgents which can be divided into two groups operating on one (intra-route) or two (inter-routes) routes and include: modified implementations of 3-opt procedure (Lin, 1965) and λ-interchange local optimization method (Osman, 1993)(λ=2), and two dedicated local search methods, based on moving/exchanging selected nodes or edges between routes.

In (Xu & Liu, 2006), Xu et al. proposed a multi-agent based particle swarm optimization (HMAS) for cluster analysis. In this framework, a group of agents forms a swarm and a group of swarms agents forms sub-populations. In these swarms, agents have the ability of self-organization, learning and detecting local environment.

The framework of (Bae & al., 2009) uses the multi-agent system to simulate the traffic signal system. The agents are the driver agent and the vehicle agent. It is the vehicle agent that is responsible for optimization tasks by exploring the simulated annealing method.

The EMAS proposed by (Hanna & Cagan, 2009) explores the A-Team architecture using a group of strategy agents which execute genetic algorithms. In each iteration, all agents are activated and perform actions based on their genetic sequence, in order to product solutions. Then, the considered agents are evaluated according to the new solutions generated. The solutions provide a basis for increasing or decreasing an agent's fitness. The evaluation of agents is based on the average solution quality in a memory. The memory is made by saving new solution found by each agent. In fact, reproduction phase is applied only for selected parents based on their fitness and new agents are created. During the selection phase the weakest individuals (agents with low fitness value) are removed from the population. This approach was applied to the travelling salesman problem.

Meignan et al. (Meignan & al., 2010) proposed the Coalition-Based Metaheuristic (CBM). It is a framework that used the multi-agent paradigm to select between the intensification techniques and the diversification techniques according to the search state. Each technique is manipulated by an agent. All agents are guided by a decision process to choose the appropriate actions which are dynamically adapted during the search using reinforcement learning. These agents are always in coalition state because they are in competition to find the best solution. There is no communication between them. The proposed approach is applied to the vehicle routing problem.

The multi-agent approach presented in (Guo & al., 2013) is composed of several agents which explore the genetic algorithm. The learning mechanism, which is built for each agent, guides the agents to choose the most appropriate genetic operators during each generation. The genetic operators are the crossover operators and the mutation operators. In fact, there is an operator pool that stores these operators to be selected. In this pool, the crossover operators and the mutation operators are saved in pairs. Each pair corresponds to one crossover operator and one mutation operator. At the beginning of each generation, two operators have the same intensification or diversification search tendency. After applying a decision making heuristic, one of these operators is selected for each agent. The decision making heuristic is performed to learn adaptively and concurrently the behavior of all agents, in order to predict the most suitable operator. The proposed approach aims at solving the long-term car pooling problem.

1.2.3 Sub-problems based agents frameworks

For sub-problems based agents frameworks, we will present briefly two well-known frameworks that have been the results of various works: Probability Collectives (PC) approaches and Distributed Constraint Optimization Problems approaches (DCOP).

Probability Collectives approaches Probability Collectives (PC) in the framework of Collective Intelligence (COIN) was first proposed by David Wolpert in 1999 (Wolpert & Tumer, 1999). It is an extension from distributed optimization methodology for modelling and controlling distributed MAS, inspired from a sociophysics viewpoint with deep connections to game theory, statistical physics, and optimization (Wolpert & Tumer, 1999; Wolpert & al., 2006). In PC, each variable is an independent agent. The action of these agents is assigned via probability distributions which are updated independently according to their local goal and to the global or system objective (Wolpert & Tumer, 1999; Bieniawski, 2005; Wolpert & al., 2006). The process is repeated until reaching equilibrium. This equilibrium concept is referred to Nash equilibrium (Basar & Olsder, 1995). The PC approach has been applied to unconstrained problems like: (Bieniawski, 2005; Kulkarni & Tai, 2008, 2009; Bhadra & al., 2006; Wolpert & al., 2006; Huang & al., 2005; Vasirani & Ossowski, 2008; Huang & Chang, 2010; Smyrnakis & Leslie , 2009), as well as constrained problems like: (Wolpert & Tumer, 1999; Bieniawski, 2005; Sislak & al., 2011; Wolpert & al., 2004; Autry, 2008; Kulkarni & Tai, 2011, 2010).

1.2.3.0.a Distributed constraint optimization approaches

Several optimization problems can be classified as Constraint Satisfaction Problems (CSP) such as the graph coloring problem, the scheduling problem, the asset allocation problem, etc. Solving a CSP is equivalent to finding an assignment of values to all variables such that all constraints are satisfied. The Distributed Constraint Satisfaction Problem resolution is the distributed version of constraint satisfaction problems resolution (CSP). In DCOP, each variable is allocated to an agent which has control of its value. Below, we present some representative DCOP algorithms.

ADOPT (Modia & al., 2005) is the first asynchronous complete algorithm for optimally solving the Distributed Constraint Optimization Problem (DCOP). Each agent must optimize a global objective function, so it must exchange the choice of variable's values to other agents. DCOP uses only local communication with neighboring agents. The global objective function corresponds to the set of constraints and each agent knows about the constraints in which its variables are involved. ADOPT uses the distributed backtrack search via a novel search strategy and backtrack thresholds. These techniques help agents to explore locally and asynchronously partial solutions. In order to guarantee a good quality solution in a reasonable time, ADOPT employs the bounded-error approximation algorithm.

In (Yeoh & al., 2010), they proposed a Branch-and-Bound ADOPT (BnB-ADOPT). It is a memory-bounded asynchronous DCOP search algorithm that employs the message-passing and the communication framework of ADOPT (Modia & al., 2005). In BnB-ADOPT algorithm, the best-first search was replaced by the depthfirst branch-and-bound search. Like ADOPT algorithm, the agents in BnB-ADOPT algorithm are implemented in asynchronous and concurrent way. The communication is only between agents that share constraints. The agents are ordered via a pseudo-tree.

Distributed stochastic algorithm (DSA) (Fabiunke, 1999) is a uniform algorithm. In each step, each agent sends its variable value, to its neighboring agents. When it modifies this value, in the previous step, and it receives the state value from its neighbors, it can decide, randomly, to keep its current value or change to a new one. But, the neighbors have to maintain their values. The modification of variable value aims at reducing violated constraints. The DSA uses a probability p to select how frequently neighboring agents change values. p is called the degree of parallel executions. DSA has been used in several DCOP with various extensions (Fabiunke, 1999; Fitzpatrick & Meertens, 2001) like graph coloring problem (Zhang & al., 2002), scheduling problem (Zhang & al., 2003), etc.

1.3 Conclusion

In this chapter, we described combinatorial optimization problems and heuristic methods which are used to solve them. Heuristic methods can generate high quality solutions in reasonable computing time. They are improved by techniques of diversification and techniques of intensification, in order to escape local optimum. Other studies explore multi-agent system to create distributed algorithms for solving optimization problems. In the second section, we introduced the agent paradigm and their applications to optimization problems. These methods are motivated by specific features offered by MAS like distributed computing, agent cooperation and dynamic decision making. Indeed, multi-agent systems have been successfully applied to solve many challenging and various problems encountered in various settings. The review above aims to describe some recent MAS-related studies to illustrate the interest of MAS for building expert and intelligent systems for problem solving. Our work shares similarities with these previous studies in the sense that it is based on the generic framework of multi-agent systems. The proposed work, as described in the next chapter, distinguishes itself by some particular features including the distributed and collaborative architecture, the design of both intensification and diversification agents as well as the decision making method based on reinforcement learning. In our work, we investigate a new solution approach for the combinatorial optimization problems based on the principles of multi-agent systems (MAS).

2

A Multi-Agent based Optimization Method for combinatorial optimization problems

This chapter presents a new Multi-Agent based Optimization Method for Combinatorial Optimization Problems (MAOM-COP). A multi-agent system (MAS) is typically composed of a group of interacting agents where each agent has one or more basic skills. The agents can collectively find solutions to a difficult problem even if each agent alone can not solve it. These agents explore several optimization techniques like local search algorithms, crossover operators and perturbation techniques. The selection of each one of these techniques is made in an intelligent way based on reinforcement learning. MAOM-COP is evaluated on a number of classical combinatorial optimization problems.

Contents

2.1 Introduction and motivation

As we have presented in the first chapter, several metaheuristic methods and multi-agent based optimization methods have been proposed in the literature to tackle optimization problems. In this work, we present a generic Multi-Agent based Optimization Method for Combinatorial Optimization Problems (MAOM-COP). MAOM-COP is generic and it can be applied to classical optimization problems. In a search algorithm, one of the most important issues is to find the right balance between diversification and intensification. MAOM-COP distinguishes itself by its learning based distributed computing model, in order to know if the search needs exploration or exploitation, in an intelligent way.

In our framework, we add a cooperative dimension to the evolutionary process. Cooperation between individuals is modeled by embodying each strategy in an autonomous and learning agent which can communicate. The team of agents navigates the search space cooperatively. Some agents of MAOM-COP, which are called intensification agents, employ local search algorithms to reach high quality local optima. Other agents, which are called diversification agents, are trigged, when the search needs to be diversified. These last agents explore perturbation techniques and crossover operators.

In the following section, we will present the architecture of MAOM-COP. Then, we will explain the behavior of each agent. In next chapters, we will apply and evaluate the performance of MAOM-COP on a number of classical combinatorial optimization problems.

2.2 MAOM-COP architecture

The proposed MAOM-COP architecture contains the following agents: decision-maker agent, intensification agents and diversification agents. The intensification agents are composed of agents which perform local search algorithms. The diversification agents are composed of two types of agents which are perturbation agent and crossover agents. Figure 2.1 illustrates the generic MAOM-COP architecture, whose components are detailed in the following sections. Algorithm 2.1 describes the generic procedure of the proposed method. In addition to the above agents, MAOM-COP relies on reinforcement learning based on decision matrices for decision making. By linking a set of conditions and a set of actions, such a matrix helps an agent to know the agents with which it will communicate, according to the state of the search process.

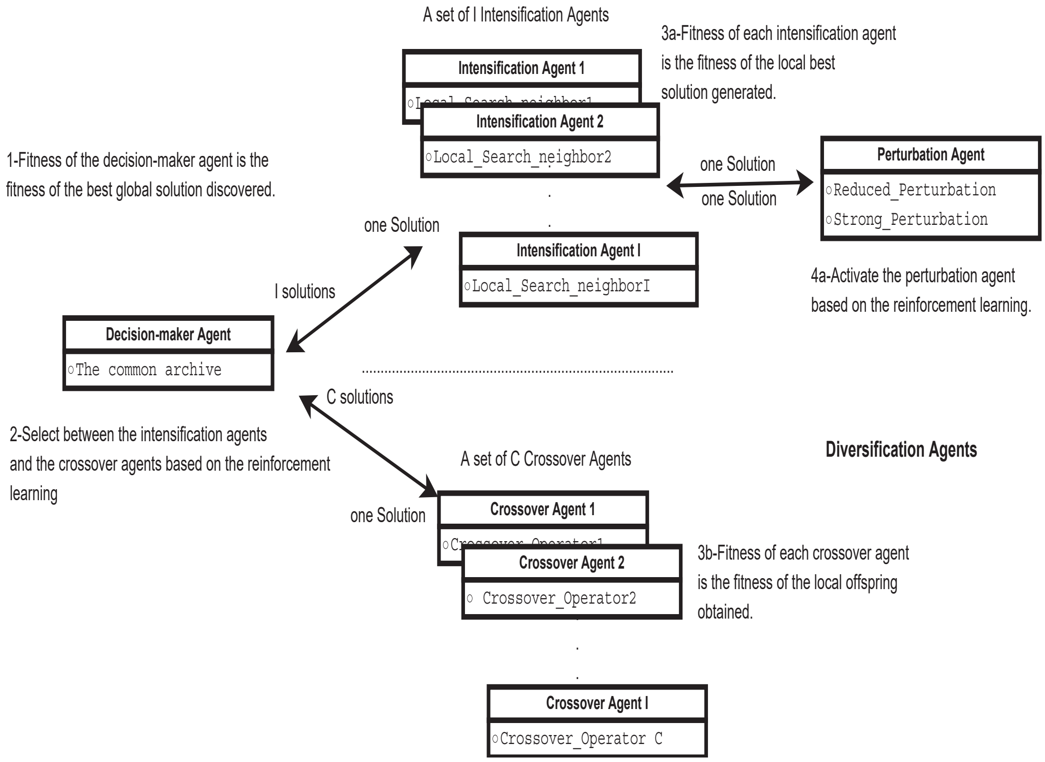

Figure 2.1 – The architecture of the search using cooperative agents in MAOM-COP

2.2.1 Decision matrix with reinforcement learning

In the proposed MAOM-COP, some agents (decision-maker agent and intensification agents) need to decide when to activate other agents and which agents to activate. Such decisions are made based on a decision matrix which is dynamically adjusted by a reinforcement learning process. This technique allows to adapt the search strategy according to the experiences acquired during the search process. For instance, after the application of an intensification action, if the search is observed to be stagnating (e.g., captured by the condition 'the best solution is not improved for a high number of iterations'), the applied action should be avoided for the next search step and an action ensuring more diversification should be favored. Inversely, if the applied action leads to a search progress (e.g., captured by the condition 'the best solution is just improved'), the same action should be given a high chance to be applied again (notion of reward).

We use a pair $(condition, action)$ to represent the decision rules. The $condition$ part corresponds to the necessary prerequisite to trigger an associated action, the $action$ part indicates which action is to be performed. Let C be the set of conditions and A the set of actions to perform. For a condition C_i, a weight W_{ij} (initialized to 0) is associated to each action A_j. The conditions are defined based on the improvement situation occurred at the end of each search generation (i.e., one $while$ iteration in Algorithm 2.2). The decision matrix W is used to dynamically influence the probability of applying each action under each condition.

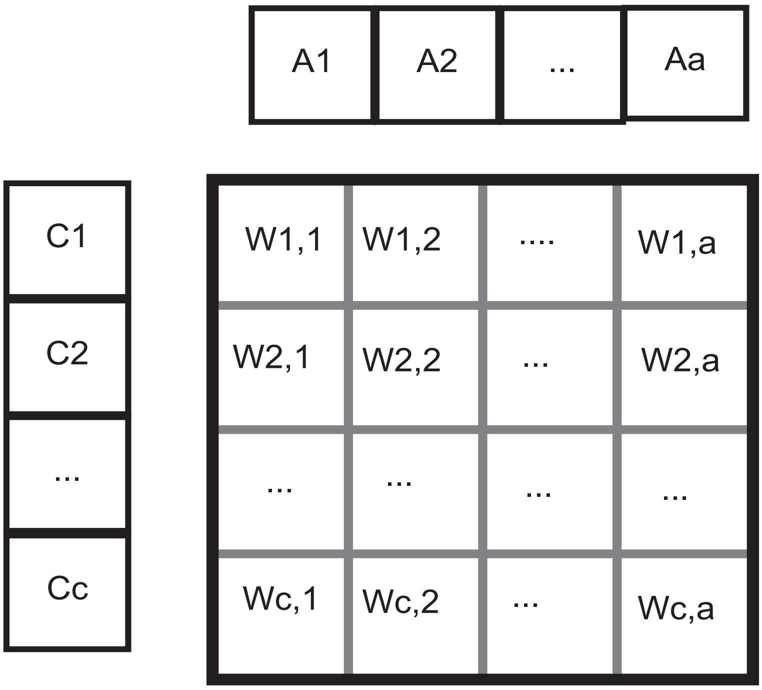

Figure 2.2 – Structure of the decision matrix

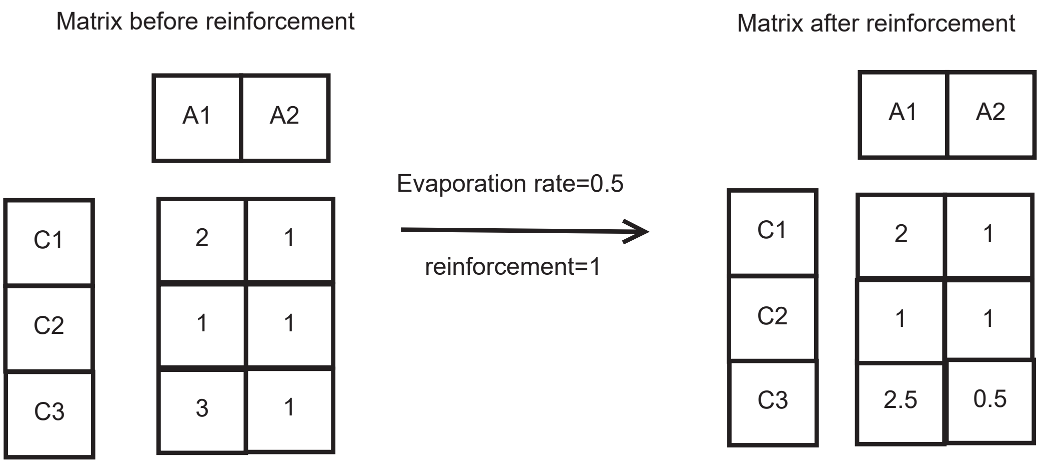

Figure 2.3 – An example of reinforcement learning with the decision matrix

We suppose that the current condition is C3 (e.g., the local best solution has been improved in recent 20 generations). At this condition, action A1 (e.g., activate intensification agents) must be reinforced for the current generation. Then, reinforcement is applied by increasing the weights $W31$ *to augment the chance of selecting the applied action under this condition (e.g.* $W31 = 3 \times 0.5 + 1 = 2.5$*). Nevertheless, the weights* $W32$ *is decreased by* μ *(e.g.* $W32 = 1 \times 0.5 = 0.5$

Given the decision matrix W (Figure 2.2), we use the following equation (Guo & al., 2013) to calculate the probability $P(C_i, A_j)$ of applying action $A_j \in A$ under condition $C_i \in C$.

$$P(C_i, A_j) = \frac{W_{ij}}{\sum_{j \in A} W_{ij}} \tag{2.1}$$

At the beginning of each generation, the improvement situation is assigned to a default condition. Then, according to the decision matrix, the appropriate action for this condition is selected according to the probability given in Eq. (2.1). At the end of each generation, the performed action is evaluated with respect to its condition and the concerned weight value is increased if an improvement in solution quality is obtained in this generation. We use a credit assignment to perform the reinforcement learning in order to identify the beneficial experiences and determine a reward for them. Here, an experience is represented as a triplet (*condition* C_i, *action* A_j, *improvement* V). When a new best local or global solution is found, the weight value W_{ij} which is related to the action of this generation is reinforced by adding a reward rate σ to W_{ij}. Before adding the reinforcement value, all the weight values W_{ij} in the decision matrix corresponding to the related condition, are decreased with an evaporation value μ, in order to enlarge the influence of the new reward obtained in the current generation. The reinforcement with reward σ is then performed using the following equations:

$$W'_{ij} = \mu \times W''_{ij} \tag{2.2}$$

$$W_{ij} = \mu \times W'_{ij} + \sigma \tag{2.3}$$

where W'_{ij} is the weight value before adding the reinforcement σ, W''_{ij} is the weight value before the evaporation μ, and σ is the learning factor.

Figure 2.3 shows an illustrative example of this reinforcement learning process (More information about the example are given in section 2.4.1). In the proposed MAOM-COP, such a matrix is used by the decision-maker agent (section 2.4) and the intensification agents (section 2.5). The respective conditions and actions used by these agents are provided in these sections.

2.3 MAOM-COP process

The MAOM-COP procedure is summarized in Algorithm 2.1. The decision-maker agent initiates the search with a random feasible solution. Based on a decision matrix W explained in the previous subsection, it decides to trigger either the intensification agents or the crossover agents belonging to diversification agents (lines 1-9 of Algorithm 2.2). As a result, one of the two following cases occurs.

— **Case 1: Intensification agents are triggered**:
This case corresponds to the situation where the decision-maker agent decides that a more intensified search is needed considering the current search state. For this, it triggers the intensification agents by sending them the

current solution (lines 10-11 of Algorithm 2.2). Each intensification agent looks for the best solution in the predefined neighborhood for q iterations by applying a local search procedure. Each agent uses a different neighborhood structure and starts with the solution received from the decision-maker agent (lines 6-11 of Algorithm 2.3).

After an iteration (i.e., one $while$ iteration in Algorithm 2.3) of the intensification agent concerned, each intensification agent decides whether it needs to communicate with the other intensification agent or it can continue its process without information exchange. This decision depends on another decision matrix Q (see section 2.5). Q has the same mechanism as the decision matrix W of the decision-maker agent, but here, the actions to be performed are either to trigger another intensification agent (for intensification) or a perturbation agent (for diversification).

When the search needs to be intensified, the agent concerned calls another intensification agent and remains blocked until it receives the best solution sent by the other intensification agent. Thereafter, the requesting agent completes its search starting with the solution received. When the desired action is about diversification, the perturbation agent is triggered (lines 17-23 of Algorithm 2.3). This agent performs one of two types of perturbations (reduced and strong perturbations) with the purpose of helping the intensification agents to move towards new search areas. According to whether the requesting intensification agent needs a small or large diversification, either a reduced perturbation or strong perturbation is performed.

The new solution from the perturbation is passed to the intensification agent to continue its search. The intensification agents execute for a number of iterations by exchanging information as we just explained (lines 27-28 of Algorithm 2.3). The best solutions obtained are forwarded to the decision-maker agent (line 38 of Algorithm 2.3). The decision-maker agent records the solutions received in an archive that represents a common memory shared by all agents in the algorithms (lines 17-26 of Algorithm 2.2).

— **Case 2: Crossover agents are triggered**:
If the crossover agents are activated (lines 13-15 of Algorithm 2.2), they are applied to two parent solutions (which are selected from the archive) to create offspring solutions (section 2.7). These new solutions are sent back to the decision-maker agent which stores them in the shared archive if they are of good quality. The best offspring solution is taken as the new starting point, to continue the search.

The decision-maker agent uses the current best solution found so far to start the next cycle (generation) of the algorithm. The values of the decision matrix are updated according to the new state reached by the last generation, in order for the decision-maker agent to activate the appropriate agents for the next generation. This search process is repeated until a stop condition is satisfied (e.g., a maximum number of generations) and the best solution discovered is retained as the final result.

Algorithm 2.1. MAOM-COP generic procedure

Require: Four types of agents: one decision-maker agent, i intensification agents, one perturbation agent, c crossover agents

Ensure: best information (i.e., solution) found

decision-maker agent is active until it needs exchanging with intensification agents or crossover agents (Algorithm 2.2)
if decision-maker agent decides to trigger intensification agents **then**
    the intensification agents are activated and decision-maker agent waits for new information from them (Algorithm 2.3)
    if an intensification agent requests help from the perturbation agent **then**
        perturbation agent is activated and the intensification agent is blocked until it receives information from perturbation agent (section 2.6)
        perturbation agent is killed after sending information to the corresponding intensification agent
    end if
    if an intensification agent requests help from the other intensification agents **then**
        the requesting intensification agent is blocked until it receives information from the other intensification agents (Algorithm 2.3)
    end if
    the intensification agents are killed after sending information to decision-maker agent
end if
if decision-maker agent decides to trigger crossover agents **then**
    the crossover agents are activated and decision-maker agent waits for new information from the crossover agents (section 2.7)
    the crossover agents are killed after sending information to decision-maker agent
end if
Return best information found

2.4 Decision-maker agent

The decision-maker agent is the coordinating agent. According to the decision making matrix W (section 2.2.1), the decision-maker agent decides whether it triggers the intensification agents (for more intensification) or crossover agents (for more diversification). It records the high-quality solutions which are discovered during the search in the shared memory (archive) (Algorithm 2.2).

The decision-maker agent thus exchanges information with the intensification agents or the crossover agents. The decision-maker agent stays alive until reaching a stop criterion (a cutoff time limit, an allowed number of generations). During its life, it is blocked when other agents are activated. So, it has only one life cycle.

2.4.1 Conditions and actions

During the search process of our algorithm, three types of solutions are used to define conditions for agent activation: the *local current solution* obtained by each agent, the *local best solution* obtained by each agent and the *global best solution* obtained among all agents in the process. The decision matrix of the decision-maker agent is composed of four different conditions which cover significant situations that may occur during the search process:

— C_1 = The algorithm does not reach g_0 generations (cycles);

— C_2 = The local or global best solution is improved in the recent g_1 generations and this improvement is a small improvement in the objective function value F;

— C_3 = The local or global best solution is improved in the recent g_1 generations and this improvement is a large improvement in the objective function value F;

— C_4 = The global best solution does not have been improved in the recent g_2 generations. This solution is a deep local optimum or an optimum solution.

where g_0, g_1 and g_2 are parameters set by the user according to the total allowed generation number or total run time.

The set of actions are:

— A_1 = Activating the intensification agents;

— A_2 = Activating the crossover agents.

At the beginning of the search or when there is a large improvement obtained by the application of an action between two successive generations (this corresponds to the situations of C_1 and C_3), the search progresses well and, in this case, it is appropriate to make intensified search by triggering the intensification agents. If the decision-maker agent observes no improvement or an insignificant improvement (this corresponds to the situations of C_2 and C_4), the search is stagnating and needs to be diversified by activating the crossover agents.

After each generation (i.e., when the activated agents return their found solution), the decision-maker agent updates its decision matrix as explained in section 2.2.1. Figure 2.3 illustrates how the decision matrix is changed by the reinforcement learning procedure. We suppose that in iteration i of Algorithm 2.2, the current condition C_3 is verified (i.e., the local best solution is greatly improved in the recent $g_1 = 20$ generations). Under this condition, action A_1 which corresponds to activating intensification agents is applied for the current generation. Reinforcement learning is applied by increasing the weight W_{31} to augment the chance of selecting the corresponding action under this condition. In Figure 2.3, we show for this example, the initial decision matrix (left), and the matrix (right) after the update with a reward value $\sigma = 1$ and an evaporation value $\mu = 0.5$.

2.4.2 Archive of elite solutions

The decision-maker agent records the best solutions discovered during the search in an archive. These solutions are generated and submitted by the intensification agents and the crossover agents. Even if the archive is shared by all the agents of the model, only the decision-maker agent is responsible to update it. Each time the decision-maker agent receives a new solution, it adds the solution in the archive, if it is of good quality and is not present already in the archive.

2.5 Intensification agents

The intensification agents are designed for intensification. During its life time, an intensification agent employs a neighborhood to generate improved solutions. During the search, each intensification agent can decide, with the help of a decision matrix, to exchange information with other alive intensification agents or with the perturbation agent depending on its state of search. At the end of each intensification agent run, the best solution found by the agent is sent to the decision-maker agent (Algorithm 2.3). Below, we explain the conditions and the actions employed by intensification agents.

2.5.1 Conditions and actions

As explained at the beginning of this section, each intensification agent can decide, with the help of a decision matrix, to exchange information with other alive intensification agents or with the perturbation agent depending on its state of search. In this section, we present the conditions and the actions employed by the intensification agents. The decision matrices are managed by the same technique of the decision matrix of decision-maker agent (section 2.2.1).

The set of the conditions are:

— C_1 = The local best solution is improved in recent q_3 generations and this improvement is a small improvement;

— C_2 = The local best solution is not improved in recent q_4 generations;

— C_3 = The local best solution is not improved in recent q_5 generations and $q_5 > q_4$.

where q_3, q_4 and q_5 are parameters set by the user according to the total generation number or total run time.

Algorithm 2.2. Decision-maker agent behavior

Require: parameter opt, $interval$ and max_opt
Ensure: A best solution S_{best}

$S \leftarrow Random\_solution$ {Random initial solution}
$S_{best} \leftarrow S$ {S_{best} records the best solution found so far}
$F_{best} \leftarrow F$ {F_{best} records the best objective value reached so far}
$opt \leftarrow 0$ {opt is the counter for consecutive non-improving local optimum}
$W \leftarrow 0$ {Initialization of the decision matrix of the decision-maker agent}
$pop \leftarrow \emptyset$ {pop is the archive of elite solutions found during the search}
while Stopping condition not reached **do**
  Update W based on $interval$, max_opt and opt {$interval$ (matching the improvement of solution between two successive iterations), max_opt and opt help to identify the current condition, sections 2.2.1 and 2.4.1}
  $Action\_type \leftarrow$ Select an action (agents) to activate based on W {section 2.4.1}
  if $Action\_type$ = Intensification agents **then**
    Activate intensification agents and send S to the Intensification agents
  else
    Activate crossover agents and send S to the crossover agents
    $opt \leftarrow 0$
  end if
  $S_1 \leftarrow \emptyset$, $S_2 \leftarrow \emptyset$ {S_1 and S_2 are two solutions received from the activated agents, initialized to empty}
  if $S_1 \neq \emptyset$ AND $S_2 \neq \emptyset$ **then**
    if $F(S_1) \geq F(S_2)$ **then**
      $S \leftarrow S_1$
    else
      $S \leftarrow S_2$
    end if
    $tr \leftarrow Exist(S_1, S_2, pop)$ {Check if S_1 and/or S_2 are in the archive pop}
    if $tr = false$ **then**
      Add S_1 and/or S_2 to pop {Add both solutions or one of them in pop}
    end if
    Let S' be the best solution between S_1 and S_2
    if $F(S') \leq F_{best}$ **then**
      $S_{best} \leftarrow S'$, $F_{best} \leftarrow F(S')$
    else
      opt=opt+1
    end if
  else
    Block this agent {The decision-maker agent waits for solutions from other agents}
  end if
end while
Return F_{best} and S_{best}

The set of actions are:

— A_1 = Activating other intensification agents;

— A_2 = Activating the reduced perturbation behavior of the perturbation agent;

— A_3 = Activating the strong perturbation behavior of the perturbation agent.

Each condition promotes a certain action. Thus, when the condition C_1 is met, one pursues an intensified search by activating other intensification (A_1). When C_2 (resp. C_3) is satisfied, the search needs to be diversified by triggering the perturbation agent with reduced (resp. strong) behavior (A_2 or A_3). The choice of the most suitable action is controlled by the corresponding decision matrix of each intensification agent.

2.6 Perturbation agent

The perturbation agent is triggered by intensification agents under specific conditions (C_2 and C_3 of section 2.5.1). Basically, this agent disrupts a solution sent by an intensification agent. The disruption is achieved by either a reduced perturbation behavior or strong perturbation behavior. Then, the resulting solution is sent back to the intensification agent which uses the perturbed solution as its new current solution. Since the perturbation agent can be called many times, it can have several life cycles.

2.6.1 Reduced perturbation technique

The perturbation agent can be triggered when an intensification agent observes a slight search stagnation (condition C_2 of section 2.5.1). From the solution received from the intensification agent, the perturbation agent performs a number of random moves to generate a new solution.

2.6.2 Strong perturbation technique

The second case where the perturbation agent can be activated is when it receives a request for strong perturbation from an intensification agent. The perturbation agent then employs the common archive of elite solutions to create a new solution.

2.7 Crossover agents

Crossover agents are agents for diversification. Each crossover agent performs a different crossover operation to generate one offspring solution. Offspring solutions are transmitted to the decision-maker agent to be a new starting point for the search process. In both cases, parents are selected from the common archive.

Algorithm 2.3. Intensification agent behavior

Require: Solution S_0 received from decision-maker agent, parameters: maximum iterations $iteration\_max$, improvement threshold $interval$, consecutive non-improving iterations max_opt_LS

Ensure: A best solution S_{best_LS}

$S \leftarrow S_0$ {S is the current solution found by each intensification agent}
$Q \leftarrow 0$ {Q is the decision matrix, section 2.5.1}
$opt = 0$ {opt is the counter for consecutive non-improving local optima}
$S_1 \leftarrow S_0$ {S_1 records the solution obtained in generation $iteration$-1}
while $iteration \leq iteration\_max$ **do**
  $V \leftarrow$ Generate the best solution by exploring an iteration of a local search
  if $F(S) \leq F(S_{best\_LS})$ **then**
    $S_{best\_LS} \leftarrow S$
  else
    $opt = opt + 1$
  end if
  if $(F(S) - F(S_1)) < interval$ or $opt = max\_opt\_LS$ **then**
    {The intensification agent is stagnating and needs helps from another intensification agent or the perturbation agent}
    $S_{perturbed} \leftarrow \emptyset$ {$S_{perturbed}$ is the solution received from another agent, initialized to empty}
    Update Q {Update the decision matrix based on the improvement of the current solution, sections 2.5.1 & 2.2.1}
    $Action\_exchange \leftarrow$ Select the agent to activate based on Q
    if $Action\_exchange$ = Triggering perturbation agent with weak behavior **then**
      Activate the perturbation agent with reduced behavior and send it solution S
    end if
    if $Action\_exchange$ = Triggering the perturbation agent with strong behavior **then**
      Activate the perturbation agent with strong behavior
      $opt \leftarrow 0$
    end if
    if $Action\_exchange$ = Triggering other intensification agents **then**
      Request the best current solution of other intensification agents
    end if
    Let $S_{perturbed}$ be the best new solution received from any of the above exchange
    if $S_{perturbed} \neq \emptyset$ **then**
      $S \leftarrow S_{perturbed}$
    else
      Block this agent {This agent waits for a solution from other agents }
    end if
  else
    $S_1 \leftarrow S$ {intensification agent continues its exploration without exchanging information}
  end if
  $iteration = iteration + 1$
end while
Return S_{best_LS} to decision-maker agent

2.8 Discussion

This chapter presented a multi-agent based optimization method for solving combinatorial optimization problems. Our method is able to select if the search needs to be intensified or diversified. This is realized by a group of agents which concurrently explore the search space but cooperate to coordinate the search and improve their behaviors. These agents are reinforced by a learning mechanism, in order to know which techniques to trigger. The intensification agents of MAOM-COP can as well be related to the VNS and the ILS methods because they use several neighborhood strategies and different perturbation techniques throughout the optimization process. The change of neighborhoods offers an adaptive mechanism for tracking the optimum in the search space. In addition, switching between several perturbation strategies aims to escape poor optima. In contrast with these two metaheuristics and due to distributed and parallel behaviors of MAOM-COP, intensification agents and perturbation agent can exchange solutions during the search.

Like memetic algorithms, MAOM-COP integrates crossover agents. Crossover agents are triggered only when a local optimum is reached. These agents can be considered as another technique of diversification that directs the search towards more promising regions of the search space. Among multi-agent based optimization methods, MAOM-COP is the first one that considers all these techniques which cover the optimization process. Other existing methods (like (Jędrzejowicz & Wierzbowska, 2006)), use only one metaheuristic in each agent and there is no efficient exchange with them.

2.9 Conclusion

Our work is motivated by appealing features of a MAS which could be advantageously used to elaborate intelligent computing systems. Compared with existing studies on the COP, this work has the following main contributions: it integrates a set of collaborative agents (tabu search agents, crossover agents, perturbation agent) which are managed dynamically by a distributed model to ensure a suitable balance of intensification and diversification of the given search space. Decision making is based on reinforcement learning which is used to adjust the probability of applying dedicated actions to trigger specific agents under specific conditions. The proposed approach is generic and could be adapted to design distributed intelligent systems for complex search problems.

MAOM-COP can be applied to different combinatorial optimization problems. In the next chapters, we will see that only neighborhood relations for intensification agents, evaluation functions, perturbation moves for perturbation agent and crossover operators for crossover agents, will be changed according to the considered problem. The learning mechanism, used to indicate which agents to activate, is the same for all problems. This includes the update of decision matrices and the definition of the conditions and actions in these matrices.

3

A Multi-Agent based Optimization Method for the Quadratic Assignment Problem

In this chapter, we apply the proposed method explained in chapter 2 to the Quadratic Assignment Problem (QAP). We will present the QAP and the most effective algorithms for this problem. Then, we will describe MAOM-QAP, i.e., the adaptation of MAOM-COP to the QAP by describing the behaviors of the agents. MAOM-QAP is evaluated using various benchmark instances. The comparison with the current state of the art approaches, shows that the proposed algorithm performs well in terms of solution quality. The content of this chapter is presented in (Sghir & al., 2015b).

Contents

3.1 Problem definition

The Quadratic Assignment Problem (QAP) is known as one of the most popular combinatorial optimization problems with a number of practical applications like backboard wiring in electronics, analysis of chemical reactions for organic compounds, design of typewriter keyboards balancing turbine runners (Burkard, 1991; Duman & Or, 2007).

Given a flow f_{ij} from facility i to facility j for all i, j in $\{1, 2, ...n\}$ and a distance d_{ab} between locations a and b for all a, b in $\{1, 2, ...n\}$, the QAP is to assign the set of n facilities to the set of n locations while minimizing the sum of the products of the flow and distance matrices. Let Π be the set of the permutation functions π: $\{1, 2, ...n\} \rightarrow \{1, 2, ...n\}$. The QAP is mathematically formulated as follows:

$$Minimize \quad \pi \in \Pi \quad F(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\pi_i \pi_j} \tag{3.1}$$

3.2 State of the art approaches for the QAP

In this section, we present a brief summary of some of the most representative heuristic algorithms for the QAP. These algorithms will be used as reference algorithms for our computational study. Note that none of these QAP approaches can be considered as the most effective method for all QAP benchmark instances, due to the differences in structures of the benchmark instances.

The robust tabu search (Ro-TS) algorithm, proposed by (Taillard, 1991), is an early and influential heuristic. Ro-TS employs the swap move which exchanges two elements of a solution (a permutation). The tabu list forbids the reverse exchange of a swap move during the next h iterations. The tabu tenure h varies randomly within a given interval. The most important new feature introduced in Ro-TS is that a complete swap neighborhood is explored in $O(n^2)$ instead of $O(n^3)$ as in previous algorithms. We use this technique in our algorithm.

The improved hybrid genetic algorithm (IHGA) is proposed by Misevicius (Misevicius, 2004). IHGA integrates a robust local improvement procedure and a new optimized crossover. The optimized crossover uses M runs of an uniform crossover to produce a child that has the best fitness value. The offspring is then improved with a tabu search procedure and a solution reconstruction procedure. The reconstruction is attained by performing a number of random swaps. IHGA uses also a shift mutation, which simply consists in shifting all the items in a wrap-around fashion by a predefined number of positions.

Misevicius (Misevicius & al., 2006) later proposed an iterated tabu search (ITS). It applies a traditional tabu search. When it reaches local optima, it triggers a perturbation phase in order to escape the attained local optimum. The found solution becomes a new starting point for the basic TS procedure. The perturbation mechanism adaptively changes the number of random perturbation moves in some interval.

The particular population-based iterated local search (PILS) proposed by (Stützle, 2006) is an extension of iterared local search (ILS). The algorithm applies the don't look bit strategy, inspired from the local search algorithms for the TSP. When a local optimum is attained, ILS executes a perturbation move that consists of exchanging k randomly chosen items. In PILS, the population contains p solutions and in each iteration q new solutions are generated. The new population of p solutions is created from the p former solutions and the q new solutions.

The cooperative parallel tabu search algorithm (CPTS), which is proposed by (James & al., 2009), applies the parallel execution on multiple processors based on several tabu search (TS) runs. The TS procedure is the same as Ro-TS (Taillard, 1991), but it uses different stopping conditions and the tabu tenure parameters for each processor participating in the algorithm. The cooperation and information exchange between TS processes are realized with the help of a global reference set.

The Breakout Local Search (BLS) proposed by (Benlic & Hao, 2013) is based on a local search phase and a dedicated perturbation phase. The local search phase aims to reach new local optima, while the perturbation phase is used to discover new promising regions. The perturbation mechanism of BLS dynamically determines the number of perturbation moves and adaptively chooses between two types of moves of different intensities depending on the current search state. The types of perturbation are a guided perturbation using a tabu list and a random perturbation. BLS is later integrated into the memetic search framework in (Benlic & Hao, 2015). BMA combines BLS as local optimizer, a crossover operator, a pool updating strategy, and an adaptive mutation mechanism. BMA outperforms its local search component (BLS).

Our proposed algorithm distinguishes itself by its multi-agent based distributed computing model which is described in the next section.

3.3 A multi-agent based optimization method for the QAP (MAOM-QAP)

In this section, we present the Multi-Agent based Optimization for the QAP (MAOM-QAP), which is an adaptation of our generic MAOM-COP method to the QAP. We consider the following agents: the decision-maker agent, two tabu search agents which are intensification agents, the perturbation agent and two crossover agents. In particular, we show the problem dependent ingredients such as the neighborhood relation manipulated by tabu search agents, the crossover operators used by crossover agents and the perturbation moves explored by perturbation agent.

3.3.1 Decision-maker agent

The Decision-maker agent selects other agents to trigger based on its decision matrix (section 2.2.1) and according to the specific condition (section 2.4.1). If other agents (tabu search agents or crossover operator agents) are trigged, the decision-maker agent waits high-quality solutions received from these agents, to record them in the shared memory (archive) (Algorithm 2.2). In addition, it generates the initial solution, in order to start the search and sends it to the appropriate agents. For the QAP, this solution consists in a simple random initial facility location assignment.

3.3.2 Tabu search agents

The tabu search agents manage the intensification search of MAOM-QAP. Each tabu search agent uses a tabu search algorithm applying two different strategies to explore the swap-based neighborhood (Algorithm 2.3). Based on their decision matrix (section 2.2.1) and according to the corresponding condition (section 2.5.1), they can request helps from another alive tabu search agent or the perturbation agent. At the end of each tabu search agent run, the best permutation found by the agent is sent to the decision-maker agent. Below, we define the two neighborhood exploration strategies employed by these agents.

3.3.2.1 Neighborhood

As explained in the introduction, a candidate QAP solution can be conveniently represented by a permutation π of $\{1, 2, ...n\}$ where π_i is the facility assigned to location i. Let $swap(i, j)$ be a move operator which exchanges the facilities located at i and j. Given a candidate solution π, let $\pi' = \pi \oplus swap(i, j)$ be the neighboring solution of π obtained by exchanging the facilities π_i and π_j of locations i and j. Then $N(\pi) = \{\pi' : \pi' = \pi \oplus swap(i, j), i, j \in \{1, 2, ...n\}, i \neq j\}$ is the set of neighboring solutions induced by the swap operator. To assess the relative quality of a neighboring solution π', i.e., the cost variation $\delta(\pi, i, j) = F(\pi') - F(\pi)$ between π and π' (also called the move gain of $swap(i, j)$), we use the incremental technique proposed in (Taillard, 1991) which can be achieved in $O(n)$ in the worst case.

3.3.2.2 Neighborhood exploration strategies

Given this neighborhood, our tabu search agents employ two different strategies to explore the neighboring solutions. Let π be the incumbent solution and $N(\pi)$ its neighborhood. Our first tabu search agent examines the whole neighborhood $N(\pi)$ (in $O(n^3)$) and retains the best neighboring solution which becomes the new incumbent solution. As such, this tabu search agent realizes a highly aggressive exploitation of the neighborhood, ensuring thus an intensified search. Our second tabu search agent operates slightly differently in two stages. First, it picks at random a location i. Then it seeks the best location j which leads to the highest $swap(i, j)$ move gain. This neighborhood exploration strategy, which is achieved in $O(n^2)$, leads to a less aggressive search. Yet, given the random choice of one of the two locations to be exchanged, this strategy provides the tabu search agent with an intensified search while ensuring some degree of diversification at the same time.

3.3.2.3 Tabu list

Each tabu search agent uses a traditional tabu list to prevent the search from revisiting a previously encountered solution. Each time a facility x_i is displaced from location i to a new location by a $swap(i, j)$ move, x_i is forbidden to move back to location i during the next h iterations. The iterations h is dynamically determined by $h = \alpha \times F(S) + rand(10)$, where $rand(10)$ takes a random number in $[1, ..., 10]$ and α is set to 0.09.

3.3.3 Perturbation agent

When tabu search agents need help from the perturbation agent under specific conditions (C_2 and C_3 of section 2.5.1), they decide to trigger the latter agent. This agent disrupts a solution sent by a tabu search agent. Two parallel behaviors, that are reduced perturbation behavior and strong perturbation behavior, are realized. The resulting solution is used then by the tabu search agent as its new current solution.

3.3.3.1 Reduced perturbation technique

In order to solve a slight search stagnation (condition C_2 of section 2.5.1), the tabu search agents can activate the perturbation agent with a reduced behavior. The last agent applies a number of random swap moves to generate a new solution, started from the solution received from tabu search agents. This is achieved by exchanging the locations of two facilities chosen randomly. Also, the number of perturbation swap moves is chosen randomly between 1 and $n/2$ (n being the number of facilities).

3.3.3.2 Strong perturbation technique

When a strong search stagnation (optimum) is encountered, the perturbation agent can receive a request for strong perturbation from a tabu search agent. The perturbation agent uses the common archive of elite solutions to create a new solution. From this archive, the perturbation agent extracts the number of occurrences of each facility i assigned to location x_i. Then, each facility i is assigned to the location having the smallest occurrence number. Additional data structures are employed to avoid the creation of the same solution for future calls to the perturbation agent.

3.3.4 Crossover agents

The decision-maker agent can trigger crossover agents, when it observes that the search is trapped into a deep optimum. For the QAP, we have two crossover agents each one performing a different crossover operation to generate one offspring solution. The two offspring solutions are sent to the decision-maker agent to be a new starting point for the search process. In each crossover agent, the parents are chosen randomly from the common archive. Each crossover agent applies one of the following crossover operators:

— The first operator consists in blending uniformly information from the parents. Given two selected parents, the crossover operator builds one offspring solution by alternatively transmitting location-facility assignments from the parents. Specifically, starting with the parent having the smallest objective value, the first crossover agent transmits the facility of the first location (i.e., with index one) to the first location of the child and then removes the assigned facility from both parents. For the second location of the child, it switches to the other parent and transmits the facility (which may be empty) of the second location (i.e., with index two) to the child. Then, this agent goes back to parent one and repeats this process until reaching the last location. Finally, the unassigned facilities of the offspring are affected to a location randomly chosen among the set of the free locations.

— The second crossover operator has the same idea of the first crossover operator, only the first $z << n$ (a parameter) location-facility assignments of each parent are transmitted to the offspring solution. The crossover agent starts from the parent who has the smallest objective value to build the child. It copies the z first location-facility assignments of this parent into the child. Then, it extracts from the other parent, the next z location-facility assignments and copies them to the child from the $z + 1$ locations. Finally, each unassigned facility is affected to a random unassigned location.

Figures 3.1 and 3.2 provide illustrating examples for these two crossover operators.

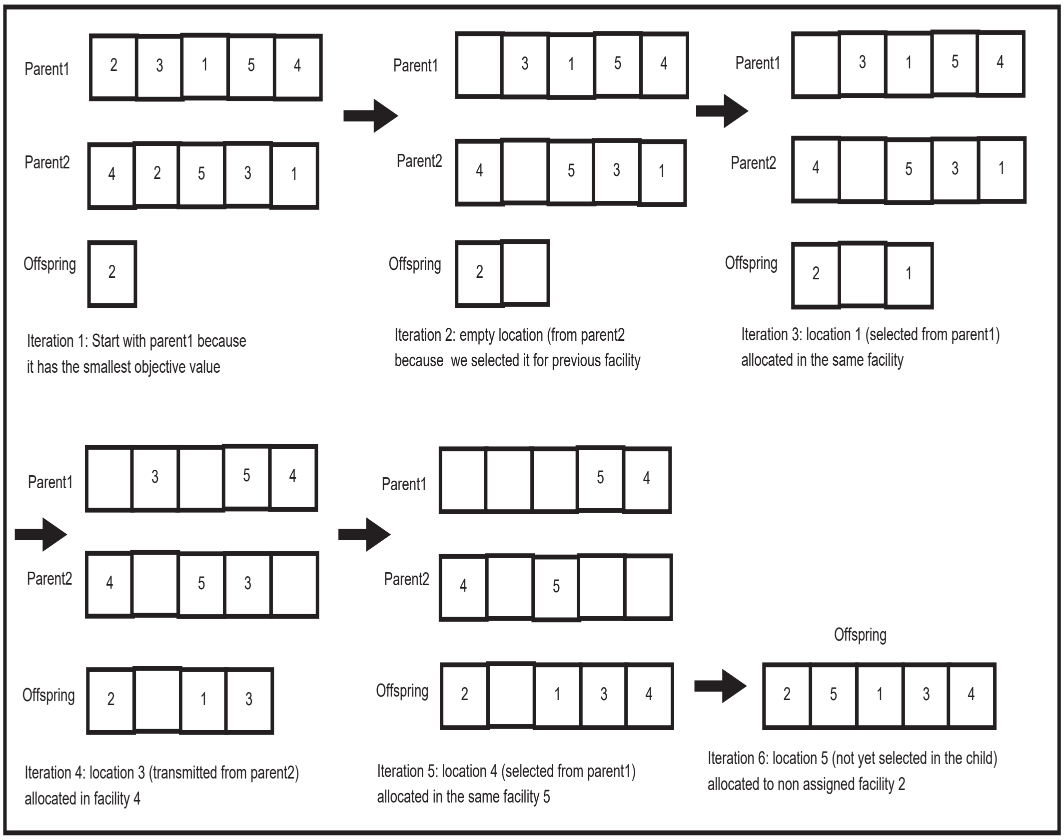

Figure 3.1 – An example for the first operator crossover used by the first crossover agent in MAOM-QAP

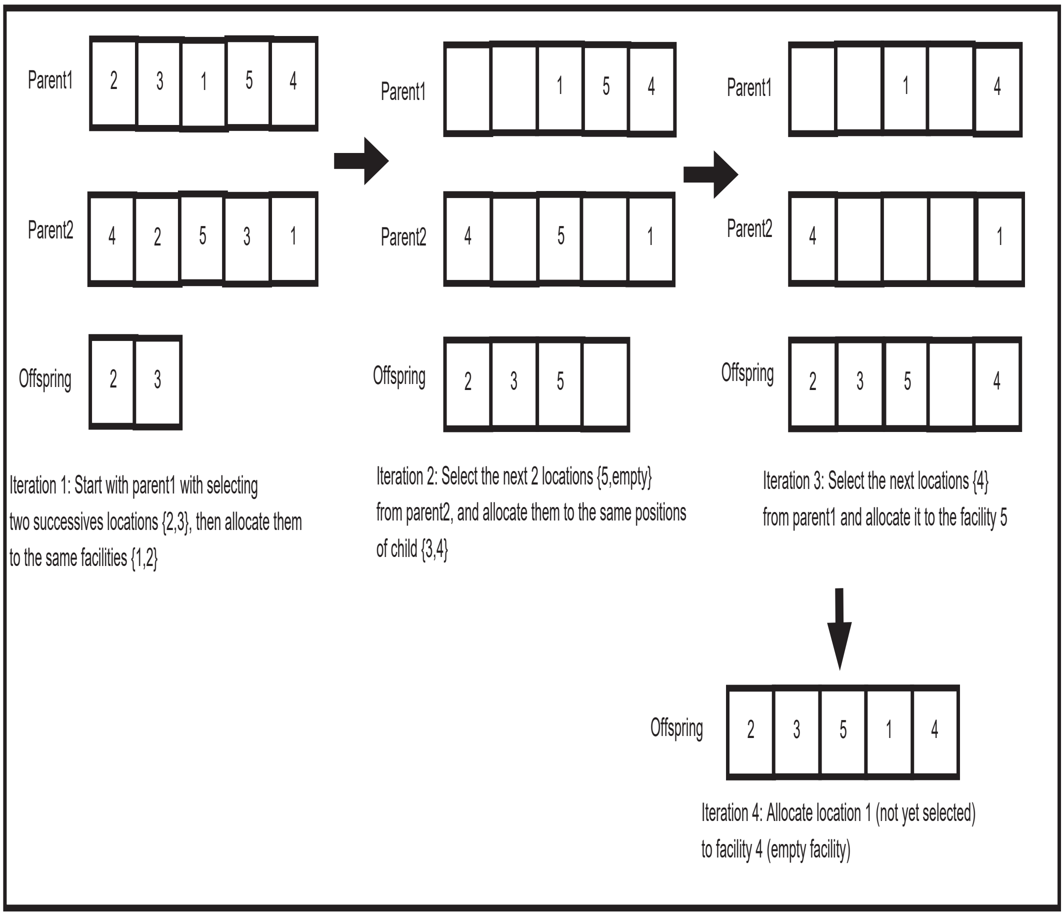

Figure 3.2 – An example for the second operator crossover used by the second crossover agent in MAOM-QAP(z=2 in this example)

3.4 Experimentation

This section presents experimental results of MAOM-QAP. We campare it with best-known algorithms from the literature, then we give the impacts of the composing agents in terms of solution quality.

3.4.1 Experimental results

MAOM-QAP was implemented in Java using the multi-agent platform Jade. The program was run on a computer with a Core I5 2.5 GHz, 8GB of RAM. To assess MAOM-QAP, tests were realized on various benchmark instances from the QAPLIB (`http://www.seas.upenn.edu/qaplib/inst.html`). The instances size n varies from 12 to 150 (indicated in the instances name).

The QAPLIB archive contains 135 instances that can be divided into four types:

1. Type I: Real-life instances obtained from practical applications;
2. Type II: Unstructured and random instances for which the distance and flow matrices are randomly generated based on a uniform distribution;

3. Type III: Randomly generated instances with structure that is similar to that of real-life instances;
4. Type IV: Instances in which distances are based on the Manhattan distance on a grid.

Following (Benlic & Hao, 2015), we focus on the set of 21 most challenging instances of types II-IV (the remaining 114 instances including all the real-life instances of Type I are easy and are not included in the paper).

We adjusted the parameters of the proposed algorithms by an experimental study. They depend on the type of the problem. The number of iterations for each local search agent ($iter\_max$) was fixed to 1000. The parameter $interval$ that evaluates the improvement of the solution was fixed to 10000 for the decision-maker agent and the tabu search agents. The parameters g_0, g_1, g_2, q_3, q_4 and q_5, which are the numbers of generations responsible for controlling the improvement of the search process (presented in section 2.4.1 and section 2.5.1), were fixed respectively to 2, 10, 2, 20, 20 and 25. The parameter rate μ used in updating the decision matrices was fixed to 0.5. The stopping condition is the elapsed time which we set to 12 hours for all the instances of size $n < 100$, and to 24 hours for the large instances of size $n >= 100$. The best-known solutions can be attained before these time limits.

We compare our MAOM-QAP to seven best-known algorithms from the literature cited in the introduction of the paper.

— Improved hybrid genetic algorithm (IHGA) (Misevicius, 2004);
— Iterated tabu search (ITS) (Misevicius & al., 2006);
— Population-based iterated local search (PILS) (Stützle, 2006);
— A hybrid genetic tabu search algorithm (MRT60) (Drezner, 2008);
— Cooperative parallel tabu search (CPTS) (James & al., 2009);
— The Breakout local search (BLS) (Benlic & Hao, 2013);
— The population-based Memetic Algorithm (BMA) (Benlic & Hao, 2015).

Our main purpose of this assessment is to compare our results with the *best-known results* ever reported by any existing algorithms of the literature. Note that these best-known results, as well as those of the reference algorithms, have been achieved by different algorithms under various conditions (different stop conditions, computing platforms etc). As a result, the comparisons with the existing methods are included only for indicative information.

Table 3.1 reports our computational results along with those of the seven reference algorithms on the unstructured instances (type II) and real-life like instances (type III). The second column 'BKS' presents for each instance the best-known objective value ever reported in the literature. For each algorithm, column $\bar{\delta}$ shows the percentage deviation of the average solution, obtained with the considered algorithm over a certain number of trials, from the best-known solution. If known, the success rate for reaching the best-known solution over several trials is given in parentheses next to the value of the $\bar{\delta}$. The CPU time (in minutes) is only given for indicative purposes. The last row indicates the averaged information.

For the unstructured instances (type II), MAOM-QAP finds the best-known solution for 7 out of the 9 instances like other algorithms. We show in Table 1 only the results of 5 most difficult instances because the 4 other instances are easy to solve. The average deviation $\bar{\delta}$ from the best-known solution is 0.341. As to the real-life like instances (type III), MAOM-QAP can attain the best-known solution for all the instances, except for the largest instance tai150b. The difference of deviation $\bar{\delta}$ between them is 0.015 over the 5 instances, which matches the performance of BLS and CPTS.

Table 3.2 presents our computational results along with those of the seven reference algorithms on the instances with grid distances (type IV). We observe that MAOM-QAP is able to reach the best-known results for 14 out of the 15 instances with a deviation $\bar{\delta}$ of 0.001 which is the best result with CPTS and BMA. As to the computing times, MAOM-QAP is more computationally expensive due to the perturbation agent whichse impact will be presented in the next section.

3.4.2 Impact of perturbation agent on MAOM-QAP

In the MAOM-QAP algorithm, we use two different perturbation techniques leading to either a reduced or strong perturbation behavior. In this section, we perform an experiment to assess the usefulness of the perturbation agent. For this, we compare MAOM-QAP and MAOM-QAP with its perturbation agent disabled by running them under the same condition as specified in section 3.4 and report the comparative results in Tables 3.3 and 3.4 where MAOM-QAP' is MAOM-QAP without the perturbation agent. These tables disclose that on all the benchmarks, MAOM-QAP without the perturbation agent fails to reach the best-known results of 21 instances. These results show that the perturbation agent reinforces the search performance of MAOM-QAP.

3.4.3 Impact of crossover agents on MAOM-QAP

In order to show the relative effectiveness of the crossover agents which represent a technique of diversification in our algorithm, we compare MAOM-QAP with and without the crossover agents. As before, we run both algorithms under the same condition as specified in section 3.4 and report the comparative results in Tables 3.5 and 3.6 where MAOM-QAP" is MAOM-QAP without the crossover agents. We observe that the algorithm without the crossover agents (MAOA-QAP") performs much worse since it can find the best-known results for only 4 out of the 21 instances. So, we can conclude that the crossover agents are indispensable for the performance of our MAOM-QAP algorithm.

Table 3.1 – Results of MAOM-QAP compared to some of the best performing QAP approaches on unstructured instances (type II) and on Real-life like instances (type III). The times are given in minutes.

Problem	BKS	MAOM-QAP		BMA		BLS		CPTS		ITS		IHGA	
		% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)
Random instances (Type II)													
tai40a	3139370	0.099(2)	83.5	0.059(2)	8.1	**0.022**(7)	38.9	0.148(1)	3.5	0.210(1)	0.8	0.209(1)	1.4
tai50a	4938796	0.320 (1)	135.2	**0.131**(2)	42.0	0.157(2)	45.1	0.440(0)	10.3	0.373(0)	3.0	0.262(0)	5.0
tai60a	7205962	0.385 (2)	178.1	**0.144**(2)	67.5	0.251(1)	47.9	0.476(0)	26.4	0.330(1)	9.7	0.583(0)	12
tai80a	13499184	**0.426**(0)	225	**0.426**(0)	65.8	0.517(0)	47.3	0.691(0)	94.8	0.494(0)	25.0	0.756(0)	53.3
tai100a	21052466	0.470 (0)	288	**0.405**(0)	44.1	0.430 (0)	39.0	0.589(0)	261.2	0.427(0)	60.0	0.606(0)	200.0
Average		0.341	107.6	0.233	45.5	0.275	43.6	0.469	79.2	0.367	19.2	0.483	54.3
Real-life like instances (Type III)													
tai50b	458821517	**0.000**(10)	14.3	**0.000**(10)	1.2	**0.000**(10)	2.8	**0.000**(10)	13.8	**0.000**(10)	0.9	**0.000**(10)	0.3
tai60b	608215054	**0.000**(10)	38.2	**0.000**(10)	5.2	**0.000**(10)	5.6	**0.000**(10)	30.4	**0.000**(10)	2.2	**0.000**(10)	0.7
tai80b	818415043	**0.000**(10)	62.7	**0.000**(10)	31.3	**0.000**(10)	11.4	**0.000**(10)	110.9	**0.000**(10)	5.8	**0.000**(10)	2.5
tai100b	1185996137	**0.000**(10)	91.2	**0.000**(10)	13.6	**0.000**(10)	16.0	0.001(8)	241.0	**0.000**(9)	23.3	**0.000**(10)	7.3
tai150b	498896643	0.077(0)	9982	**0.060**(1)	78.1	0.100(0)	80.5	0.076(0)	7377.8	0.100(1)	60.0	0.111(2)	38.3
Average		0.015	1030.8	0.012	25.9	0.020	23.3	0.015	1554.8	0.020	18.4	0.022	9.8

Table 3.2 – Comparative results between MAOM-QAP and some of the best performing QAP approaches on grid-based (type IV) instances. The times are given in minutes

Problem	BKS	MAOM-QAP		BMA		BLS		CPTS		MRT60	
		% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)	% $\bar{\delta}_{avg}$	t(m)
sko72	66256	**0.000**(10)	63.3	**0.000**(10)	3.5	**0.000**(10)	4.1	**0.000**(10)	69.6	**0.000**(10)	19.9
sko81	90998	**0.000**(10)	208.5	**0.000**(10)	4.3	**0.000**(10)	13.9	**0.000**(10)	121.4	**0.000**(10)	31.9
sko90	115534	**0.000**(10)	256.4	**0.000**(10)	15.3	**0.000**(10)	16.6	**0.000**(10)	193.7	**0.000**(10)	48.5
sko100a	152002	**0.000**(10)	321	**0.000**(10)	22.3	0.001(9)	20.8	**0.000**(10)	304.8	**0.000**(10)	73.6
sko100b	153890	**0.000**(10)	322.2	**0.000**(10)	6.5	**0.000**(10)	10.8	**0.000**(10)	309.6	**0.000**(10)	73.6
sko100c	147862	**0.000**(10)	324.8	**0.000**(10)	12.0	**0.000**(10)	15.5	**0.000**(10)	316.1	**0.000**(10)	73.6
sko100d	149576	**0.000** (9)	330	0.006(9)	20.9	0.001(5)	38.9	**0.000**(10)	309.8	**0.000**(10)	73.6
sko100e	149150	**0.000**(1r 0)	343.3	**0.000**(10)	11.9	**0.000**(10)	42.5	**0.000**(10)	309.1	**0.000**(10)	73.6
sko100f	149036	**0.000**(10)	320	**0.000**(10)	23.0	**0.000**(10)	17.3	0.003(4)	310.3	0.000(9)	43.5
wil100	273038	**0.000**(10)	355	**0.000**(10)	14.5	**0.000**(10)	18.9	**0.000**(10)	316.6	**0.000**(10)	73.6
tho150	8133398	0.011 (0)	523	0.008(3)	416.4	0.023(1)	268.8	0.013(0)	1991.7	**0.003**(3)	1223.6
Average		0.001	229	0.001	50.1	0.002	42.6	0.001	413.9	0.000	164.5

Table 3.3 – Impact of perturbation agent on MAOM-QAP on the unstructured instances (type II) and on Real-life like instances (type III): MAOM-QAP is MAOM-QAP with the perturbation agent and MAOM-QAP’ is MAOM-QAP without the perturbation agent

Problem	BKS	MAOM-QAP		MAOM-QAP’	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
tai40a	3139370	**0.099(2)**	83.5	4.556(0)	0.00
tai50a	4938796	**0.320(1)**	135.2	7.64(0)	20.3
tai60a	7205962	**0.385(2)**	178.1	4.71(0)	22.1
tai80a	13499184	**0.426(0)**	225	3.898(0)	30.5
tai100a	21052466	**0.470(0)**	288	4.74(0)	28.45
Average		**0.341**	107.6	5.419	11.26
tai50b	458821517	**0.000(10)**	14.3	13.587(0)	0.00
tai60b	608215054	**0.000(10)**	38.2	8.758(0)	0.00
tai80b	818415043	**0.000(10)**	62.7	11.823(0)	0.00
tai100b	1185996137	**0.000(10)**	91.2	14.182(0)	9.16
tai150b	498896643	**0.077(0)**	9982	13.542(0)	83.1
Average		**0.015**	1030.8	11.447	9.22

Table 3.4 – Impact of perturbation agent on MAOM-QAP on grid-based (type IV) instances. MAOM-QAP’ is MAOM-QAP without the perturbation agent

Problem	BKS	MAOA-QAP		MAOA-QAP’	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
sko72	66256	**0.000(10)**	63.3	3.156(0)	0.00
sko81	90998	**0.000(10)**	208.5	4.581(0)	12
sko90	115534	**0.000(10)**	256.4	5.789(0)	11.2
sko100a	152002	**0.000(10)**	321	4.865(0)	15
sko100b	153890	**0.000(10)**	322.2	5.889(0)	13.8
sko100c	147862	**0.000(10)**	324.8	4.19(0)	14.1
sko100d	149576	**0.000(10)**	330	3.86(0)	30
sko100e	149150	**0.000(10)**	343.3	4.245(0)	25.4
sko100f	149036	**0.000(10)**	320	3.79(0)	15.4
wil100	273038	**0.000(10)**	355	5.228(0)	18.1
tho150	8133398	**0.011(0)**	523	3.699(0)	45
Average		**0.001**	229	4.143	13.33

Table 3.5 – Impact of crossover agents on MAOM-QAP on grid-based (type IV) instances. MAOM-QAP" is MAOM-QAP without the crossover agents

Problem	BKS	MAOM-QAP		MAOM-QAP"	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
sko72	66256	**0.000(10)**	63.3	1.99(0)	4.5
sko81	90998	**0.000(10)**	208.5	2.457(0)	12.4
sko90	115534	**0.000(10)**	256.4	2.75(0)	15.7
sko100a	152002	**0.000(10)**	321	2.486(0)	12.3
sko100b	153890	**0.000(10)**	322.2	1.25(0)	11.8
sko100c	147862	**0.000(10)**	324.8	3.724(0)	13.78
sko100d	149576	**0.000(10)**	330	2.785(0)	25.89
sko100e	149150	**0.000(10)**	343.3	1.42(0)	15.9
sko100f	149036	**0.000(10)**	320	3.75(0)	22.4
wil100	273038	**0.000(10)**	355	2.15(0)	28.2
tho150	8133398	**0.011(0)**	523	3.489(0)	34
Average		**0.001**	229	2.56	31.5

Table 3.6 – Impact of crossover agents on MAOM-QAP on unstructured instances (type II) and on Real-life like instances (type III): MAOM-QAP is MAOM-QAP with crossover agents and MAOM-QAP" is MAOM-QAP" without crossover agents

Problem	BKS	MAOM-QAP		MAOM-QAP"	
		$\bar{\delta}$	t(m)	$\bar{\delta}$	t(m)
tai40a	3139370	**0.099(2)**	83.5	2.8(0)	2.08
tai50a	4938796	**0.320(1)**	135.2	3.78(0)	15.12
tai60a	7205962	**0.385(2)**	178.1	2.75(0)	18.4
tai80a	13499184	**0.426(0)**	225	3.82(0)	20.1
tai100a	21052466	**0.470(0)**	288	3.28(0)	20.5
Average		**0.341**	107.6	2.87	8.46
tai50b	458821517	**0.000(10)**	14.3	4.78(0)	0.00
tai60b	608215054	**0.000(10)**	38.2	5.96(0)	0.00
tai80b	818415043	**0.000(10)**	62.7	5.2(0)	5.2
tai100b	1185996137	**0.000(10)**	91.2	5.11(0)	8.4
tai150b	498896643	**0.077(0)**	9982	6.45(0)	23
Average		**0.015**	1030.8	5.29	3.66

3.5 Conclusion

In this chapter, we introduced a multi-agent algorithm for the Quadratic Assignment Problem based on different techniques of intensification and diversification. The decision-maker agent is the central agent which decides the most suitable agent to activate and maintains a shared memory to record the elite solutions discovered during the search. Its decisions are influenced by a learning-based probabilistic strategy which dynamically adjusts the application probability of a particular action under a specific condition. On the other hand, the tabu search agents are introduced to ensure an intensified examination of specific search zones while the perturbation agents and crossover agents are used to diversify the search.

Our computational study shows that the proposed approach performs well on the tested benchmark instances in terms of solution quality.

4

A Multi-Agent based Optimization Method for the Graph Coloring Problem

In this chapter, we apply the proposed method to the Graph Coloring Problem (GCP). We will start with the problem definition and a brief review of popular graph coloring algorithms. Then, we will define the agents of MAOM-GCP, which adapts the MAOM-COP to the GCP. The proposed algorithm will be evaluated on graph coloring benchmarks. The comparative study shows that MAOM-GCP is able to reach the best known solution of several instances. The content of this chapter is published in (Sghir & al., 2015a)

Contents

4.1 Problem definition

Given an undirected graph $G = (V, E)$ with vertices set V, edges set E and an integer k. A k-coloring of G is represented by $S = V_1, V_2, ..., V_k$. The value $V(x)$ of a vertex x defines the color of x. The vertices with color r ($1 < r < k$) correspond to a color class. If two adjacent vertices x and y are colored with the same color r, vertices x and y are declared as conflicting vertices, the edge $[x, y]$ is a conflicting edge, and r is a conflicting color. A coloring graph with no conflicting edge is called a legal k-coloring.

The graph coloring problem (GCP) aims at determining the smallest integer k (called chromatic number of G, $\aleph(G)$) such that there exists a legal k-coloring of G. There are four different strategies to represent the search space: legal strategy, penalty strategy, k-fixed partial legal strategy, k-fixed penalty strategy. We adopt the k-fixed penalty strategy like many studies. In this strategy, we fix the number k of colors and we accept all possible k-colorings which can be legal or illegal solutions. Given a k-coloring $S = \{V_1, V_2, ..., V_k\}$, the evaluation function f consists in calculating the conflict number induced by S such that:

$$f(S) = \sum_{u,v \in E} \delta_{uv} \tag{4.1}$$

where

$$\delta_{uv} = \begin{cases} 1, & \text{if } \mathrm{u} \in V_i \ , v \in V_j \ , \ i = j \\ 0, & \text{otherwise} \end{cases}$$

Based on this function, we seek to minimize its value. So, in order to find a legal k-coloring S, the evaluation function f(S) = 0.

The GCP can be associated with a variety of real-world applications, such as the frequency assignment (Smith & al., 1998), the satellite range scheduling (Zufferey & al., 2008), the crew scheduling (Gamache & al., 2007), the printed circuit testing (Garey & al., 1976), the timetabling (Burke & al., 2007), the register allocation (DeWerra & al., 1999). The GCP is a well-known NP-complete problem (Garey & Johnson, 1979).

4.2 State of the art approaches for the GCP

In this section, we make a review on popular heuristic algorithms solving the GCP. Greedy search is the first heuristic algorithm for the GCP. The largest saturation degree heuristic (DSATUR) and the recursive largest first heuristic (RLF) (Leighton, 1979) are the most successful algorithms. These algorithms are often employed to generate initial solutions for advanced metaheuristic algorithms.

The GCP can be solved by metaheuristics which can be divided into three types of methods: local search methods, population based methods and hybrid methods.

Several local search methods are applied to GCP, such as simulated annealing (Kirkpatrick & al., 1983), tabu search (Glover, 1986), variable neighborhood search (Mladenovic & Hansen, 1997), iterated local search (Chiarandini & Stützl, 2002) and large scale neighborhood search (Trick & Yildiz, 2007). Hertz and Werra (Hertz & de Werra, 1987) were the first who used tabu search (TS) known as Tabucol to find a solution to the graph coloring problem using k-fixed penalty strategy.

Another complete version of Tabucol is obtained by Dorne and Hao (Dorne & Hao, 1998). Zufferey et al. (Blochliger & Zufferey, 2008) proposed a variant of Tabucol known as Partialcol which integrated a reactive component to adjust the length of the time of the moves. Porumbel et al. (Porumbel & al., 2009) elaborated a new local search algorithm known as position guided tabu search (PGTS) heuristic which adds techniques to avoid local optima. Hertz et al. (Hertz & al., 2008) proposed variable search space (VSS-COL) which alternates different techniques from the three different local search heuristics: Tabucol, Partialcol, and a third tabu search algorithm proposed by (Gendron & al., 2007).

Population based methods are, also, used to solve GCP as genetic algorithm (Holland, 1975), ant colony optimization (Dorigo & al., 1991), particle swarm optimization (Kennedy & Eberhart, 1995).

Hybrid methods are methods which combine different techniques from local search methods and population based methods.
Hao and Galinier (Galinier & Hao, 1999) proposed an hybrid coloring algorithm (HCA) that is based on tabu search and genetic algorithm. HCA integrates the greedy partition crossover (GPX) operator which combines color classes instead of specific color assignments. Lim and Wang (Lim & Wang, 2004) used various metaheuristic which are genetic algorithm, simulated annealing and tabu search. Sivanandam et al. (Sivanandam & al., 2005) elaborated a new permutation based representation of the graph coloring problem. They used a parallelism model for genetic algorithm (PGA) based on Message Passing Interface (MPI) getting three crossover operators. David (Chalupa, 2011) proposed two algorithms for GCP. A multi-agent evolutionary algorithm (MEA) based on multi-agent system where an agent represents a tabu search procedure. The second algorithm is a pseudo reactive tabu search (PRTS) integrating a new online learning strategy. Lu and Hao (Lu & Hao., 2010) proposed a memetic algorithm (MACOL) integrating several distinguished features such as an adaptive multi-parent crossover (AMPaX) operator which is inspired from GPX crossover operator and a distance-and-quality based replacement criterion for pool updating. It uses the Tabucol as a local search algorithm.

In (Titiloye & Crispin, 2011), Olawale et al. proposed a distributed hybrid quantum annealing algorithm. Quantum simulated annealing is a population of agents cooperating to optimize a shared cost function defined as the total energy between them. This algorithm finds better results than those of any known algorithm, for some graphs. Wu and Hao introduced, in (Wu & Hao, 2012), a forward independent set extraction strategy to reduce the initial graph. From the reduced graph, they trigger a backward coloring process which uses extracted independent sets as new color classes for intermediate subgraph coloring. This algorithm provides new upper bounds for other graphs. This method is, then, improved in (Hao & Wu, 2012). Moalic and Gondran (Moalic & Gondran, 2015) proposed a memetic algorithm using tabu search. The main characteristic of this algorithm is to work with a population of only two individuals.

Compared to these popular graph coloring algorithms, MAOM-GCP is the first algorithm which explores the tabu search with other operators in a multi-agent system. We will describe its characteristics in the next section.

4.3 A multi-agent based optimization method for the GCP (MAOM-GCP)

We propose a Multi-Agent based Optimization method for the Graph Coloring Problem (MAOM-GCP) based on our generic MAOM-COP presented in chapter 2. In MAOM-GCP, the agents are the learners who can handle various diversification techniques and other intensification techniques to direct the search towards promising areas. We consider the following agents: the decision-maker agent, two tabu search agents, the perturbation agent and two crossover agents.

4.3.1 Decision-maker agent

We name this agent as decision-maker agent because it is the agent which starts the search cycle of the algorithm by generating an initial solution, then, it decides to select other agents to trigger and finally finishes the search. Based on its decision matrix (section 2.2.1) and according to the state of search (section 2.4.1), it decides whether the search process needs to be intensified or diversified. If other agents are trigged, the decision-maker agent waits them, until it receives best solutions generated by tabu search agents or crossover operator agents, then it maintains all these solutions in an archive.

4.3.1.1 The initial solution

The decision-maker agent creates an initial legal coloring using the greedy largest saturation degree heuristic (DSATUR) (Algorithm 4.1) (Brélaz, 1979). Then, starting with this initial coloring, it randomly displaces the vertices whose color number is higher than the given color number k to a color class between $[1, k]$. This procedure usually leads to an illegal k-coloring which will be repaired by MAOM-GCP.

4.3.1.2 Archive of elite solutions

The decision-maker agent saves the best k-coloring, received from tabu search agents and crossover agents, in an archive. The archive represents a shared memory

Algorithm 4.1. The greedy largest saturation degree heuristic (DSATUR): The saturation degree of a vertex as the number of different colors to which it is adjacent (colored vertices).

Require: Graph G
Ensure: the initial k-coloring S_0
1: **while** All the vertices are not colored **do**
2: Arrange the vertices by decreasing order of degrees.
3: Color a vertex of maximal degree with color i
4: Choose a vertex with a maximal saturation degree. If there is an equality, choose any vertex of maximal degree in the uncolored subgraph.
5: Color the chosen vertex with the least possible (lowest numbered) color.
6: $i = i + 1$
7: return to 4
8: **end while**

between all agents. It is updated by the decision-maker agent with new solutions of good quality.

4.3.2 Tabu search agents

The decision-maker agent can activate two tabu search agents, when it observes that the search process needs to be intensified based on its decision matrix. Each tabu search agent applies a specific strategy based on a particular neighborhood to seek new solutions. During the search, a tabu search agent can exchange its solutions with another alive tabu search agent or with a perturbation agent. These communications depend on a decision matrix, conditions and actions explained in (section 2.2.1) and (section 2.5.1). At the end of each tabu search agent run, the best k-coloring found by each agent is sent to the decision-maker agent. The behavior of the tabu search agent is described in Algorithm 2.3. Below, we define the used neighborhood structures for each tabu search agent.

4.3.2.1 Neighborhoods

A candidate solution for GCP can be generated by changing the color class of vertices. Different modifications lead to different neighborhood structures. In this work, we explore 3 neighborhoods: the vertex neighborhood which changes the color of some conflicting vertices, the class neighborhood which changes the color of some or all vertices of a conflicting color class, and the non-increasing neighborhood which changes the color of some vertices without increasing the total number of conflicting edges.

4.3.2.2 Neighborhood exploration strategies

In MAOM-GCP, we use two complementary neighborhood strategies due to the cooperation act realized by each tabu search agent. One of these strategies, performed by our first tabu search agent, changes the colors of conflicting vertices to produce new k-colorings. This is done by moving a conflicting vertex x from its original color class V_i to the best possible other color class V_j $(i \neq j)$ (this change or move is denoted by (x, i, j)). The new color class for each conflicting vertex x is chosen among those which are not assigned to vertices adjacent to x. Among these color classes found, the best possible color class (in terms of fitness minimization) is selected for the considered conflicting vertex.

Our second tabu search agent uses the same mechanism of selecting the best color class to be assigned to vertices as the first tabu search agent. The difference is that these vertices are not the set of conflicting vertices, but the vertices that are adjacent to conflicting vertices. The tabu search agent chooses the best color class for each vertex belonging to the set of adjacent vertices of conflicting vertices. The best color allocated must not belong to the color classes allocated to conflicting vertices.

For these two neighborhood strategies, tabu search agents evaluate each move using an incremental evaluation technique. This technique consists in maintaining a special data structure that records the move values for each candidate neighborhood move (Dorne & Hao, 1998; Fleurent & Ferland, 1996; Galinier & Hao, 1999). A Δ matrix is used, in which element $\Delta(x, j)$ corresponds to the value gain of changing the current color of node x from color i to color j. Each element can be initialized in $O(|V|)$ operations following this expression:

$$\delta(x, j) = \sum_{y \in x(I_{C_j}(y) - I_{C_i}(y))} \tag{4.2}$$

where for x, $N(x) = y \in V \mid (y, x) \in E$, and I_A is the indicator variable of set A, defined as:

$$I_A(x) = \begin{cases} 1, & \text{if x} \in A \\ 0, & \text{otherwise} \end{cases}$$

Based on expression 4.2, the Δ matrix can be initialized. After that, it can be updated as we describe in Algorithm 4.2.

4.3.2.3 Tabu list

Each tabu search agent uses a tabu list to forbid the reverse moves. When a move (x,i,j) is generated, vertex x is forbidden to move back to color class V_i for the next h iterations (called tabu tenure). The tabu tenure is dynamically determined by $h = f(S) + r(10)$, where $r(10)$ is a random number between 1 and 10 (Galinier & Hao, 1999). The stop condition of each tabu search is a fixed number of iterations.

4.3.3 Perturbation agent

The perturbation agent, triggered by tabu search agents, creates a disturbed k-coloring solution by exploring two types of perturbations. The new k-coloring is then sent to the tabu search agent for further improvement.

Algorithm 4.2. Incremental evaluation technique for updating the Δ matrix

```
for y ∈ N(x) do
    if y ∉ C_i then
        Δ(y, i) = Δ(y, i) − 1
    end if
    if y ∉ C_j then
        Δ(y, j) = Δ(y, j) + 1
    end if
    if y ∈ C_j then
        for c = 1 to k do
            if c ≠ j then
                Δ(y, c) = Δ(y, c) − 1
                Δ(x, c) = Δ(x, c) − 1
            end if
        end for
    end if
    if y ∈ C_i then
        for c = 1 to k do
            if c ≠ i then
                Δ(y, c) = Δ(y, c) + 1
                Δ(x, c) = Δ(x, c) + 1
            end if
        end for
    end if
end for
```

4.3.3.1 Reduced perturbation technique

The reduced perturbation technique can be triggered when a tabu search agent observes a slight search stagnation (condition C_2 of section 2.4.1). From the k-coloring received from the tabu search agent, the perturbation agent makes t moves to create a new solution, where each move changes randomly the color of a conflicting vertex of the incumbent solution. The number t of moves is chosen randomly between 1 and $conf$ (where $conf$ is the number of conflicting vertices).

4.3.3.2 Strong perturbation technique

The strong perturbation technique is performed when a tabu search agent observes deep search stagnation. The perturbation agent uses the shared archive of elite k-colorings to create a new solution. It extracts the number of occurrences of each vertex x colored by each color class V_i. Starting with an uncolored graph, each vertex x is colored with a color class V_i which has the smallest occurrence number. Dedicated data structures are employed to avoid the creation of the same solution for future calls to the perturbation agent.

4.3.4 Crossover agents

When the decision-maker agent decides to activate the crossover agents, two crossover agents are created based on two different crossover operators from the literature: the AMPaX operator (Lu & Hao., 2010) and the GPX operator (Galinier & Hao, 1999). These operators are among the best crossover operators for GCP. The new k-coloring solutions are sent to the decision-maker agent to continue the search process. We will describe the two crossover operators used by these agents.

4.3.4.1 GPX operator

The GPX crossover (Galinier & Hao, 1999) uses two random parent k-colorings S_1 and S_2 from the archive. In each step, the k classes V_1, V_2,..., V_k of the offspring k-coloring S_0 are created. At the first step, the class V_1 is built by selecting the class having the maximum number of vertices in parent S_1. The second class V_2 of S_0 is built by the same idea but considering the second parent S_2. Other color classes are built considering two parents S_1 and S_2 successively. Once k color classes are built, each left uncolored vertex is allocated to a random color.

4.3.4.2 AMPaX operator

The AMPaX operator (Lu & Hao., 2010) is an extension of GPX operator. It uses randomly more than 2 parents from the archive to produce offspring. For each class color of the new offspring, the color classes of all parents are considered. In each step, the color class with the maximal cardinality in all m parents individuals, is chosen. Then, all vertices colored with this color class are removed from all m parents individuals. The current parent which has been selected, can be reconsidered only after a few number of steps. This mechanism is integrated to avoid focusing in a single parent, so creating other k-coloring solutions.

4.4 Experimentation

In this section, we present experimental results of our MAOM-GCP on well-known DIMACS coloring benchmarks. Then, we compare the results with other state of the art coloring algorithms from the literature.

The DIMACS graphs are the recognized standard benchmarks in the literature for evaluating the performance of graph coloring algorithms (Johnson & al., 1996). The DIMACS graphs are composed from 12 random graphs (DSJC125.x, DSJC250.x, DSJC500.x and DSJC1000.x, x = 1, 5 and 9), 6 flat graphs (flat300 x 0, x=20, 26 and 28; flat1000 x 0, x = 50, 60 and 76), 8 Leighton graphs (le450 15x, le450 25x, x = a, b, c and d), 12 random geometric graphs (R125.x, R250.x, DSJR500.x and R1000.x, x = 1, 1c and 5), 2 huge random graphs (C2000.5 and C4000.5), 2 class scheduling graphs (school1 and school1.nsh) and 1 latin square graph (latin square 10).

These instances can be classified into two categories: easy graphs and difficult graphs. Easy graphs can be solved very easily by most modern coloring heuristics. Difficult graphs can not be solved by all algorithms which can reach chromatic number or the best known results. We only mention our computational results on the set of difficult graphs.

Table 4.1 – Computational results of MAOM-GCP on the difficult DIMACS challenge benchmarks (Part I)

Instances	n	ne	$dens$	k^*	$References$	MAOM-GCP k	MAOM-GCP hit	MAOM-GCP $time(m)$
DSJC250.5	250	15,668	0.50	28	(Galinier & Hao, 1999; Galinier & al., 2008; Malaguti & al, 2008) (Hertz & al., 2008; Porumbel al., 2009, 2010) (Titiloye & Crispin, 2011; Wu & Hao, 2012)	28	10/10	5
DSJC500.1	500	12,458	0.10	12	(Blochliger & Zufferey, 2008; Galinier & al., 2008) (Hertz & al., 2008; Porumbel al., 2009, 2010) (Titiloye & Crispin, 2011; Wu & Hao, 2012) (Hao & Wu, 2012; Moalic & Gondran, 2015)	12	10/10	6
DSJC500.5	500	62,624	0.50	47	(Moalic & Gondran, 2015)	-	-	-
				48	(Galinier & Hao, 1999; Blochliger & Zufferey, 2008) (Galinier & al., 2008; Hertz & al., 2008; Malaguti & al, 2008) (Porumbel al., 2010; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	48	10/10	85
DSJC500.9	500	112,437	0.90	126	(Blochliger & Zufferey, 2008; Galinier & al., 2008; Hertz & al., 2008) (Malaguti & al, 2008; Porumbel al., 2010) (Titiloye & Crispin, 2011; Wu & Hao, 2012) (Porumbel al., 2009; Hao & Wu, 2012; Moalic & Gondran, 2015)	126	10/10	320
DSJC1000.1	1000	49,629	0.10	20	(Galinier & Hao, 1999; Blochliger & Zufferey, 2008; Galinier & al., 2008) (Hertz & al., 2008; Porumbel al., 2010) (Titiloye & Crispin, 2011; Wu & Hao, 2012) (Malaguti & al, 2008; Hao & Wu, 2012; Moalic & Gondran, 2015)	20	10/10	441
DSJC1000.5	1000	249,826	0.5	82	(Moalic & Gondran, 2015)	-	-	-
				83	(Galinier & Hao, 1999; Malaguti & al, 2008) (Porumbel al., 2010) (Wu & Hao, 2012; Hao & Wu, 2012)	83	10/10	205
DSJC1000.9	1000	449,449	0.90	222	(Galinier & Hao, 1999; Blochliger & Zufferey, 2008; Hertz & al., 2008) (Porumbel al., 2010; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012; Moalic & Gondran, 2015)	222	4/10	801
DSJR500.1c	500	121,275	0.97	85	(Hertz & al., 2008; Titiloye & Crispin, 2011) (Wu & Hao, 2012)	85	10/10	60
DSJR500.5	500	58,862	0.47	122	(Hertz & al., 2008; Prestwich, 2002) (Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	122	3/10	480

Table 4.2 – Computational results of MAOM-GCP on the difficult DIMACS challenge benchmarks(Part II)

Instances	n	ne	$dens$	k^*	$References$	MAOM-GCP k	MAOM-GCP hit	MAOM-GCP $time(m)$
R250.5	250	14,849	0.48	65	(Blochliger & Zufferey, 2008; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	65	10/10	42
R1000.1c	1000	485,090	0.97	98	(Blochliger & Zufferey, 2008; Malaguti & al, 2008) (Porumbel al., 2009; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	98	10/10	55
R1000.5	1000	238,267	0.48	234	(Hertz & al., 2008; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	240	2/10	1120
le450_15c	450	16,680	0.17	15	(Galinier & al., 2008; Malaguti & al, 2008) (Hertz & al., 2008; Porumbel al., 2009, 2010) (Titiloye & Crispin, 2011; Wu & Hao, 2012)	15	10/10	40
le450_15d	450	16,750	0.17	15	(Galinier & al., 2008; Hertz & al., 2008; Malaguti & al, 2008) (Porumbel al., 2009, 2010; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	15	10/10	50
le450_25c	450	17,343	0.17	25	(Blochliger & Zufferey, 2008; Malaguti & al, 2008) (Porumbel al., 2009; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	25	10/10	120
le450_ 25d	450	17,425	0.17	25	(Blochliger & Zufferey, 2008; Malaguti & al, 2008) (Porumbel al., 2009; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	25	10/10	42
flat300_26_0	300	21,633	0.48	26	(Blochliger & Zufferey, 2008; Malaguti & al, 2008) (Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	26	10/10	40
flat300_28_0	300	21,695	0.48	28	(Hertz & al., 2008; Wu & Hao, 2012; Hao & Wu, 2012)	30	5/10	500
flat1000_50_0	1000	245,000	0.49	50	(Galinier & al., 2008; Hertz & al., 2008; Malaguti & al, 2008) (Porumbel al., 2009, 2010; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	50	10/10	40
flat1000_60_0	1000	245,830	0.49	60	(Galinier & al., 2008; Hertz & al., 2008; Malaguti & al, 2008) (Porumbel al., 2009, 2010; Titiloye & Crispin, 2011) (Wu & Hao, 2012; Hao & Wu, 2012)	60	10/10	45
flat1000_76_0	1000	246,708	0.49	81	(Hao & Wu, 2012; Moalic & Gondran, 2015)	-	-	-
				82	(Malaguti & al, 2008; Porumbel al., 2009) (Titiloye & Crispin, 2011; Wu & Hao, 2012)	82	10/10	280
C2000.5	2000	999,836	0.50	145	(Hao & Wu, 2012)	-	-	-
				146	(Wu & Hao, 2012)	147	1/5	8000
latin_sqr_10	900	307,350	0.76	97	(Titiloye & Crispin, 2011)	98	2/10	600

Our MAOM-GCP was implemented in Java using the multi-agent platform Jade. The program was run on a computer with a Core I5 2.5 GHz, 8GB of RAM.

Each instance was solved 10 times independently (5 times for very large graphs). We stopped the algorithm when a legal k-coloring is found or the fixed execution timeout is reached. For all instances, a timeout limit of 240 CPU hours was used except for the large graph C2000.5 where a limit of 500 CPU hours (note that large computing times are usually allowed in the literature on GCP). We adjusted the parameters of the proposed algorithms by an experimental study. The number of iterations for each tabu search agent ($iter\_max$) was fixed to 1000. The parameters max_opt (for decision-maker agent) and max_opt_TS (for tabu search agent), that evaluate the improvement of solutions between generations, were fixed to 20 and 2 for respectively. For $interval$, we considered the same value 10 for the same agents. The rate μ used in updating the decision matrices was fixed to 0.9.

Table 4.1 and Table 4.2 summarize the computational results of our MAOM-GCP. Columns 2-4 show the features of the tested instances: the number of vertices (n), the number of edges (ne) and the density of the graph ($dens$). Columns 5 and 6 correspond to the best known results k^* ever reported in the literature and the corresponding references. The remaining columns give the computational results of our MAOM-GCP: the smallest number of colors needed to obtain a legal k-coloring, the success rate ($\#hit$) and the average time for reaching the best legal k-coloring ($time$ in minutes).

Table 4.1 and Table 4.2 show that the results obtained by our MAOM-GCP are competitive with respect to many state of the art algorithms in terms of solution quality (i.e., the number of colors used). It can reach previous best known results except for 7 very difficult cases (DSJC500.5, DSJC1000.5, flat300_28_0, flat1000_76_0, latin_sqr_10, C2000.5 and R1000.5) for which very few algorithms are able to attain the best known results. For these 7 instances, MAOM-GCP the deviation between our results and the best-known results is respectively 0.021 (DSJC500.5), 0.012 (DSJC1000.5), 0.034 (flat300_28_0), 0.012 (flat1000_76_0), 0.002 (R1000.5), 0.013 (C2000.5) and 0.01 (latin_sqr_10) respectively.

4.5 Conclusion

In this section, we presented a multi-agent based optimization algorithm for the Graph Coloring Problem. MAOM-GCP is based on distributed programming realized by multi-agent system which is reinforced by a technique of learning, in order to manage the search to the right decision. In fact, a decision-maker agent decides if the search process needs to be intensified or diversified based on a decision matrix, so two different types of agents are trigged. Tabu search agents, responsible for intensification search, explore two different neighbor structures, and apply a tabu search algorithm to generate progressively a legal k-coloring.

These last agents can get helps from other agents, when the solution can not be further improved. It is the perturbation agent which applies a reduced perturbation move or a strong perturbation move, in order to create another depart solution for tabu search agents. Crossover agents are trigged, to escape deep local optima, by performing two different recombination operators. These agents create a new solution based on a an elite solution archive which is built and updated by the decision-maker agent. All the best k-coloring found by tabu search agents and crossover agents are maintained in this archive, in the decision-maker agent. The proposed algorithm is evaluated on DIMACS coloring benchmarks. The comparative study shows that it is able to reach best known solutions of several instances.

5

A Multi-Agent based Optimization Method for the Winner Determination Problem

In this chapter, we present another application of the proposed method to the Winner Determination Problem (WDP) in combinatorial auctions. In the next section, we will describe the problem. In section 5.2, we will give an overview of algorithms for the WDP. Then, we will apply the proposed method to the WDP. Section 5.4 will contain the experimentations of MAOM-WDP using the WDP benchmarks. In the appendix of this chapter, we will present another algorithm for the WDP. It is a Recombination-Based Tabu Search Algorithm for the WDP (TSX_WDP). TSX_WDP will be evaluated using the same benchmarks. TSX_WDP is presented in (Sghir & al., 2013).

Contents

5.1 Problem definition

The auction consists of an auctioneer wishing to maximize his/her selling revenue and a set of bidders wishing to minimize their cost. Examples of the most widely known auctions are the English auction, the Holland's auction, the Sealed envelope auction and the Vickrey auction (Klemperer, 2004). These auctions typically treat only a single item for each sell. Combinatorial auctions are multi-item auctions, which allow bids on a combination of items (Cramton & al., 2006; Jawad & al., 2007).

In combinatorial auctions, we have a set of items which are exposed to buyers. Buyers offer different bids. Each bid is defined by a subset of items with a price. Two bids are conflicting, if they share at least one item. The Winner Determination Problem (WDP) is to determine a conflict-free allocation of items that maximizes the auctioneer's revenue defined as the sum of the valuations of the winning bids. The WDP is known to be a NP-hard combinatorial optimization problem with a number of practical applications like the e-commerce, the games theory and the resources allocation in multi-agent systems (Vries & Vohra, 2003; Jawad & al., 2007).

Formally, given a set of m items $M = \{1, 2, ..., m\}$ to sell and a set of n bids $N = \{1, 2, ...n\}$. Each bid j is a tuple $< S_j, P_j >$ where S_j is a subset of items covered by bid j, and P_j is the price of bid j. Let B be a $m \times n$ binary matrix such that $B_{ij} = 1$ if object $i \in S_j$, $B_{ij} = 0$ otherwise. Furthermore, we define the decision variable x_j for each bid j such that $x_j = 1$, if bid j is a winning bid, 0 otherwise. Formally, the WDP can be stated as the following binary integer optimization problem.

$$Maximize \quad f(x) = \sum_{j \in N} P_j x_j \tag{5.1}$$

subject to

$$\sum_{j \in N} B_{ij} x_j \leq 1, i \in M \tag{5.2}$$

The objective function (5.1) is to maximize the auctioneer's gain calculated by the sum of prices of the winning bids, while the constraints expressed by formula (5.2) ensure that an item appears at most in one winning bid. We present a simple example to understand better the notations used in the modeling of the WDP. Let us consider a set of five items $1, 2, 3, 4, 5$ to sell by auction and four bids. Each bid is represented by a couple $< S_j, P_j >$ where P_j indicates the price of bid j containing a set of items S_j. The following bids are:

— Bid1: $(1, 2), 250$

— Bid2: $(1, 2, 3), 400$
— Bid3: $(3, 4, 5), 500$
— Bid4: $(4, 5), 200$

Bid1 contains a set of two items $(1, 2)$ which the price is 250. Bid3 has as price 500 for the set of three items $(3, 4, 5)$. Bid1 and Bid3 can constitute a winning allocation maximizing the gain of the seller. The total price of the sale is 750.

5.2 State of the art approaches for the WDP

Several algorithms were proposed to solve the winner determination problem. These algorithms can be divided into two categories: the exact algorithms and the stochastic algorithms. For the exact algorithms, we can quote: Branch-on-Items (BoI), Branch-on-Bids (BoB) (Sandholm & Suri, 2000), CABoB (Sandholm & al., 2001), Combinatorial Auction Structural Search (CASS) (Fujishima & al., 1999), Combinatorial Auctions Multi-unites Search (CAMUS) (Leyton-Brow & al., 2000). In (Rothkopf & al., 1998), an algorithm of dynamic programming for the WDP was introduced. Nisan (Nisan, 2000) proposed a linear programming algorithm for the WDP. Holland and Sullivan (Holland & O'sullivan, 2004) used the constraints programming to solve a particular combinatorial of Vickrey auction.

Some stochastic algorithms were proposed for the WDP.

Casanova (Hoos & Boutilier, 2000) is a local search algorithm proposed by Hoos and Boutilier. Casanova begins with an empty allocation where all bids are considered unsatisfied. In each iteration, an unsatisfied bid is selected to be added in the allocation. Any incompatible bid, which can occur in the current allocation, is removed, when new bids are added. The selection of a bid is based on the following strategies:

— With a probability w_p (Walk probability), an unsatisfied bid is randomly selected.
— With a probability 1-w_p, unsatisfied bids are classified according to their profit. The profit is the price of a bid divided by the number of items covered by this bid. Then, with a probability n_p (Novelty probability), the best unsatisfied bid, which has the biggest profit value, is selected to be added in the current allocation. Otherwise, with a probability 1-n_p, the best second unsatisfied bid is chosen.

In (Guo & al., 2006), Guo et al. proposed the SAGII algorithm which is a simulated annealing combined with the Branch-and-Bound algorithm for the WDP. SAGII begins with a preprocessing to exclude the bids which can lead to the optimal solutions. The search process is composed of three components:

— A branch-and-Bound algorithm applied to the items subsets of the current allocation;
— A simulated annealing algorithm used to select the best unsatisfied bid to be added in the current allocation;
— A random movement performed to select randomly an unsatisfied bid to be considered in the current allocation.

SAGII starts from an empty allocation. A penalty function is used to eliminate the incompatible bids. The Branch-and-Bound algorithm is executed with a probability $p_1 = 0.2$. The simulated annealing algorithm is performed with a probability $p_2 = 0.7$. The random movement is applied with a probability $1 - p_1 - p_2$.

The local search (SLS) proposed by (Boughaci & al., 2009) starts with a possible initial allocation and tries to improve it, by searching for a better solution in the current neighborhood. A Random Key Encoding (RK) is used to generate the initial solution (Bean, 1994). Then, in each iteration, an unsatisfied bid is selected to be integrated into the current allocation. Any contradictory bid, in the current allocation, is removed. Two criteria are fixed for the bid selection. The first criterion consists in choosing an unsatisfied bid in a random way with a fixed probability w_p. The second criterion consists in choosing, with a probability 1-w_p, the best unsatisfied bid that maximizes the gain of the seller.

The tabu search (TS) elaborated by (Boughaci & al., 2009) begins with the RK algorithm (Bean, 1994) to generate the initial solution. The best neighbor is selected for the next solution. To produce neighbor solutions, TS performs two moves which are built in the following way:

— the best unsatisfied bid, which maximizes the total profit of the current allocation, when it will be inserted, is selected. All incompatible bids, in the current allocation, are removed;

— the search space is composed of the items which are not covered by the bids in the current allocation. The best bid covering such items is chosen. All incompatible bids in the current allocation are removed;

After generating all neighbor configurations, the best configuration is selected to be a candidate solution. To escape the visited allocations, a list maintains the bids recently selected.

The memetic algorithm (MA) proposed by (Boughaci & al., 2010) starts by the RK algorithm. Then, it selects C individuals from the current population P to participate in the reproduction phase. C contains the best individuals C_1, which have the highest fitness, and the diverse individuals C_2, which are the individuals the most diverse in the population P. The diversity is measured using a similarity function which calculates the number of the common bids between two individuals. Two parents are selected randomly from C. They are combined to generate a new individual. To locate more effectively solutions, the mutation phase is replaced by a stochastic local search (SLS). The population is updated with the new individual based on the quality and the diversity criteria.

In (Wu & Hao, 2015), Wu and Hao developed an algorithm for the WDP by recasting the WDP into the maximum weight clique problem (MWCP). They solve the transformed problem using a recent heuristic dedicated to the MWCP. A memetic algorithm (MA) was proposed. The proposed algorithm incorporated a novel selection strategy and a specific crossover operator. The stochastic local search (SLS) was used for the intensification search.

We explore the operators and the techniques used in these reference algorithms in our multi-agent model, in order to create the first multi-agent based optimization algorithm solving the WDP. In the next section, we will present the components of the proposed algorithm.

5.3 A Multi-agent based optimization method for the WDP (MAOM-WDP)

In this section, we apply our multi-agent approach to the Winner Determination Problem (MAOM-WDP). We use the following agents: the decision-maker agent, two tabu search agents which are the intensification agents, the perturbation agent and two crossover agents. Below, we describe the behaviors of these agents.

5.3.1 Decision-maker agent

The decision-maker agent generates a simple non conflicting allocation by selecting random items. The decision-maker agent uses a decision matrix (section 2.2.1) which helps it to decide which agents to activate between crossover agents and tabu search agents. It maintains all high-quality solutions, received from other agents, in a shared memory.

5.3.2 Tabu search agents

Two tabu search agents are responsible for the intensification search of MAOM-WDP. During their search, these agents can exchange with another alive tabu search agent or with a perturbation agent based on their decision matrices (section 2.2.1) and according to the corresponding condition (section 2.5.1). They send the best allocation found to the decision-maker agent. Below, we define the used neighborhood strategies for each tabu search agent.

5.3.2.1 Neighborhood exploration strategies

A candidate solution is represented by an allocation A (a dynamic vector). Each element of this allocation A receives the winning bid. Each bid is an object composed of the list of items and the associated prices. The first tabu search agent explores the neighborhood strategy proposed by (Boughaci & al., 2009) (section 5.2).

The second tabu search agent performs the following neighborhood strategy:

— The initial candidate (unsatisfied) bids are sorted according to their utility prices;

— For each candidate bid B_x, a binary gain function is used to verify if the bid can increase the revenue of the current allocation when the bid is inserted;

— Let Q be the set of winning bids that are in conflict with the current candidate bid B_x, Let $f(Q)$ be the revenue of the set of winning bids Q, and $f(B_x)$ the price of the candidate bid B_x. The gain function returns true if $f(Q) < f(B_x)$ and returns false otherwise;

— According to this expression, a candidate bid B_x can enter in the current allocation only if its price $f(B_x)$ is higher than the revenue of other winning bids which are conflicting with B_x in the current allocation (i.e., the gain function is true);
— The gain of B_x, when it is selected to be added in the current allocation, is calculated according to the following function:

$$Gain(B_x) = f(A) - f(Q) + f(B_x) \tag{5.3}$$

— When a bid B_x is inserted in the current allocation A, the bids of Q which are conflicting with B_x are removed from A;
— The steps mentioned previously are iterated until all the initial candidate bids are visited and possibly added in the current allocation A.

5.3.2.2 The tabu list and the tabu tenure management

The tabu search agents use a tabu list to forbid recently visited solutions from being revisited. A bid that is chosen to be inserted in the current allocation A is forbidden to be removed for the next tt iterations. This number of iterations, named the tabu tenure, is calculated dynamically by the function: $tt = L + \lambda + f(A)$ where L is randomly chosen from the interval [0, 9] and λ is empirically fixed to 0.6. Notice that we allow a move to be accepted in spite of being tabu if the move leads to a solution better than any found so far. This is called the aspiration criterion.

5.3.3 Perturbation agent

The perturbation agent is activated by a tabu search agent when it needs diversification search under specific conditions (C_2 and C_3 of section 2.5.1). This agent creates a new perturbed solution that manages the search towards other regions. It performs two parallel behaviors which are reduced perturbation behavior and strong perturbation behavior. The resulting solution is sent to the tabu search agent.

5.3.3.1 Reduced perturbation technique

The reduced perturbation technique is activated when the tabu search agent observes a slight search stagnation (condition C_2 of section 2.5.1). The perturbation agent chooses randomly one candidate unsatisfied bid from the available ones. Then, the selected bid is inserted in the allocation received from the tabu search agent. All the contradictory bids are removed from this allocation.

5.3.3.2 Strong perturbation technique

The strong perturbation technique is applied, when the tabu search agent observes a strong search stagnation. Based on the archive of elite solutions, the perturbation agent extracts the number of occurrences of each bid appeared in the high-quality allocations. Then, the bids, which have the smallest occurrence number, are inserted in the current non conflicting allocation. In order to create a new solution in each call of the perturbation agent, data structures are employed to save the visited solutions.

5.3.4 Crossover agents

Two crossover agents are activated, when the decision-maker agent observes a local optimum reached based on its decision matrix. These two agents apply crossover operations to produce new offspring allocations. The first crossover agent explores the crossover operator which was proposed by (Boughaci & al., 2010) (section 5.2). The second crossover agent employs the recombination operator which is described below.

This operator aims to transform the good properties of the parents towards the offspring. These criteria have to assure that the offspring inherits the properties of the parents. The pseudo-code of the recombination operator is given in Algorithm 5.1. Given two parent allocations I_1 and I_2 from the common archive, these parents share the highest number of bids. The second crossover agent constructs the offspring I_0 in k steps until all the bids of the two parents are visited. This operator is inspired by the idea of backbone used in (Benlic & Hao, 2011; Wang & al., 2013). In the first step, the set of bids, that are shared by the parents, are identified and directly transmitted to I_0. Then the following steps are performed:

— Choose the bid with the lowest price from each parent (lines 4 and 5 from Algorithm 5.1);
— The two selected bids are candidates bids that can be inserted in the offspring, if they are not conflicting bids. This is by conserving, the best bids, which have the highest revenue (lines 6 and 7 from Algorithm 5.1);
— Remove the selected bids from their parents, even if they are not inserted in the offspring (lines 9 and 10 from Algorithm 5.1);
— Repeat the previous steps until all the bids of the parents are examined and removed.

An example of this recombination operator is provided in Figure. 5.1.

The two allocations, generated by the crossover agents, are sent to the decision-maker agent. They will be the new current allocation for the search process.

5.4 Experimentations of MAOM-WDP

We present in this section experimental results of MAOM-WDP on the set of well-known WDP benchmarks. MAOM-WDP was implemented in Java using the platform Jade. The program was run on a computer with a Core I5 2.5GHz, 8GB of RAM. Tests were made on various benchmarks of diverse sizes defined in (Lau & Goh, 2002). These benchmarks take into account several factors like the prices, bidders preferences and object distribution on bids. They can be divided into five groups where each group contains 100 instances:

— REL 500-1000: From in101 to in200: m = 500, n = 1000
— REL 1000-1000: From in201 to in300: m = 1000, n = 1000
— REL 1000-500: From in401 to in 500: m = 1000, n = 500

Algorithm 5.1. The recombination operator of the second crossover agent

---

Require: two parent solutions I_1 and I_2
Ensure: An offspring solution I_0

$I_0 \leftarrow \emptyset$, $D_1 \leftarrow \emptyset$, $D_2 \leftarrow \emptyset$
Sort the bids in each parent according to their prices
while I_1 and I_2 are not empty **do**
    $D_1 \leftarrow first\_element(I_1)$
    $D_2 \leftarrow first\_element(I_2)$
    if D_1 and/or D_2 are no conflict bids with the bids in I_0 **then**
        add D_1 and/or D_2 to I_0
    end if
    remove D_1 from I_1
    remove D_2 from I_2
end while
Return Child I_0

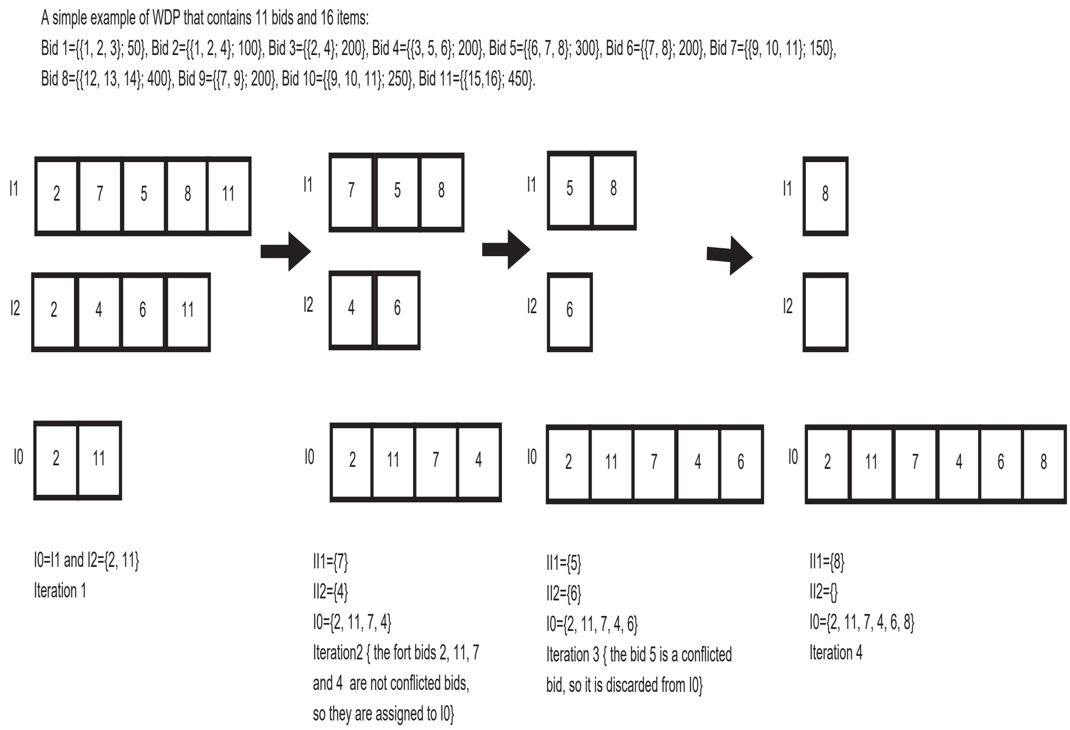

Figure 5.1 – An example of the recombination operator of MAOM-WDP algorithm

— REL 1000-1500: From in501 to in 600: m = 1000, n = 1500
— REL 1500-1500: From in601 to in 700: m = 1500, n = 1500

We adjusted the parameters of MAOM-WDP by an experimental study. The number of iterations for each tabu search agent ($iter\_max$) was fixed to 500. The parameters max_opt (for decision-maker agent) and max_opt_TS (for tabu search agents), which evaluate the improvement of solutions between generations, were fixed to 20 and 25 respectively. As $interval$, we considered the same value 1000 for the same agents. The rate μ used in updating the decision matrices was fixed to 0.9.

5.4.1 Experimental results

In Tables 5.1, 5.2, 5.3, 5.4, and 5.5, we provide the computational results of MAOM-WDP on the set of the five groups of benchmarks. Given that there are 500 instances, we show only some results of each group, like in some recent papers (Boughaci & al., 2010). Columns give the following computational statistics of each tested instance: the $maximumrevenue$ obtained by the MAOM-WDP algorithm over the 10 independent trials $Rbest$, the $averagerevenue$ over the 10 trials $Ravg$, the $worstrevenue$ over the 10 trials $Rworst$ and the average CPU time in seconds $AvgTime$. These tables show that the values of $Ravg$ are equal to the values of $Rbest$ in all instances.

Table 5.1 – Some results obtained by MAOM-WDP on REL 500-1000 for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in101	69585.298	69585.298	69585.298	96
in102	72518.222	72518.222	72518.222	65
in103	70999.247	70999.247	70999.247	81
in104	71327.641	71327.641	71327.641	75
in105	73351.044	73351.044	73351.044	102
in106	66440.95	66440.95	66440.95	81
in107	68796.927	68796.927	68796.927	74
in108	74867.585	74867.585	74867.585	76
in109	64662.355	64662.355	64662.355	79
in110	66549.957	66549.957	66549.957	71

Table 5.2 – Some results obtained by MAOM-WDP on REL 1000-1000 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in201	81557.742	81557.742	81557.742	66
in202	90537.285	90537.285	90537.285	61
in203	86239.213	86239.213	86239.213	64
in204	84879.397	84879.397	84879.397	59
in205	83758.599	83758.599	83758.599	62
in206	87544.451	87544.451	87544.451	64
in207	93115.569	93115.569	93115.569	68
in208	91774.549	91774.549	91774.549	57
in209	86441.696	86441.696	86441.696	59
in210	89962.396	89962.396	89962.396	56

Table 5.3 – Some results obtained by MAOM-WDP on REL 1000-500 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in401	77417.482	77417.482	77417.482	10
in402	76273.336	76273.336	76273.336	11
in403	74843.958	74843.958	74843.958	10
in404	78761.690	78761.690	78761.690	12
in405	75915.900	75915.900	75915.900	10
in406	72863.324	72863.324	72863.324	10
in407	76365.717	76365.717	76365.717	11
in408	77018.833	77018.833	77018.833	10
in409	73188.62	73188.62	73188.62	15
in410	73791.65	73791.65	73791.65	18

Table 5.4 – Some results obtained by MAOM-WDP on REL 1000-1500 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in501	88656.95	88656.95	88656.95	112
in502	86236.911	86236.911	86236.911	95
in503	83718.749	83718.749	83718.749	93
in504	85600.002	85600.002	85600.002	84
in505	83071.930	83071.930	83071.930	73
in506	83059.438	83059.438	83059.438	74
in507	90288.472	90288.472	90288.472	81
in508	84033.386	84033.386	84033.386	88
in509	86045.479	86045.479	86045.479	87
in510	88163.815	88163.815	88163.815	85

Table 5.5 – Some results obtained by MAOM-WDP on REL 1500-1500 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in601	107823.098	107823.098	107823.098	120
in602	99718.150	99718.150	99718.150	85
in603	98577.454	98577.454	98577.454	82
in604	102332.650	102332.650	102332.650	85
in605	111645.103	111645.103	111645.103	92
in606	101496.527	101496.527	101496.527	83
in607	104616.624	104616.624	104616.624	95
in608	102231.73	102231.73	102231.73	84
in609	100697.634	100697.634	100697.634	86
in690	106754.424	106754.424	106754.424	63

5.4.1.1 Comparative results for MAOM-WDP

In this section, we show the comparative study of MAOM-WDP with other algorithms from the literature: Casanova (Hoos & Boutilier, 2000), SAGII (Guo & al., 2006), SLS (Boughaci & al., 2009), TS (Boughaci & al., 2009), MA (Boughaci & al., 2010), MN/TS (Wu & Hao, 2015).

In Table 5.6, we present the generic comparative results for each group. In this table, rows μ correspond to the average of best objective value of the 100 instances in each group. Rows $time$ represent the average time to reach the best solution. $\delta(\%)$ is the deviation of the MAOM-WDP algorithm with respect to each reference algorithm. The deviations are calculated respectively as follows: $\mu_{MAOM-WDP} - \mu_{algo\_X})/\mu_{MAOM-WDP}$ where *algo_X* is one of the five reference algorithms. Since the compared algorithms are implemented in different languages and run in different computer, the comparison is focused on solution quality that can be reached by each algorithm. The computing time is provided only for indicative purposes. The results of the reference algorithms are extracted from the corresponding papers except the results of Casanova are given by (Guo & al., 2006).

Table 5.6 shows that MAOM-WDP gives an improvement between 32% and 48% in solution quality compared to Casanova in shorter time. MAOM-WDP outperforms TS (the improvement rate is between 4% and 11%), SLS (the improvement rate is between 4% and 10%), MA (the improvement rate is between 2% and 9%). The results of MAOM-WDP are close to the results of MN/TS. The deviation is between -2% and 0 %.

5.5 Conclusion

In this chapter, we proposed a multi-agent based optimization algorithm for the winner determination problem. The proposed algorithm combines different techniques of diversification and techniques of intensification. The tabu search agents

Table 5.6 – Comparative results between MAOM-WDP Casanova, MA, SLS, TS, SAGII, MN/TS on WDP benchmarks: rows μ correspond to the average of the best objective value of the 100 instances in each group. Columns $time$ represent the average time to reach the best solution.

Test Set	100 instances	REL-500-1000	REL-1000-500	REL-1000-1000	REL-1000-1500	REL-1500-1500
MAOM-WDP	Time	70	10	53	85	101
	μ	70215.711	75540.68	87.292.848	87041.037	106093.955
Casanova	Time	119.46	57.74	111.42	168.24	165.92
	μ	37053.78	51248.79	51990.91	56406.74	65661.03
$\delta_{MAOM-WDP/Casanova}(\%)$		47.22	32.15	40.44	35.19	38.11
TS	Time	91,07	25.84	104,30	223,37	175.68
	μ	65286.94	71985.34	81633.63	77931.41	97824.64
$\delta_{MAOM-WDP/TS}(\%)$		7.01	4.7	6.48	10.46	7.79
SLS	Time	22.35	5.91	14.19	14.97	16.47
	μ	64216.14	72206.07	82120.31	79065.08	98877.07
$\delta_{MAOM-WDP/SLS}(\%)$		8.54	4.61	5.92	9.16	6.8
MA	Time	56.64	14.98	33.05	24.51	28.22
	μ	65740.25	73604.62	83304.20	79644.64	99957.96
$\delta_{MAOM-WDP/MA}(\%)$		6.37	2.56	4.56	8.49	5.78
SAGII	Time	38.06	24.46	45.37	68.82	91.78
	μ	64922.02	73922.10	83728.34	82651.49	101739.64
$\delta_{MAOM-WDP/SAGII}(\%)$		7.53	2.14	4.08	5.04	4.1
MN/TS	Time	12.28	0.38	3.12	6.39	2.64
	μ	71470.93	75540.68	89158.98	89552.18	108627.17
$\delta_{MAOM-WDP/MN/TS}(\%)$		-1.75	0	-2.09	-2.8	-2.33

are responsible for the intensification search. One of these tabu search agent explores new neighborhood strategy for the WDP. The new strategy of the selection of the best neighbor helps the algorithm to maintain, a diversification of the population what leads to a good compromise between the intensification and the diversification. The use and the update of the tabu tenure, in each iteration of the algorithm, improve the diversification in order to discover other areas in the search space.

The crossover agents employ crossover operators as another tool for the diversification of the search space. One of these agent explores a new technique of crossover for the WDP, that gives the priority to the invariants bids to stay in the new descendant. Then, this agent adds the conflict-free bids from the two parents selected according to their prices. The new crossover strategy aims to take the good information of the parents, then try to find other different efficient solutions. The new descendant solution can change the direction of the search because it is the new starting point for other iterations of the tabu search. The proposed MAOM-WDP is evaluated on a set of 500 benchmark instances. The comparative study with reference algorithms shows that it is able to reach solution of very high quality. Another centralized algorithm, named TSX_WDP, is presented in the appendix of this chapter.

5.A Appendix: A Recombination-based Tabu Search algorithm for the WDP (TSX_WDP)

We propose a dedicated tabu search algorithm (TSX_WDP) for the Winner Determination Problem (WDP) in combinatorial auctions. TSX_WDP integrates two complementary neighborhoods designed respectively for the purpose of intensification and diversification. To escape deep local optima, TSX_WDP employs a backbone-based recombination operator to generate new starting points for tabu search and to displace the search into unexplored promising regions. The recombination operator operates on elite solutions previously found which are recorded in a global archive. In this section, we present these key components. Then, the performance of the proposed algorithm is assessed on a set of 500 well-known WDP benchmark instances. Comparisons with state of the art algorithms demonstrate the effectiveness of TSX_WDP.

5.A.1 TSX_WDP algorithm

The generic TSX_WDP algorithm is formalized in Algorithm 5.2. The algorithm starts with an empty allocation in which no bd is chosen and tries to improve it, by looking for a better solution in the current neighborhood. In each iteration, the best authorized bids are selected among the candidate bids to be included in the current allocation. This is achieved with an intensification move (lines 7-9 of Algorithm 5.2). This intensification move is the neighborhood strategy performed by the second tabu search agent of MAOM-WDP and developed in section 5.3.2.1. When no bit can be found to increase the revenue with the intensification move, TSX_WDP switches to the perturbation move by choosing a random bid from the candidate bids (line 11 of Algorithm 5.2). In both cases, the choice of the bids depends on a status of the tabu list which is updated after each move. Any conflicting bid, being able to occur in the current allocation, when new bids are considered, is removed (lines 13 and 14 of Algorithm 5.2).

The search process is repeated for a fixed number $Itermax$ of iterations. During these $Itermax$ iterations, if the current best solution can not be updated for consecutive p (fixed experimentally) moves, the best local optimum found so far is inserted into the archive P and a recombination operator (Algorithm 5.1 & section 5.3.4) is activated to generate a new starting point for a new round of the tabu search procedure (lines 20-25 of Algorithm 5.2). The tabu search steps starts again with the new offspring. The best solution is the best revenue found during these iterations.

5.A.2 Experimentations of TSX_WDP

This section gives experimental results of TSX_WDP which was implemented in Java. The program was run on a computer with a Core I5 2.5GHz, 8GB of RAM. We adjusted the parameters of the proposed algorithms by an experimental study: The maximum number of iterations ($itermax$) was fixed to 200 and the parameter

Algorithm 5.2. TSX_WDP for the Winner Determination Problem

Require: A matrix M, a parameter $Itermax$, Vector of bids B, Parameter p
Ensure: a vector of winning bids A^* and its revenue $f(A^*)$

$Iter \leftarrow 0$ {Iteration counter}, Initiate $tabu\_list$
$A^* \leftarrow A \leftarrow \varnothing$
$opt \leftarrow 0$ {An integer that will be incremented if the current solution doesn't improve in two consecutive iterations opt returns to 0, when it exceeds the value p, after activating the recombination operator}
initialize $tabu\_list$
$P \leftarrow \varnothing$ {An archive of the best local optima encountered A^*}
while $(Iter < Itermax)$ **do**
  Construct neighborhoods from A based on the intensification move
  if There exists intensification move **then**
    Choose an overall best allowed neighbor A' according to max gain criterion and by considering M {to remove from A' any conflicting bid) {section 5.3.2.1}
  else
    Apply the perturbation move by choosing a random bid from B to create a neighbor A'
  end if
  $A \leftarrow A'$ (Move to the selected neighboring solution A')
  Update $tabu\_list$ {section 5.3.2.2} and B {delete the winner bids from B and add the looser bids in it}
  if $f(A) > f(A^*)$ **then**
    $A^* \leftarrow A$
  else
    $opt \leftarrow opt + 1$
  end if
  if $opt = p$ **then**
    Add A^* to the Archive P
    $I_1, I_2 \leftarrow$ Parent_Selection(P) { section 5.3.4 }
    $I_0 \leftarrow$ Recombination_Operator(I_1, I_2) { section 5.3.4 }
    $A \leftarrow I_0$
    $opt \leftarrow 0$
  end if
  $Iter \leftarrow Iter + 1$
end while
Return (A^* and $f(A^*)$)

responsible for the tabu tenure λ was fixed to 0.00006. Each instance was solved 40 times independently by the TSX_WDP algorithm with different random seeds.

In Tables 5.8, 5.10 and 5.11, the computational results of the TSX_WDP are presented on the set of the five groups of benchmarks. Given that there are 500 instances, we show only some results of each group, like in some recent papers (Boughaci & al., 2010). According to this table, the values of $Ravg$ are very close to the values of $Rbest$ in most of cases and these two values are even equal for certain instances (for example for in101, in102, in205...). These tables show that the proposed algorithm can consistently reach high quality solutions for the tested problems.

Table 5.7 – Some results obtained by TSX_WDP on REL 500-1000 instances for benchmarks

Instances	Rbest	Ravg	Rworst	AvgTime
in101	69585.298	69585.298	69585.298	88
in102	72518.222	72518.222	72518.222	76
in103	69730.618	69475.485	65903.632	75
in104	71327.641	70765.941	65948.396	78
in105	73351.044	71570.624	68899.994	93
in106	66361.943	66361.943	66361.943	73
in107	68796.927	68087.087	63208.126	71
in108	74867.585	74867.585	74867.585	76
in109	64662.355	63063.546	60265.685	70
in110	65446.198	65446.198	65446.198	72

Table 5.8 – Some results obtained by TSX_WDP on REL 1000-1000 instances for benchmarks

Instances	Rbest	Ravg	Rworst	AvgTime
in201	81557.742	80383.277	79331.63	56
in202	89289.573	86815.261	81291.193	52
in203	86239.213	83941.410	77220.427	54
in204	84879.397	84374.869	76822.810	55
in205	83748.837	83748.837	83748.837	57
in206	87544.451	84866.292	78889.312	56
in207	93115.569	90605.049	85924.110	61
in208	91774.549	90543.192	79460.979	56
in209	86441.696	85261.813	80749.28	54
in210	89962.396	88281.194	79813.790	54

Table 5.9 – Some results obtained by TSX_WDP on REL 1000-500 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in401	77417.482	77191.182	70628.481	12
in402	76273.336	76153.051	74469.073	10
in403	74843.958	74356.247	69989.28	10
in404	78761.690	78597.224	77939.364	10
in405	75915.900	75640.510	74899.125	10
in406	72863.324	72671.474	71424.453	10
in407	76365.717	76066.503	72325.694	10
in408	77018.833	76606.838	71892.212	10
in409	70035.529	69789.998	68800.204	9
in410	73628.485	73212.462	71107.518	10

Table 5.10 – Some results obtained by TSX_WDP on REL 1000-1500 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in501	83738.040	83506.552	82605.443	107
in502	83297.340	82546.590	76751.565	82
in503	83718.749	82017.955	78112.719	81
in504	83944.901	82772.535	77217.558	76
in505	83071.930	81876.413	78909.275	66
in506	83059.438	82252.613	78694.650	64
in507	90288.472	85525.706	83484.521	79
in508	84033.386	83588.301	80031.616	77
in509	86045.479	85169.719	79655.527	75
in510	88163.815	87802.967	77338.362	74

Table 5.11 – Results obtained by TSX_WDP on REL 1500–1500 instances for benchmarks

Instance	Rbest	Ravg	Rworst	AvgTime
in601	107246.248	102862.848	96840.461	117
in602	99668.269	97854.579	91452.904	78
in603	98577.454	96567.287	95219.36	75
in604	101713.602	100786.326	99395.413	78
in605	107919.106	103579.211	92948.474	80
in606	101496.527	100090.342	91790.496	79
in607	100336.777	98225.923	95251.78	82
in608	102231.73	100540.091	95641.925	78
in609	100697.634	100060.045	90727.512	76
in690	106754.424	103128.505	95636.147	57

In order to further show the effectiveness of the algorithm, we present a comparative study with six state of the art algorithms from the literature: Casanova (Hoos & Boutilier, 2000), SAGII (Guo & al., 2006), SLS (Boughaci & al., 2009), TS (Boughaci & al., 2009), MA (Boughaci & al., 2010) and MN/TS (Wu & Hao, 2015).

In Table 5.12, we show the generic comparative results for each group. Table 5.12 discloses that TSX_WDP gives an improvement between 31% and 47% in solution quality compared to Casanova. The TSX_WDP algorithm finds better solutions in shorter time, than Casanova. In addition, it shows good performances of the TSX_WDP algorithm in solving the WDP compared to SLS. The improvement rate is between 4% and 8%. The results of TSX_WDP is better than TS in quality and in time (the improvement rate is between 4% and 9%). TSX_WDP outperforms MA. The deviation is between 2% and 7%. Finally, TSX_WDP produces better results than SAGII which is based on sophisticated Branch-and-Bound and preprocessing tools (The deviation is between 1% and 7%). Compared to MN/TS algorithm, which is the most successful algorithm, the deviation is between -5% and -0.5 %. Thus, we can conclude that the TSX_WDP algorithm discovers good results for the five groups of benchmarks.

Table 5.12 – Comparative results between TSX_WDP and Casanova, MA, SLS, TS, SAGII, MN/TS on WDP benchmarks: rows μ correspond to the average of the best objective value of the 100 instances in each group. Columns $time$ represent the average time to reach the best solution.

Test Set	100 instances	REL-500-1000	REL-1000-500	REL-1000-1000	REL-1000-1500	REL-1500-1500
TSX_WDP	Time	74.19	9.45	48.98	75.92	90.61
	μ	69647.975	75274.184	86786.159	85577.806	103178.732
Casanova	Time	119.46	57.74	111.42	168.24	165.92
	μ	37053.78	51248.79	51990.91	56406.74	65661.03
$\delta_{TSX/Casanova}(\%)$		46.79	31.91	40.09	34.08	36.36
TS	Time	91,07	25.84	104,30	223,37	175.68
	μ	65286.94	71985.34	81633.63	77931.41	97824.64
$\delta_{TSX/TS}(\%)$		6.26	4.36	5.93	8.93	5.18
SLS	Time	22.35	5.91	14.19	14.97	16.47
	μ	64216.14	72206.07	82120.31	79065.08	98877.07
$\delta_{TSX/SLS}(\%)$		7.79	4.07	5.37	7.61	4.16
MA	Time	56.64	14.98	33.05	24.51	28.22
	μ	65740.25	73604.62	83304.20	79644.64	99957.96
$\delta_{TSX/MA}(\%)$		5.61	2.21	4.01	6.93	3.12
SAGII	Time	38.06	24.46	45.37	68.82	91.78
	μ	64922.02	73922.10	83728.34	82651.49	101739.64
$\delta_{TSX/SAGII}(\%)$		6.78	1.79	3.52	3.41	1.39
MN/TS	Time	12.28	0.38	3.12	6.39	2.64
	μ	71470.93	75540.68	89158.98	89552.18	108627.17
$\delta_{TSX/MN/TS}(\%)$		-2.54	-0.58	-2.66	-4.43	-5.01

6

A Multi-Agent based Optimization Method for the Multidimensional Knapsack Problem

This chapter shows another application of the proposed method to the Multidimensional knapsack problem (MKP). In the first section, we will present this problem. Then, we will give a brief overview of algorithms for the MKP. Section 6.3 will describe the behaviors of the agents of MAOM-MKP. In section 6.4, MAOM-MKP will be tested using the OR-library benchmarks, then, it will be compared to the current state of the art approaches.

Contents

6.1 Problem definition

The Multidimensional Knapsack Problem (MKP) consists in selecting a subset of objects (or items), in order to maximize their total profit. The selected objects must not violate a set of knapsack constraints. The Multidimensional Knapsack Problem (MKP) can be formulated as:

$$Maximize \quad f(x) = \sum_{j=1}^{n} p_j x_j \tag{6.1}$$

subject to

$$\sum_{j=1}^{n} r_{ij} x_j \leq b_i \tag{6.2}$$

$$x_j \in \{0, 1\} \tag{6.3}$$

where $i = \{1, ..., m\}$ and $j = \{1, ..., n\}$

According to these expressions, the decision variables $x_j = 1$, if the object j is selected, 0 otherwise. p_j is the profit associated to j. Each of the m constraints is called a knapsack constraint. A set of n objects with profit $p_j > 0$ and a set of m resources are given. Each object j consumes an amount $r_{ij} \geq 0$ from each resource i. When $m = 1$, the MKP degenerates to the knapsack problem. It can be solved in pseudo-polynomial time. However, when $m > 1$, it becomes a strongly NP-hard problem and exact techniques can only be used to solve small instances sizes.

The MKP can formulate many real-world application like capital budgeting problem (Markowitz & Manne, 1957), allocating processors and databases in a distributed computer system problem (Gavish & Pirkul, 1979), cutting stock problem (Gilmore & Gomory, 1966) and project selection and cargo loading problem (Shih, 1979). The MKP can be considered as a generic 0-1 integer programming problem with non-negative coefficients.

6.2 State of the art approaches for the MKP

We give a brief overview of some of the most representative algorithms for the MKP. The best algorithms will be used to evaluate the proposed algorithm. Exact algorithms and metaheuristic have been developed for the MKP. On the one hand, the branch and bound algorithms (Shih, 1979) were proposed as exact algorithms. For instance, Gavish and Pirkul (Gavish & Pirkul, 1985) proposed a branch and bound algorithm with tighter upper bounds, combined with relaxation techniques.

On the other hand, several metaheuristics have appeared in the literature. A simulated annealing, based on the add-interchange-drop technique for handling the constraints, was presented by Drexl (Drexl, 1988).

Hanafi and Fréville (Hanafi & Fréville, 1998) elaborated a tabu search algorithm using the surrogate constraints information. A genetic algorithm is proposed by Chu and Beasley (Chu & Beasley, 1998). Vasquez and Hao (Vasquez & Hao, 2001) presented a hybrid approach combining the linear programming and the tabu search. This algorithm integrates the drop-add repair operator based on the pseudo-utility ratios to generate feasible solutions. Sakawa and Kato (Sakawa & Kato, 2003) proposed a genetic algorithm with double strings based on a decoding algorithm. Vasquez and Vimont (Vasquez & Vimont, 2005) proposed a hybrid method combining the linear programming with an efficient tabu search. Puchinger et al. (Puchinger & al., 2005) presented a cooperative combination of a memetic algorithm and a branch-and-cut algorithm. These algorithms run in parallel and asynchronous way by exchanging information during the optimization process.

Li et al. (Li & al., 2006) elaborated a genetic algorithm based on the orthogonal design. A scatter search method was applied in (Hanafi & Wilbaut, 2008) for the MKP. Kong et al. (Kong & al., 2008) presented a binary ant system (BAS) algorithm based on the drop-add repair operator. In (Boyer & al., 2009), two heuristics were provided. The first one uses surrogate relaxation, and the relaxed problem is solved via a modified dynamic programming algorithm. The second one combines a limited-branch-and-cut procedure with the previous heuristic, and tries to improve the bound obtained by exploring some nodes that have been rejected by the modified dynamic programming algorithm.

Zou et al. (Zou & al., 2011) employed a novel global harmony search algorithm to solve MKP. Bonyadi and Li (Bonyadi & Li, 2012) proposed a discrete electromagnetism-like mechanism (DEM) which integrates the drop-add repair operator. Langeveld and Engelbrechta (Langeveld & Engelbrecht, 2012) elaborated a set-based particle swarm optimization (SBPSO) algorithm based on the penalty function method to handle the constraints. An ant colony optimization algorithm for binary knapsack problem under fuzziness was proposed by Changdar et al. (Changdar & al., 2013). A simplified binary version of the artificial fish swarm algorithm applied to the MKP was provided in (Abul Kalam Azad & Ana Maria , 2015). We can find an interesting review of approaches for the MKP in (Fréville, 2004).

Based on this review, we elaborate the first multi-agent algorithm for the MKP (MAOM-MKP). MAOM-MKP, which is presented in the next section, combines several techniques and operators from existing algorithms for the MKP in an intelligent way using multi-agent system and reinforcement learning.

6.3 A multi-agent based optimization method for the MKP (MAOM-MKP)

This section describes the multi-agent based optimization algorithm for the MKP (MAOM-MKP). The agents are the decision-maker agent, two tabu search agents, the perturbation agent and two crossover agents. We explain the specific characteristics of the proposed method to be applied to the MKP.

6.3.1 Decision-maker agent

In MAOM-MKP, the selection of agents to activate, is handled by the decision-maker agent using its decision matrix (section 2.2.1) under a specific condition (section 2.4.1). The decision-maker agent starts the optimization process by generating an initial solution. This initial solution is a simple feasible configuration which contains random non conflicting objects. This solution is sent to the appropriate agents. Then, when other agents are activated, decision-maker agent waits high-quality solutions received from tabu search agents or crossover agents. Decision-maker agent maintains these solutions in the shared memory. (Algorithm 2.2)

6.3.2 Tabu search agents

Tabu search agents apply tabu search algorithm with two different neighborhood strategies (Algorithm 2.3). During their optimization process, they can exchange information with another alive tabu search agent or with the perturbation agent. The triggered agent depends on their decision matrices (section 2.2.1) and the corresponding condition (section 2.5.1). The best solutions generated are sent to the decision-maker agent. Below, we describe the two neighborhood exploration strategies explored by these agents.

6.3.2.1 Neighborhood

A candidate MKP solution can be represented by binary decision variables x_j where $x_j = 1$, if the object j is selected, 0 otherwise. In our algorithm, a solution is a configuration (a dynamic vector) V that contains the selected objects (variables $x_j = 1$). The objective value of a solution is the sum of the prices of the objects selected. Let $move(x, x')$ be a move operator which changes a small set of components of x giving x'. The neighborhood of x is the subset of configurations reachable from x in one move. According to the predefined representation, this move operator is to remove objects from the current configuration and to add other non selected objects to it, at the same time. The neighborhood which satisfies the constraints is the classical add/drop neighborhood.

6.3.2.2 Neighborhood exploration strategies

In MAOM-MKP, we have two tabu search agents. Each agent applies a strategy to explore the neighboring solutions. The first tabu search agent explores the whole neighborhood by removing an object j from the current configuration and adds the best non selected object j' to it. j' is chosen from objects that do not belong to the current configuration. The best retained neighboring solution is a feasible configuration that does not violate the capacity constraints.

The second tabu search agent examines a reduced neighborhood by picking a random object j from the current configuration. Then, this object is replaced by the best non selected object j' that improves the total profit of the current configuration. This exploration strategy reduces the aggressive exploitation of the first tabu search agent and gives some aspect of diversification.

In order to reduce the complexity search of a neighboring solution, these two strategies employ a matrix $\delta(x, x')$ (Expression 6.4) that stores the move gain, if an object j is replaced by an object j'.

$$\delta(x, x') = (F + f(x')) - f(x) \tag{6.4}$$

where F is the total profit of the current configuration, $f(x)$ is the profit of an object j and $f(x')$ is the profit of an object j'.

6.3.2.3 Tabu list

The tabu search agents use a traditional tabu list to store the selected objects of all completed moves. When an object j' is added to the new configuration, it is forbidden to be selected during the next h iterations. The iterations h is dynamically determined by $h = \alpha \times F + rand(10)$ where $rand(10)$ takes a random number in $[1, ..., 10]$ and α is set to 0.1 fixed by experimentation tests.

6.3.3 Perturbation agent

During their optimization process, tabu search agents can reach specific cases that require diversification tasks (C_2 and C_3 of section 2.5.1). In these cases, it can trigger the perturbation agent to perturb the current configuration. The perturbation agent applies two techniques of perturbation which are reduced perturbation technique and strong perturbation technique. After generating the perturbed solution, the perturbation agent sends it to the requester tabu search agent which uses it as its new current configuration.

6.3.3.1 Reduced perturbation technique

The perturbation agent triggers the reduced perturbation behavior, when the tabu search agent detects a slight search stagnation (condition C_2 of section 2.5.1). This perturbation consists in performing add/remove random moves to generate a new feasible configuration. L random objects are removed from the received configuration. The removed objects are replaced by S random non selected objects, generating a new configuration that satisfies the constraints.

6.3.3.2 Strong perturbation technique

In the second case, the perturbation agent can activate the strong perturbation behavior, when the tabu search agent encounters a strong search stagnation. Based on the common archive of elite solutions, the perturbation agent creates a new solution to manage the search towards new regions. It extracts the number of occurrences of each object j which has been selected in a high quality solution belonging to the archive. The new configuration contains the non conflicting objects which have the smallest occurrence number. Like other problems solved in this thesis, we use an additional data structure (matrix that maintains the visited solutions) that avoids the creation of the same solution by the perturbation agent.

6.3.4 Crossover agents

The decision-maker can activate two crossover agents to escape deep local optima. The crossover agents apply two different crossover operators to generate two new solutions which are sent to the decision-maker agent. For these crossover agents, the parents selection is based on diversity criterion. The parents are selected from the common archive as follow:

The diversity of the solutions is measured by a similarity function which calculates the number of common objects between two configurations, in the archive. Based on the similarity values, a random number of diverse solutions are selected. From these solutions, two parents are randomly picked up. Below, we present the two crossover operators explored by the crossover agents:

— The first crossover operator gives priority to one parent to transmit all its objects to the offspring. A parent is selected to give its objects value to the offspring. Starting from the parent having the smallest objective value, the first crossover agent transmits all its objects to the offspring. Then, the non conflicting objects are extracted from the second parent, to be added to the configuration of the offspring.

— The second crossover operator gives chance to only one object from each parent to be added to the offspring, at each step. The second crossover agent starts from the parent which has the smallest total profit value to build the offspring. This agent copies the first object of this parent to the offspring. Then, it extracts from other parent, the next object and transmits it to the configuration of the offspring. Each selected object has to be deleted from its parent and from other parent (if another parent contains the selected object). An object, that violates the capacities constraints, is discarded. Notice that the objects, in each parent, are sorted according to their profits.

6.4 Experimentation

We implemented MAOM-MKP in Java using the multi-agent platform Jade. The code was run on a computer with a Core I5 2.5 GHz, 8GB of RAM. We have tested our algorithm using a large sized benchmark 0â1 MKP test instances. This benchmark was described in (Chu & Beasley, 1998) and it was extracted from OR-library (`http://people.brunel.ac.uk/mastjjb/jeb/info.html`). There are instances with 5, 10, and 30 constraints and 100, 250, and 500 variables. The values of the tightness ratio α for resource capacities $b_k(b_k = \alpha \sum_{j=1}^{n} a_{kj}, k = 1, 2, ..., m)$ are 0.25, 0.50 and 0.75. For each $n \times m \times \alpha$ combination, there is ten instances, that is give a total of 270 instances.

According to the experimentations, the parameters of MAOM-MKP were adjusted as follows: the number of iterations for each local search agent ($iter\_max$) was fixed to 1000. The parameter $interval$ which computes the quality of the improvement of the solution between generations was fixed to 200 for both decision-maker agent and tabu search agents. For controlling the optimization process and updating the decision matrices, the parameters g_0, g_1, g_2, q_3, q_4 and q_5 were set respectively to 2, 10, 20, 24, 2 and 4. The parameter rate μ used in updating the decision matrices was fixed to 0.9. The stopping condition is set to one hour for all the instances except hard instances (instances of $30 \times 500 \times 0.75$ combination) that have be run for 24 hours.

MAOM-MKP is compared with the best heuristic methods available in the literature presented in 6.2:

— Genetic algorithm (GA) (Chu & Beasley, 1998);

— Discrete electromagnetism-like mechanism (DEM) (Bonyadi & Li, 2012);
— Set-based particle swarm optimization (SBPSO) (Langeveld & Engelbrecht, 2012);
— Simplified binary version of the artificial fish swarm algorithm (newS-bAF-SA) (Abul Kalam Azad & Ana Maria , 2015).

The best known solutions for the MKP, as far as we know, were obtained by Vasquez and Hao (Vasquez & Hao, 2001). Then, they were improved by Vasquez and Vimont (Vasquez & Vimont, 2005). The performance of the obtained results are evaluated using the percentage (%) gap between the best objective function value and the optimal value of the LP (linear programming) relaxation. The gap is defined as:

$$Gap\% = \frac{optimal\ LP\ value - best\ objective\ value}{optimal\ LP\ value} \times 100 \tag{6.5}$$

In table 6.1, T represents the average computational time (in seconds) of a problem set. n is the number of variables for each instance and m is the number of the capacities constraints. We cite, in this table, the average results obtained in each $n \times m \times \alpha$ combination. We give some results of the proposed MAOM-MKP in Tables 6.2, 6.3, 6.4.

From table 6.1 and considered average results, we observe that the DEM approach outperforms other methods. Our proposed algorithm MAOM-MKP reaches better results for the group of instances with 30 constraints and 100 variables. Considering average results, MAOM-MKP gives better results than SBPSO. The percentage average gap is 0.93 %.

6.5 Conclusion

In this chapter, we proposed a multi-agent algorithm for the multidimensional knapsack problem. We kept the proposed generic model MAOM-COP and we adapted it to the studied problem. We selected two different neighborhood strategies from the literature for the intensification agents. Two crossover operators are used by the crossover agents. The perturbation agent helps the tabu search agents to diversify their searches by performing two different perturbation techniques. The trigged perturbation technique depends on the state of search according to the decision matrices. The experimental results show that MAOM-MKP gives high quality solutions.

Table 6.1 – Comparative results between MAOM-MKP and some of the best performing MKP approaches

Prob. set		GA		DEM		SBPSO		newS-bAFSA		MAOM-MKP	
m	n	Gap	T	Gap	T	Gap	T	Gap	T	Gap	T
5	100	0.59	20.0	**0.58**	13.0	1.11	-	0.59	14.9	0.72	35
	250	**0.14**	174.4	**0.14**	25.0	1.86	-	0.22	127.4	0.9	112
	500	**0.05**	70.5	**0.05**	95.0	2.66	-	0.17	696.6	0.98	260
10	100	**0.94**	314.4	**0.94**	19.0	1.14	-	1.00	19.5	0.97	45
	250	0.30	276.8	**0.28**	31.0	1.53	-	0.46	164.6	0.92	124
	500	0.14	734.1	**0.12**	155.0	1.86	-	0.35	860.7	0.98	560
30	100	1.69	128.5	1.68	14.0	1.50	-	1.73	44.2	**0.98**	60
	250	0.68	847.9	**0.65**	57.0	1.86	-	0.90	328.8	0.97	222
	500	0.35	1384.0	**0.34**	252.0	1.98	-	0.70	1487.3	0.97	3600
Average		0.54	438.9	**0.53**	73.4	1.72	-	0.68	416.0	0.93	557

Table 6.2 – Some results obtained by MAOM-MKP on instances with 30 constraints from MKP benchmarks

CB	OR	MOAM-MKP	CB	OR	MOAM-MKP
30.500.00	115868	114282	30.250.00	56693	55955
30.500.01	114667	112432	30.250.01	58318	57513
30.500.02	116661	114922	30.250.02	56556	55973
30.500.03	115237	113635	30.250.03	56863	56061
30.500.04	116353	114433	30.250.04	56629	55904
30.500.05	115604	114667	30.100.00	21946	21708
30.500.06	113952	112778	30.100.01	21716	21353
30.500.24	304404	302818	30.100.02	20754	20191
30.500.25	296894	294951	30.100.17	42719	42453
30.500.26	303233	301149	30.100.18	42230	41917
30.500.27	306944	305952	30.100.19	41700	41208
30.500.28	303057	302140	30.100.24	58884	58781
30.500.29	300460	299194	30.100.25	60011	59879
30.250.27	152841	151227	30.100.26	58132	57912
30.250.28	149568	148550	30.100.27	59064	58878

Table 6.3 – Some results obtained by MAOM-MKP on instances with 10 constraints from MKP benchmarks

CB	OR	MOAM-MKP	CB	OR	MOAM-MKP
10.500.00	117726	116513	10.250.22	151900	150287
10.500.01	119139	118614	10.250.23	151275	150104
10.500.02	119159	117550	10.250.24	151948	150473
10.500.03	118802	117316	10.250.27	153520	152463
10.500.05	119454	118092	10.250.26	153131	152575
10.500.06	119749	118767	10.500.00	117726	116513
10.500.07	118288	116750	10.500.01	119139	118614
10.500.08	117779	116754	10.500.05	119454	118092
10.500.09	119125	118207	10.500.06	119749	118767
10.500.25	301730	299186	10.500.07	118288	116750
10.250.00	59187	58601	10.500.08	117779	116754
10.250.01	58662	57770	10.500.09	119125	118207
10.250.02	58094	57240	10.100.10	41395	40885
10.250.21	148772	147110	10.100.11	42344	42094
10.250.22	151900	150287	10.100.12	42401	41925

Table 6.4 – Some results obtained by MAOM-MKP on instances with 5 constraints from MKP benchmarks

CB	OR	MOAM-MKP	CB	OR	MOAM-MKP
5.100.00	24381	240079	5.250.26	148607	147247
5.100.01	24274	24112	5.250.27	149772	148268
5.100.02	23551	23463	5.250.28	155075	153981
5.100.03	23534	23275	5.250.29	154662	153538
5.100.04	23991	23789	5.100.00	24381	240079
5.100.05	24613	24396	5.100.01	24274	24112
5.100.06	25591	25343	5.100.02	23551	23463
5.100.07	23410	231548	5.100.03	23534	23275
5.100.08	24216	24068	5.100.04	23991	23789
5.100.09	24411	24153	5.100.05	24613	24396
5.100.10	42757	42382	5.100.25	58959	58737
5.250.00	59312	58618	5.100.26	61538	61105
5.250.01	61472	60679	5.100.27	61520	61170
5.250.02	62130	61751	5.100.28	59453	59202
5.250.03	59446	58744	5.100.29	59965	59544

General Conclusion

Principal contributions

In this thesis, we proposed the Multi-Agent based Optimization Method for Combinatorial Optimization Problems (MAOM-COP). This method is based on multi-agent system and combines some features of several other well-established metaheuristics including tabu search, variable neighborhood search and evolutionary methods. In the first chapter, we made a brief literature review of the most popular heuristic and multi-agent based optimization approaches for combinatorial optimization problems.

In the second chapter, we presented the proposed method MAOM-COP, which explores several techniques of intensification and techniques of diversification. These techniques are manipulated by agents. In fact, tabu search agents perform intensification search by applying different neighborhood strategies. The perturbation agent and the crossover agents aim at escaping the current local optimum. During the optimization process, the decision-maker agent controls the search and decides which and when activating other agents according to the optimization state. The selection of agents to trigger is made dynamically, based on decision matrices whose values are adjusted using the reinforcement learning.

In the third chapter, we explored MAOM-COP to solve the quadratic assignment problem. The resulted MAOM-QAP is composed of the following agents: a decision-maker agent, two tabu search agents, two crossover agents and a perturbation agent. The decision-maker agent saves high quality solutions received from other agents, in an archive. Tabu search agents perform tabu search with two different strategies to explore the neighboring solutions. One of them explores the whole neighborhood swap moves to generate the best solution and the other uses a reduced search space to find an other best solution. These agents can exchange solutions with each other and they can trigger the perturbation agent to escape local optimum. A dynamic tabu tenure is used to control the degree of diversification introduced into the search. The perturbation agent applies two techniques of disturbed moves. A strong perturbation technique is made based on the archive, to create a new solution. A reduced perturbation technique makes some random perturbations in the current assignment. The crossover agents apply two informative recombination operators. The activated agents are adaptively determined based on the current search state and the decision matrices. We evaluated the performance of our algorithm by comparing it with the current best-performing approaches, using the set of QAPLIB instances. The proposed MAOM-QAP attained the best-known result for many instances.

In the forth chapter, we considered the graph coloring problem and we proposed MAOM-GCP to solve it. MAOM-GCP is an other application of MAOM-COP described in chapter 2. MAOM-GCP has the same characteristic of MAOM-COP with necessary adaptation for the GCP. Moreover, the agents have the same features and they communicate in the same way of other applications of the proposed approach. Nevertheless, the differences include the neighborhood strategies, the perturbation moves and the recombination operators. As we mention, tabu search agents are the intensification agents. They apply tabu search with two neighborhood strategies. The first tabu search agent changes the colors of conflicting vertices to produce new k-coloring. A conflicting vertex is colored with the best possible other color class. The new color class is chosen among those which are not assigned to vertices adjacent to the conflicting vertex. The second tabu search agent uses the same mechanism of selecting the best color class to be assigned to vertices as the first tabu search agent. However, the modified vertices are the adjacent of conflicting vertices and not conflicting vertices. The perturbation agent performs two perturbation moves. The reduced perturbation move is applied, when there is a slight search stagnation. From a solution received from a tabu search agent, the perturbation agent makes random moves to create a new solution. It consists in a random change of the color of a conflicting vertex of the received solution. The strong perturbation move is performed when a tabu search agent observes a deep search stagnation. The perturbation agent explores the shared archive to create a new solution. It extracts the number of occurrences of each vertex colored by each color class. Each vertex is affected to a color class which has the smallest occurrence number. The crossover agents use two different recombination operators taken from the literature. We compared the results of MAOM-GCP with best known results using DIMACS coloring benchmarks. The comparative study shows that it is able to reach best known solutions of several instances.

In the fifth chapter, we presented an application of MAOM-COP to the Winner Determination Problem given MAOM-WDP. MAOM-WDP adds the following features comparing with other WDP approaches: the choice of the best neighbor is based on the selection of several bids instead of one bid. The update of the tabu tenure is dynamically adjusted, that improves the diversification. In addition, a new technique of crossover operator, for the WDP, is introduced. This operator gives the priority to the invariants bids to stay in the new solution, in order to take the good information of parents. Tabu search agents and crossover agents explore the proposed neighborhood strategy with the existing one, while the perturbation agent applies two perturbation moves. The reduced move consists in making random modification of the solution received from tabu search agents. The strong move uses the archive to generate a new solution whose bids often appear in high-quality solutions. The computational study on a set of 500 benchmark instances shows that MAOM-WDP finds high quality results on tested benchmark instances. Another representative algorithm (TSX_WDP) is proposed to solve the WDP in the appendix of this chapter.

In the sixth chapter, we applied MAOM-COP to the multidimensional knapsack problem. As other algorithms elaborated in this thesis, MAOM-MKP has its specific characteristics. Each tabu search agent applies a strategy to explore the neighboring solutions. The first tabu search agent visits the whole neighborhood by removing an object from the current configuration and adds the best non selected object to it. The second tabu search agent selects a random object from the current configuration. Then, it replaces it by the best non selected object that improves the total profit. The perturbation agent performs two perturbations moves. The reduced perturbation move consists in performing add/remove random moves to generate a new feasible configuration, starting from the solution received from the tabu search agent, while, the strong perturbation move consists in generating a new solution. Based on the common archive, the perturbation agent selects the objects which have been less selected in the configurations belonging to the archive. When the crossover agents are activated, two different recombination operators are performed to introduce a higher degree of diversification. MAOM-MKP is tested using large sized benchmarks from OR-library. According to the comparison results with best MKP approaches, MAOM-MKP was able to reach good quality solutions.

Future research perspectives

Last years, agent paradigm has emerged as an interesting alternative for solving different optimization problems. In fact, specific features of software agents, like autonomy, reactiveness or ability to work in teams, provide a promising tool for solving optimization problems. This thesis focuses on a multi-agent approach, dedicated for solving hard combinatorial optimization problems. The proposed approach belongs to the group of approaches combining metaheuristics, optimization problem solving, multi-agent system and learning paradigms. The main goal of the proposed MAOM-COP is to investigate the multi-agent system to create cooperative multi-search methods. These methods explore several existing metaheuristics and their techniques. The reinforcement learning is used to dynamically select activating agents according to the state of search. Several interesting directions that improve the performance of the proposed approach, can be envisaged in the future.

First, concerning the optimization process, one possibility which was not investigated during this thesis is to exploit other local search algorithms like iterated local search or simulated annealing, for intensification agents. The number of intensification agents and diversification agents depends on the problem solved. We can apply the proposed approach to other problems. Other improvement that relates to search process is to activate several simultaneous independent search agents. Each search agent starts with a different initial solution from the common archive that makes different trajectory in the search space. It is the search space decomposition which is based on the idea of dividing the whole search space into smaller subspaces, solving the resulting subproblems.

Second, we may focus on the reinforcement learning. The proposed approach used intelligent agents that carry out some set of operations based on decision matrices whose values are regulated by reinforcement learning. Indeed, better learning, that can depend on the solved problem, will make the agents more informed and allow them to choose the most appropriate operation to activate.

Finally, the current version of the proposed model can be considered as a proof-of-concept implementation. One interesting perspective to improve the computational efficiency of the proposed approach is to envisage a dedicated implementation. Other possibility is to use parallel and distributed programming in the design and the implementation of metaheuristics to speed up the search. Especially reducing the search time is important in case of complex optimization problems where the search time is critical.

References

Abul Kalam Azad, M.D., & Ana Maria, A.C.R. (2015). Solving Large 0-1 Multidimensional Knapsack Problems by a New Simplified Binary Artificial Fish Swarm Algorithm. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 14(3), 313–330.

Atkinson, J.B. (1998). A greedy randomized search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *J. Oper. Res. Soc*, 49, 700–708.

Autry, B. (2008). University Course Timetabling with Probability Collectives. *Master's Thesis, Naval Postgraduate School Montery, CA, USA*.

Avanthay, C., Hertz, A., & Zufferey, N. (2003). A Variable Neighborhood Search for Graph Coloring. *Eur. J. Oper. Res.*, 151, 379–388.

Avila-Abascal, P., & Talukdar, S.N. (1996). Cooperative Algorithms and Abductive Causal Networks for the Automatic Generation of Intelligent Substation Alarm Processors. *In Proceedings of International Conference on Intelligent Systems Applications to Power Systems, January 1996*, 7–1, Orlando, FL.

Bae, C.B., Cho, K.Y., Ji, S.H., Kim, B.Y., & Suh, M.W. (2009). Multi-agent based traffic simulation and integrated control of freeway corridors: Part 2 integrated control optimization. *Journal of Mechanical Science and Technology*, 23(5), 1374–1382.

Barbucha, D., & Jędrzejowicz P. (2007). An Agent-Based Approach to Vehicle Routing Problem. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 1(2), 301–306.

Barbucha, D. (2013). Solving Instances of the Capacitated Vehicle Routing Problem Using Multi-agent Non-distributed and Distributed Environment. *Agent-Based Optimization Studies in Computational Intelligence*, 456, 55–75.

Basar, T., & Olsder, G.J. (1995). Dynamic Non-Cooperative Game Theory. *Academic Press, London/New York, January 1995*, Revised 2nd edition of 1982 book with the same title.

Baum, E.B. (1987). Towards practical 'neural' computation for combinatorial optimization problems. *In AIP Conference Proceedings 151 on Neural Networks for Computing, March 1987*, 53–58, Woodbury, NY, USA.

Bean, J.C. (1994). Genetics and random keys for sequencing and optimization. *ORSA J Comput*, 6(2), 154–160.

Bell, J.E., & McMullen, P.R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Adv. Eng. Inform.*, 1, 41–48.

Benlic, U., & Hao, J.K. (2011). A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation*, 15(5), 624–472.

Benlic, U., & Hao, J.K (2013). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9), 4800–4815.

Benlic, U., & Hao, J.K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 42(1), 584–595.

Bhadra, S., Shakkotai, S., & Gupta, P. (2006). Min-cost selfish multicast with network coding. *IEEE Transactions on Information Theory*, 52(11), 5077–5087.

Bieniawski, S.R. (2005). Distributed Optimization and Flight Control Using Collectives. *PhD Dissertation, Stanford University, CA, USA*.

Blochliger, I., & Zufferey, N. (2008). A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comput. Oper. Res.*, 35(3), 960–975.

Bonyadi, M.R., & Li, X. (2012). A new discrete electromagnetism-based metaheuristic for solving the multidimensional knapsack problem using genetic operators. *Oper. Res. - Int. J.*, 12, 229–252.

Boughaci, D., Belaid, B., & Habiba, D. (2009). A memetic algorithm for the optimal winner determination problem. *Soft Computing*, 13(8-9), 905-917.

Boughaci, D., Belaid, B., & Habiba, D. (2010). Local Search Methods for the Optimal Winner Determination Problem in Combinatorial Auctions. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 9(2), 165–180.

Boyer, V., Elkihel, M., & El Baz, D. (2009). Heuristics for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 199, 658–664.

Braysy, O. (2001). Genetic algorithms for the vehicle routing problem with time windows. *Technical Report, January 2001*, University of Vaasa, Vaasa, Finland.

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22(4), 251–256.

Burkard, R.E. (1991). Locations with spatial interactions: The quadratic assignment problem. *In Mirchandani and L. Francis (Eds.), Discrete Location Theory*, New York, USA.

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

Carlton, W., & Barnes, J. W. (1996). Solving the Traveling Salesman Problem with Time Windows Using Tabu Search. *IIE Trans*, 28, 617–629.

Chalupa, D. (2011). Population-based and learning-based metaheuristic algorithms for the graph coloring problem. *In Krasnogor, N., Lanzi, P.L. (eds.), GECCO, July 2011*, 465–472, Dublin, Ireland.

Changdar, C., Mahapatra, G.S., & Pal, R.K. (2013). An Ant colony optimization approach for binary knapsack problem under fuzziness. *Applied Mathematics and Computation*, 223, 243–253.

Chen, S.Y., Talukdar, S.N., & Sadeh, N.M. (1993). Job-Shop-Scheduling by a Team of Asynchronous Agents. *In Workshop on Knowledge-Based Production, Scheduling and Control, IJCAI-93, August 1993*, Chambery, France.

Chiarandini, M., & Stützle, T. (2002). An application of iterated local search to graph coloring. *In Johnson, D.S., Mehrotra, A., Trick, M. (eds.), Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, September 2002*, 112â125, Ithaca, New York, USA.

Chu, P.C., & Beasley, J.E. (1998). A genetic algorithm for the multidimensional knapsack problem. *J. Heuristics*, 4, 63–86.

Colorni, A., Dorigo, M., Maniezzo, V., & Trubian, M. (1994). Ant system for job-shop scheduling. *Belgian J. Oper. Res. Stat. and Comput. Sci.*, 34, 39–54.

Cramton, P., Shoham, Y., & Steinberg, R. (2006). Combinatorial auctions. *MIT Press, Cambridge*.

De Souza, P. (1993). Asynchronous Organizations for Multi-Algorithm Problems. *Ph. D. dissertation, Dept. of Electrical and Computer Engineering*, Carnegie Mellon University, Pittsburgh, PA.

DeWerra, D., Eisenbeis, C., Lelait, S., & Marmol, B. (1999). On a graph-theoretical model for cyclic register allocation. *Discrete Applied Mathematics*, 93(2-3), 191–203.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). Positive feedback as a search strategy. *Technical Report 91-016, Politecnico di Milano, Italy*.

Dorne, R., & Hao, J.K. (1998). Tabu Search for graph coloring, T-coloring and Set T-colorings. *In Osman, I.H., et al. (eds.), Metaheuristics 1998: Theory and Applications. ch. 3. Kluver Academic Publishers*.

Drexl, A. (1988). A simulated annealing approach to the multiconstraint zeroâone knapsack problem. *Computing*, 40, 1–8.

Drezner. (2008). Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers and Operations Research*, 35(3), 717–736.

Duarte, A., Escudero, L.F., Marti, R., Mladenovic, N., Pantrigo, J.J., & Sanchez-Oro, J. (2012). Variable Neighborhood Search for the Vertex Separation Problem. *Comput. and Oper. Res.*, 39(12), 3247–3255.

Duman, E., & Or, I. (2007). The quadratic assignment problem in the context of the printed circuit board assembly process. *Computers and Operations Research*, 34(1), 163-179.

Fabiunke, M. (1999). Parallel distributed constraint satisfaction. *In Proc. Intern. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA-99), July 1999*, 1585–1591, Monte Carlo Resort, Las Vegas, Nevada, USA.

Ferber, J. (1999). Multi-agent systems: an introduction to distributed artificial intelligence *Addison-Wesley (Eds.): Multi-agent systems: an introduction to distributed artificial intelligence*.

Fitzpatrick, S., & Meertens, L. (2001). An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. *In Proc. 1st Symp. on Stochastic Algorithms: Foundations and Applications, December 2001*, 49–64, Berlin, Germany.

Fleurent, C., Ferland, J.A. (1996). Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63, 437–461.

Fleurent, C., & Glover F. (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS J. Comput*, 11, 198–204.

Fréville, A. (2004). The multidimensional 0â1 knapsack problem: an overview. *Eur. J. Oper. Res.*, 155, 1–21.

Fujishima, Y., Leyton-Brown, K., & Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: optimal and approximate approaches. *In 16th International Joint Conference on Artificial Intelligence, August 1999*, 48–53, Stockholm, Sweden.

Galinier, P., & Hao, J.K. (1999). Hybrid evolutionary algorithms for graph coloring. *J. Comb Optim.*, 3(4), 379–397.

Galinier, P., Hertz, A., & Zufferey, N. (2008). An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2), 267–279.

Gamache, M., Hertz, A., & Ouellet, J.O. (2007). A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers and Operations Research*, 4(8), 2384–2395.

Garey, M.R., Johnson, D.S., & So, H.C. (1976). An application of graph coloring to printed circuit testing. *IEEE Transactions on Circuits and Systems*, 23, 591–599.

Garey, M.R., & Johnson, D.S. (1979). Computers and intractability: A guide to the theory of NP-completeness. *W.H. Freeman and Company, San Francisco*.

Gavish, B., & Pirkul, H. (1979). Allocation of databases and processors in a distributed computing system. *In J. Akoka (ed.), Management of Distributed Data Proc., North-Holland*, 215–231.

Gavish, B., & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31, 78–105.

Gendron, B., Hertz, A., & St-Louis, P. (2007). On edge orienting methods for graph coloring. *J. of Comb. Optim.*, 13(2), 163–178.

Gilmore, P.C., & Gomory, R.E. (1966). The theory and computation of knapsack functions. *Operations Research*, 14, 1045–1075.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Comput. Open Res*, 13, 533–549.

Guo, Y., Lim, A., Rodrigues, B., & Zhu, Y. (2006). Heuristics for a bidding problem. *Computers and Operations Research*, 33(8), 2179–2188.

Guo, Y., Goncalves, G., & Hsu, T. (2013). A Multi-agent Based Self-adaptive Genetic Algorithm for the Long-term Car Pooling Problem. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 12(1), 45–66.

Hao, J.K., & Wu, Q. (2012). Improving the extraction and expansion method for large graph coloring. *Discrete Applied Mathematics*, 2397–2407.

Hanafi, S., & Fréville, A. (1998). An efficient tabu search approach for the 0â1 multidimensional knapsack problem. *Eur. J. Oper. Res.*, 106, 659–675.

Hanafi, S., & Wilbaut, C. (2008). Scatter search for the 0-1 multidimensional knapsack problem. *Journal of Mathematical Modelling and Algorithms*, 7, 143–159.

Hanna, L.D., & Cagan, J. (2009). Evolutionary Multi-Agent Systems: An Adaptive and Dynamic Approach to Optimization. *Journal of Mechanical Design*, 131(1).

Hertz, A., & de Werra, D. (1987). *Using tabu search techniques for graph coloring*. *Comput.*, 39(4), 345–351.

Hertz, A., Plumettaz, M., & Zufferey, N. (2008). Variable space search for graph coloring. *Discret. Appl. Math.*, 156(13), 2551–2560.

Holland, J.H. (1975). Adaption in natural and artificial systems. *The University of Michigan Press, Ann Harbor*.

Holland, A. , O'sullivan, B. (2004). Towards fast vickrey pricing using constraint programming. *Artificial Intelligence Review*, 21(3-4), 335–352.

Hoos, HH., & Boutilier, C. (2000). Solving combinatorial using stochastic local search. *In Proceedings of the 17th National Conference on Artificial Intelligence, August 2000*, 22–29, Austin, Texas.

Huang, C.F., Bieniawski, S., Wolpert, D., & Strauss, C.E.M. (2005). A Comparative Study of Probability Collectives Based Multiagent Systems and Genetic Algorithms. *In Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO, June 2005*, 751–752, Washington, USA.

Huang, C.F., & Chang, B.R. (2010). Probability Collectives Multi-agent Systems: A Study of Robustness in Search. *In Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.), ICCCI 2010, Part II. LNCS, November 2010*, 6422, 334–343, Kaohsiung, Taiwan.

James, T., Rsego, C., & Glover, F. (2009). A Cooperative Parallel Tabu Search Algorithm for the Quadratic Assignment Problem. *European Journal of Operational Research*, 195(3), 810–826.

Jawad, A., Teodor, GC., Michel, G., & Monia, R. (2007). Combinatorial auctions. *Annals of Operational Research*, 153(1), 131–164.

Jędrzejowicz, P., & Ratajczak-Ropel, E. (2013). Triple-Action Agents Solving the MRCPSP/Max Problem. *Agent-Based Optimization Studies in Computational Intelligence*, 456, 103–122.

Jędrzejowicz, P., & Wierzbowska, I. (2006). Jade-based A-team environment. *In Proceedings of the 6th International Conference on Computational Science, May 2006*, 719-726, Reading, UK.

Johnson, D.S., Trick, M. (1996). Cliques, coloring, and satisfiability: second DIMACS implementation challenge. *Dimacs Series in Discrete Mathematics and Theoretical Computer Science (Book 26), American Mathematical Society*.

Kennedy, J., Eberhart, R.C. (1995). Particle swarm optimization. *In Proceedings of IEEE Int. Conf. Neural Netw., December 1995*, 1942–1948, Piscataway, NJ, USA.

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing. *Sci*, (220), 671-680.

Klemperer, P. (2004). Auctions: Theory and Practice *Princeton, N.J.: Princeton University Press*.

Kong, M., Tian, P., Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional knapsack problem. *Comput. Oper. Res.*, 35, 2672–2683.

Kulkarni, & A.J., Tai, K. (2008). Probability Collectives for Decentralized, Distributed Optimization: A Collective Intelligence Approach. *In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, October 2008*, 1271–1275, Singapore, .

Kulkarni, A.J., & Tai, K. (2009). Probability Collectives: A Decentralized, Distributed Optimization for Multi-Agent Systems. *In Mehnen, J., Koeppen, M., Saad, A., Tiwari, A. (eds.), Applications of Soft Computing*, 441–450.

Kulkarni, A.J., & Tai, K. (2010). Probability Collectives: A Multi-Agent Approach for Solving Combinatorial Optimization Problems. *Applied Soft Computing*, 10(3), 759–771.

Kulkarni, A.J., Tai, K. (2011). A Probability Collectives Appraoch with a Feasibility-based Rule for Constrained Optimization. *Applied Computational Intelligence and Soft Computing*, 2011(3859).

Langeveld, J., & Engelbrecht, A.P. (2012). Set-based particle swarm optimization applied to the multidimensional knapsack problem. *Swarm Intell.*, 6, 297–342.

Lau, H.C., & Goh, Y.G. (2002). An intelligent brokering system to support multiagent web-based for thparty logistics. *In Proceedings of the 14th International Conference on Tools with Artificial Intelligence, November 2002*, 54-61, Washington, DC, USA.

Lee, H., Murthy, S., Haider, W., & Morse,D. (1995). Primary Production Scheduling at Steel making Industries. *IBM report*, 40(2), 231.

Leighton, F.T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6), 489–506.

Leyton-Brown, K., Shoham, Y., & Tennenholtz, M. (2000). An algorithm for multiunit combinatorial auctions. *In Proceedings of the 17th National Conference on Artificial Intelligence, August 2000*, Austin, Texas.

Li, H., Jiao, Y., Zhang, L., & Gu, Z. (2006). Genetic algorithm based on the orthogonal design for multidimensional knapsack problems. *Advances in Natural Computation. Springer Berlin/Heidelberg*, 696–705.

Lim, A., & Wang, F. (2004). Meta-heuristics for robust graph coloring problem. *In Proceedings of 16th IEEE International Conference on Tools with Artificial Intelligence, November 2004*, 514–518, Boca Raton, FL, USA.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.*, 44, 2245–2269.

Lu, Z., Hao, J.K. (2010). A memetic algorithm for graph coloring. *Eur. J. Oper. Res.*, 203(1), 241–250.

Malaguti, E., Monaci, M, Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2), 302–316.

Markowitz, H.M., & Manne, A.S. (1957). On the solution of discrete programming problems. *Econometrica*, 25, 84–110.

Martin, O., & Otto, S.W. (1996). Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63, 57–75.

Meignan, D., Koukam, A., & Créput, J.C. (2010). Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6), 859–879.

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

Misevicius, A. (2004). An improved hybrid genetic algorithm: New results for the quadratic assignment problem. *Knowledge Based Systems*, 17(2-4), 65–73.

Misevicius, A., Lenkevicius, A., & Rubliauskas, D. (2006). Iterated tabu search: An improvement to standard tabu search. *Information Technology and Control*, 35(3), 187–197.

Mladenovic, N., & Hansen, P. (1997). Variable Neighborhood Search. *Comput. Oper. Res.*, 24, 1097–1100.

Mladenovic, N., Urosevic, D., Perez-Brito, D., & Garcia-Gonzalez, C. G. (2010). Variable neighborhood search for bandwidth reduction. *European Journal of Operational Research*, 200(1), 14–27.

Moalic, L., & Gondran, A. (2015). The new memetic algorithm HEAD for graph coloring: an easy way for managing diversity. em In Ochoa and Chicano (Eds.), EvoCOP2015, Lecture Notes in Computer Science, April 2015, 9026, 173–183, Copenhagen, Denmark.

Modia P.G., Shena W.M., Tambea M., & Yokoo M. (2005). Adopt: asynchronous distributed constraint optimization with quality guarantees. *Journal of Artificial Intelligence, Distributed Constraint Satisfaction*, 161(1â2), 149–180.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Technical report, Tech. Rep. No. 790, Caltech Concurrent Computation Program*, California Institue of Technology. *Eur. J. Oper. Res.*, 200, 14–27.

Nisan, N. (2000). Bidding and allocation in combinatorial auctions. *In Proceedings of ACM Conference on Electronic Commerce (EC'00). ACM Press, October 2000*, 1–12, Minneapolis, USA.

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

Papadimitriou, C.H., & Steiglitz. K (1998). Combinatorial Optimization : Algorithms and Complexity. *Edition: Dover Books on Computer Science*.

Porumbel, D.C., Hao, J.K., & Kuntz, P. (2009). Diversity control and multi-parent recombination for evolutionary graph coloring algorithms. *In C. Cotta, P. Cowling (Eds.), EvoCOP 2009, Lecture Notes in Computer Science, April 2009*, 5482, 121-132, Tübingen, Germany.

Porumbel, D.C., Hao, J.K., & Kuntz, P. (2009). Position-Guided Tabu Search Algorithm for the Graph Coloring Problem. *In Stützle, T. (ed.)*, 3(5851), 148–162.

Porumbel, D.C., Hao, J.K., & Kuntz, P. (2010). A Search Space "Cartography" for Guiding Graph Coloring Heuristics. *Computers Operations Research*, 37(4), 769–778.

Prestwich, S. (2002). Coloration neighbourhood search with forward checking. *Annals of Mathematics and Artificial Intelligence*, 34(4), 327–340.

Puchinger, J., & Raidl, G. R. (2005). Relaxation Guided Variable Neighborhood Search. *In Proceedings of the XVIII Mini EURO Conference on VNS, November 2005*, Tenerife, Spain.

Puchinger, J., Raidl, G.R., & Gruber, M. (2005). Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In Proceeding of the 6th Metaheuristics International Conference, August 2005, 775–780, Vienna, Austria.

Rachlin, J., Wu, F., Murthy, S., Talukdar, S.,Sturzenbecker, M., Akkiraju, R., Fuhrer, R., Aggarwal, A., Yeh, J., Henry, J., & Jayaraman, J. (1996). Forest View: A System For Integrated Scheduling In Complex Manufacturing Domains. *IBM report*, 0.

Reinelt, G. (1994). The Traveling Salesman, Computational Solutions for TSP Applications. *Lecture Notes in Computer Science. Springer*, 840.

Rothkopf, MH., Pekee, A., Ronald, M. (1998). Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1131-1147.

Sakawa, M., & Kato, K. (2003). Genetic algorithms with double strings for 0â1 programming problems. *Eur. J. Oper. Res.*, 144, 581–597.

Sandholm, T., & Suri, S. (2010). Improved optimal algorithm for combinatorial auctions and genericizations. *In Proceedings of the 17th National Conference on Artificial Intelligence, August 2000*, 90–97, Austin, Texas.

Sandholm, T., Suri, S., Gilpin, A., & Levine, D. (2001). CABoB: a fast optimal algorithm for combinatorial auctions. *In Proceedings of the International Joint Conferences on Artificial Intelligence, August 2001*, 1102-1108, Seattle, WA, USA.

Sghir, I., Hao, J.K., ben Jaafar, I., & Ghédira, K. (2013). A Recombination-Based Tabu Search Algorithm for the Winner Determination Problem. *Selected and revised paper in the International Conference of Evolution Artificielle (EA 2013),* *Bordeaux, France, October 2013. Lecture Notes in Computer Science*, 8752, 157–167.

Sghir, I., Hao, J.K., ben Jaafar, I., & Ghédira, K. (2015). A Distributed Hybrid Algorithm for the Graph Colorin g Problem. *Selected and revised paper in the International Conference of Evolution Artificielle (EA 2015), Lyon, France, October 2015. Lecture Notes in Computer Science*, 9554, 205–218.

Sghir, I., Hao, J.K., ben Jaafar, I.,& Ghédira, K. (2015). A Multi-Agent Based Optimization Method applied for the Quadratic Assignment Problem. *Expert Systems With Applications Journal*, 42(23), 9252–9263.

Shih, W. (1979). A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem. *Journal of the Operational Research Society*, 30, 369–378.

Sislak, D., Volf, P., Pechoucek, M., & Suri, N. (2011). Automated Conflict Resolution Utilizing Probability Collectives Optimizer. *IEEE Transactions on Systems, Man and Cybernetics: Applications and Reviews*, 41(3), 365–375.

Sivanandam, S.N., Sumathi, S., & Hamsapriya, T. (2005). A hybrid parallel genetic algorithm approach for graph coloring. *Int. J. Knowl. Based Intel. Eng. Syst.*, 9, 249–259.

Smith, D.H., Hurley, S., & Thiel, S.U. (1998). Improving heuristics for the frequency assignment problem. *European Journal of Operational Research*, 107(1), 76–86.

Smyrnakis, M., & Leslie, D.S. (2009). Sequentially Updated Probability Collectives. *In Proceedings of 48thIEEE Conference on Decision and Control and 28th Chinese Control Conference, December 2009*, 5774–5779, Shanghai, China.

Stützle, T. (1998). Applying iterated local search to the permutation flow shop problem. *Technical Report AIDA-98-04*, FG Intellektik, TU Darmstadt.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3), 1519–1539.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17, 443–455.

Talukdar, S. N., & Ramesh, V.C. (1994). A Multi-Agent Technique for Contingency Constrained Optimal Power Flows. *In Proceedings of the Power Industry Computer Applications Conference (PICA-93), IEEE Trans. on Power Systems, May, 1993*, 9(2), 855–861, Phoenix, USA.

Talukdar, S.N., Souza, P. de, & Murthy S. (1993). Organizations for Computer-Based Agents. *Engineering Intelligent Systems*, 1(2), 56-69.

Talukdar, S., Baerentzen, L., Gove, A., & De Souza, P. (1996). Asynchronous Teams: Cooperation Schemes for Autonomous Agents. *Journal of Heuristics*, 4(4), 295–321.

Tate, D.M., & Smith, A.E. (1995). A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, 2(1), 73–83.

Titiloye, O., & Crispin, A. (2011). Graph Coloring with a Distributed Hybrid Quantum Annealing Algorithm. *In J. O'Shea, N. Nguyen, K. Crockett, R. Howlett, L. Jain (eds.), Agent and Multi-Agent Systems: Technologies and Applications*, 553-562.

Trick, M.A., & Yildiz, H. (2007). A Large Neighborhood Search Heuristic for Graph Coloring. *In Van Hentenryck, P., Wolsey, L.A. (eds.), CPAIOR, May 2007*, 4510, 346–360, Brussels, Belgium.

Tsen, C. K. (1995). Solving Train Scheduling Problems Using A-Teams. *Ph.D. dissertation, Electrical and Computer engineering Department*, Carnegie Mellon University, Pittsburgh, USA.

Vasirani, M., Ossowski, S. (2008). Collective-Based Multiagent Coordination: A Case Study. *In Artikis, A., OâHare, G.M.P., Stathis, K., Vouros, G.A. (eds.), ESAW 2007. LNCS (LNAI), October 2007*, 4995, 240–253, Athens, Greece.

Vasquez, M., & Hao, J.K. (2001). A hybrid approach for the 0-1 multidimensional knapsack problem. *In Proceeding of the 17th Int. Joint Conference on Artificial Intelligence, August 2001*, 328–333, Seattle, Washington, USA.

Vasquez, M., & Vimont, Y. (2005). Improved results on the 0â1 multidimensional knapsack problem. *Eur. J. Oper. Res.*, 165, 70–81.

Vries de S,Vohra, R. (2003). Combinatorial auctions a survey. *INFORMS Journal on Computing*, 15, 284–309.

Wang, Y., Lu, Glover, F., & Hao, J.K. (2013). Backbone guided tabu search for solving the UBQP problem. *Journal of Heuristics*, 19(4), 679–695.

Wolpert, D.H., & Tumer, K. (1999). An introduction to Collective Intelligence. *Technical Report, NASA ARC-IC-99-63, NASA Ames Research Center*.

Wolpert, D.H., Antoine, N.E., Bieniawski, S.R., & Kroo, I.R. (2004). Fleet Assignment Using Collective Intelligence. *In Proceedings of the 42nd AIAA Aerospace Science Meeting Exhibit, January 2004*, Reno, Nevada.

Wolpert, D.H. (2006). Information Theory–The Bridge Connecting Bounded Rational Game Theory and Statistical Physics. *In Braha, D., Minai, A.A., Bar-Yam, Y. (eds.), Complex Engineered Systems*, 262–290.

Wolpert, D.H., Strauss, C.E.M., & Rajnarayan, D. (2006). Advances in Distributed Optimization using Probability Collectives. *Advances in Complex Systems*, 9(4), 383–436.

Wu, Q., & Hao, J.K. (2012). Coloring large graphs based on independent set extraction. *Computers and Operations Research*, 39, 283–290.

Wu, Q., & Hao, J.K. (2015). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, 42(1), 355–365.

Xu, K., & Liu, H. (2006). A Multi-Agent Particle Swarm Optimization Framework with Applications. *1st International Symposium on Pervasive Computing and Applications, August 2006*, 1–6, Urumchi, Xinjiang, China.

Yeoh W., Felner A., & Koenig S. (2010). BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *Journal Of Artificial Intelligence Research*, 38, 85–133.

Zhang, W., Wang, G., & Wittenburg, L. (2002). Distributed stochastic search for distributed constraint satisfaction and optimization: Parallelism, phase transitions and performance. *In Proceedings of AAAI-02 Workshop on Probabilistic Approaches in Search, July 2002*, 53–59, Edmonton, Alberta, Canada.

Zhang, W., Xing, Z., Wang, G., & Wittenburg, L. (2003). An Analysis and Application of Distributed Constraint Satisfaction and Optimization Algorithms in Sensor Networks. *In Proceedings of the second international joint conference on Autonomous agents and multiagent systems, AAMAS '03, July 2003*, 185–192, Melbourne, VIC, Australia.

Zou, D., Gao, L., & Li, S. (2011). Solving 0-1 knapsack problem by a novel global harmony search algorithm. *Appled Soft Computing*, 11, 1556–1564.

Zufferey, N., Amstutz, P., & Giaccari, P. (2008). Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11(4), 263–277.

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

université angers

Thèse de Doctorat

Inès Sghir

Une méthode d'optimisation à base de système multi-agents pour l'optimisation combinatoire

A Multi-Agent based Optimization Method for Combinatorial Optimization Problems

Résumé

Nous élaborons une approche multi-agents pour la résolution des problèmes d'optimisation combinatoire nommée MAOM-COP. Elle combine des métaheuristiques, les systèmes multi-agents et l'apprentissage par renforcement. Les heuristiques manquent d'une vue d'ensemble sur l'évolution de la recherche. Notre objectif consiste à utiliser les systèmes multi-agents pour créer des méthodes de recherche coopératives. Ces méthodes explorent plusieurs métaheuristiques. MAOM-COP est composée de plusieurs agents qui sont l'agent décideur, les agents intensificateurs et les agents diversificateurs (agents croisement et agent perturbation). A l'aide de l'apprentissage, l'agent décideur décide dynamiquement quel agent à activer entre les agents intensificateurs et les agents croisement. Si les agents intensificateurs sont activés, ils appliquent des algorithmes de recherche locale. Durant leurs recherches, ils peuvent s'échanger des informations, comme ils peuvent déclencher l'agent perturbation. Si les agents croisement sont activés, ils exécutent des opérateurs de recombinaison. Nous avons appliqué MAOM-COP sur les problèmes suivants : l'affectation quadratique, la coloration des graphes, la détermination des gagnants et le sac à dos multidimensionnel. MAOM-COP possède des performances compétitives par rapport aux algorithmes de l'état de l'art.

Mots clés
Multi-agents, recherche coopérative, optimisation combinatoire, intensification, diversification, métaheuristique.

Abstract

We elaborate a Multi-Agent based Optimization Method for Combinatorial Optimization Problems named MAOM-COP. It combines metaheuristics, multi-agent systems and reinforcement learning. Although the existing heuristics contain several techniques to escape local optimum, they do not have an entire vision of the evolution of optimization search. Our main objective consists in using the multi-agent system to create intelligent cooperative methods of search. These methods explore several existing metaheuristics. MAOM-COP is composed of the following agents: the decision-maker agent, the intensification agents and the diversification agents which are composed of the perturbation agent and the crossover agents. Based on learning techniques, the decision-maker agent decides dynamically which agent to activate between intensification agents and crossover agents. If the intensifications agents are activated, they apply local search algorithms. During their searches, they can exchange information, as they can trigger the perturbation agent. If the crossover agents are activated, they perform recombination operations. We applied MAOM-COP to the following problems: quadratic assignment, graph coloring, winner determination and multidimensional knapsack. MAOM-COP shows competitive performances compared with the approaches of the literature.

Key Words
Multi-agent, cooperative search, combinatorial optimization, intensification, diversification, metaheuristics.