Learning-driven optimization approaches for combinatorial search problems

Yangming Zhou

▶ **To cite this version:**

Yangming Zhou. Learning-driven optimization approaches for combinatorial search problems. Computation and Language [cs.CL]. Université d'Angers, 2017. English. NNT : 2017ANGE0100 . tel-03340381

HAL Id: tel-03340381
https://theses.hal.science/tel-03340381

Submitted on 10 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE

université angers

Thèse de Doctorat

Yangming ZHOU

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
Label europen
sous le sceau de l'Université Bretagne Loire

École doctorale : 503 (STIM)

Discipline : Informatique, section CNU 27
Unité de recherche : Laboratoire d'Études et de Recherches en Informatique d'Angers (LERIA)

Soutenue le 15 decembre 2017
Thèse n° : 1

Learning-Driven Optimization Approaches for Combinatorial Search Problems

JURY

Rapporteurs :	**M^me^ Clarisse DHAENENS**, Professeur, Université Lille 1
	M^me^ Christel VRAIN, Professeur, Université d'Orléans
Examinateur :	**M. Jean-Charles BILLAUT**, Professeur, Université de Tours
Directeur de thèse :	**M. Jin-Kao HAO**, Professeur, Université d'Angers
Co-directrice de thèse :	**M^me^ Béatrice DUVAL**, Professeur, Université d'Angers

Acknowledgement

I would like to thank many people who have helped me during my PhD study. Without them, I could not have completed this thesis.

First of all, I would like to express my greatest gratitude to my advisor Prof. Jin-Kao Hao. He has been a constant source of support, encouragement and inspiration throughout my four years PhD study. With his warm personality and character, he has been an invaluable friend for me. It was an honor and pleasure to be one of his PhD students.

My sincere gratitude also goes to Prof. Bréatrice Duval for being the second advisor of my thesis. Thank for her enthusiasm, support and scientific guidance she gave me in the past four years. Not only has she been there to discuss work-related topics, but more importantly, she has been there as a friend.

I have been also very fortunate to have professor Jean-Charles Billaut in my thesis committee.

I would like to thank professors Clarisse Dhaenens and Christel Vrain. They read the whole thesis and made important comments on its content. This dissertation benefited greatly from their comments and suggestions.

Many of my research papers during the PhD study are results of cooperation. I would like to thank all co-authors of my papers. Specially, I also would like to appreciate Prof. Fred Glover for the ever fruitful and constructive discussion with him. The discussion with him always makes scientific research full of sunlight.

I am grateful to Dr. Yuning Chen, Dr. Yan Jin, Dr Ines Sghir and Dr. Xavier Schepler for the many discussions about my research work. They were always ready to share their knowledge and experience, and give suggestions towards my research project.

I offer my special thanks to Catherine Pawlonski, Christine Bardaine, Eric Girardeau and Marie-Christine Welsch. They not only taught me the necessary rules and technical skills in the lab, but also gave me so many helpful suggestions.

I would like to acknowledge, whether named or not, I thank all my colleagues and friends at the LERIA with whom I have spent a great time in the past four years.

I wish to thank the China Scholarship Council (CSC) for providing me with financial support throughout the four years in France.

Finally, I want to express my deepest appreciation to my parents and two elder sisters for always supporting me in my efforts. But most of all I would like to thank my wife Man Li for her love, devotion and understanding and for constantly believing in me.

Contents

General Introduction

Motivation

Combinatorial search problems are ubiquitous in many areas of science and engineering. Prominent examples include finding shortest or cheapest round trips in graphs or finding models of propositional formula. Other well-known combinatorial examples are encountered in scheduling, timetabling, resource allocation, production planning and computer-aided design. Most of these problems are computationally difficult and require considerable expertise and computational time.

To handle these problems, modern heuristic approaches such as evolutionary algorithms and stochastic local search algorithms are among the most relevant solution approaches. During the last two decades, these methods have been successfully applied to a wide variety of hard combinatorial search problems in practical applications. With the continual increase of the problem scale and complexity of the available data, it becomes increasingly urgent to study new solution methods capable of exploring very large scale space in an informed and intelligent way. These new search methods will be able to find high quality solutions to large-scale problems with reasonable computing time and handle problems of very high complexity that can hardly be tackled by the existing methods.

One increasing popular way is to incorporate knowledge into heuristic search approaches. It has been shown that machine learning techniques could be helpful during all the phases of the heuristic search algorithms, such as to evaluate the objective function, to build the starting solution or initial population, to manage the population, to incorporate knowledge in the operators, to set the suitable parameter values, and even to perform algorithm configuration and algorithm selection based on features extracted from instances. In this thesis, we are interested in designing learning-driven heuristic optimization approaches to solve hard combinatorial optimization problems. Each proposed approach will be thoroughly evaluated with extensive computational experiments.

Objectives

The main objective of this thesis will be the design, implementation and validation of new intelligent search methods based on learning techniques. In this thesis, we will investigate the benefit offered by different machine learning techniques when machine learning is hybridized with heuristic approaches. The main objective can be further divided into several specific objectives:

— Investigate and classify heuristic optimization methods using machine learning techniques.

— Design single trajectory search methods incorporating machine learning techniques (e.g., probability learning) to guide the search process, resulting in high performance metaheuristic algorithms.

— Develop evolutionary search methods integrating machine learning techniques (e.g., opposition-based learning) to enhance the population diversity and improve evolutionary search.

— Devise data mining driven heuristic search approaches, which use association analysis techniques (e.g., frequent patter mining) to mine useful information from high-quality solutions collected from the previous search, thus guiding the search to promising search region.

— Evaluate the proposed methods based on some significant combinatorial search problems from different families, including grouping problems such as Graph Coloring Problem (GCP); subset selection problems such as Maximum Diversity Problem (MDP); and permutation problems such as Quadratic Assignment Problem (QAP).

— Improve the proposed learning-driven heuristic search approaches and apply them to solve some relevant and challenging combinatorial search problems.

Contributions

In this thesis, we have proposed three learning-driven heuristic optimization approaches. Our proposed approaches combined different learning techniques with heuristic algorithms for solving three important categories of combinatorial optimization problems, respectively. The main contributions of this thesis are summarized as follows:

— We make a brief survey on researches which use machine learning techniques to help heuristic algorithms. We unify and organize the related literature according to the different purposes of using machine learning techniques, such as to improve the quality of solutions, to speed up the search process, to optimize heuristic algorithms parameters, and to conduct algorithm selection.

— For the grouping problems such as GCP, we propose a *Probability learning based Local Search (PLS)*. The basic idea of PLS approach is to iterate through a group selection phase (to generate a starting solution S according to a *probability matrix* P that indicates for each item its chance to belong to each group), an optimization phase (to obtain an improved solution S' from S), and a probability learning phase. During the probability learning phase, by comparing the starting solution S and the improved solution S', we try to know whether each item moved from its original group to a new group in S' or stayed in its original group of S. The viability of the proposed PLS approach is verified on a well-known representative grouping problem, i.e., graph coloring, and the corresponding algorithm is denoted as PLSCOL. Compared with the general PLS approach, the PLSCOL algorithm introduces two improvements. Considering the specific feature of GCP where color groups are interchangeable, we introduce the group matching procedure to find a group-to-group correspondence between a starting solution and its improved solution. Also, instead of using the descent-based local search to search for a legal coloring, we adopt a more elaborated coloring algorithm (i.e., the well-known tabu search procedure). Experimental studies on popular DIMACS benchmark graphs indicate that PLSCOL achieves competitive performances compared to a number of well-known coloring algorithms. This work has published in *Expert Systems and Applications* [Zhou *et al.*, 2016]. A further work of an improved probability learning based local search for graph coloring [Zhou *et al.*, 2017a] is submitted to *Applied Soft Computing* in April 2016 and was revised in September 2017.

— For the subset selection problems such as MDP, we present an Opposition-Based Memetic Algorithm (OBMA, published in *IEEE Transactions on Evolutionary Computation* [Zhou *et al.*, 2017c]), which integrates the concept of Opposition-Based Learning (OBL) into the well-known memetic search framework. OBMA explores both candidate solutions and their opposite solutions during its initialization and evolution processes. Combined with a powerful local optimization procedure and a rank-based quality-and-distance pool updating strategy, OBMA establishes a suitable balance between exploration and exploitation of its search process. Computational results on 80 popular MDP benchmark instances show that the proposed algorithm matches the best-known solutions for most of instances, and finds improved best solutions (new lower bounds) for 22 instances. We provide experimental evidences to highlight the beneficial effect of opposition-based learning for solving MDP.

— For the permutation problems such as QAP, we develop a hybrid approach called Frequent Pattern Based Search (FPBS) [Zhou *et al.*, 2017d] that combines frequent pattern mining and optimization. The proposed method uses a data mining procedure to mine frequent patterns from a set of high-quality solutions collected from previous search, and the mined frequent patterns are then employed

to build starting solutions that are improved by an optimization procedure. After presenting the general approach and its composing ingredients, we illustrate its application to solve the well-known and challenging quadratic assignment problem. Computational results on the 21 hardest benchmark instances show that the proposed FPBS approach competes favorably with state-of-the-art algorithms both in terms of solution quality and computing time.

We would like to mention that during the preparation of this thesis, we also investigated the well-known minimum different dispersion problem and the critical node problems. An effective iterated local search has been proposed for solving the minimum different dispersion problem, and this work has been published in *Knowledge-Based Systems* [Zhou and Hao, 2017b]. While for the critical node problems, preliminary experimental results are reported in *GECOO 2017* as a conference article [Zhou and Hao, 2017a], and improved studies are summarized in a submitted but still unpublished article [Zhou *et al.*, 2017e] which is available in *arXiv*. Since these two pieces of works are not within the family of "learning driven heuristic optimization", they will not be detailed in this thesis.

Organization

The organization of the whole thesis is displayed in Figure 1. Specifically, this thesis is organized as follows:

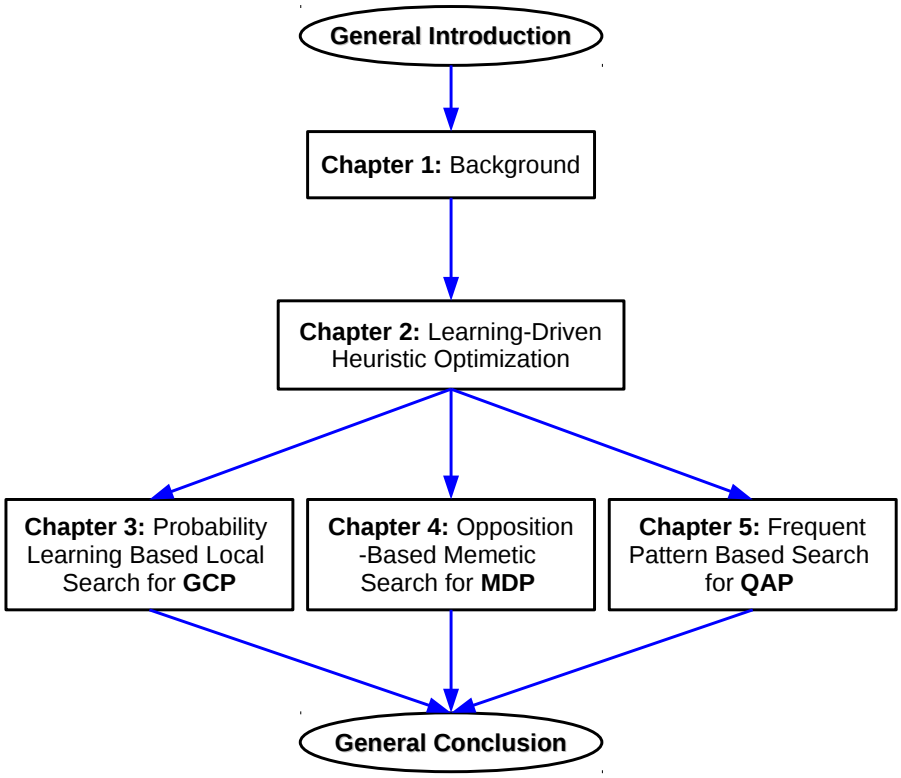

Figure 1: The structure of the whole thesis.

— **Chapter 1** starts with the discussion of combinatorial optimization problems (COPs) and then formally introduces three well-known examples considered in this thesis, i.e., graph coloring problem, maximum diversity problem, and quadratic assignment problem. Finally, we discuss the general solution approaches for solving COPs, including metaheuristics and hybrid metaheuristics.

— **Chapter 2** introduces the concept of learning-driven heuristic optimization. Then, we give a short introduction of machine learning and data mining. It is followed by a summary of related surveys

on using machine learning to help heuristic algorithms. Finally, we conduct a brief literature review of learning-driven heuristic optimization approaches by organizing them according to four different objectives of using machine learning techniques.

— **Chapter 3** applies the probability learning technique to solve grouping problems, specifically for the graph coloring problem. We present a Probability learning based Local Search (PLS) approach for grouping problems. We first describe in detail the main components of the proposed PLS including group selection procedure, optimization procedure, probability updating and probability smoothing procedure. Then, a case study of PLS is made on the well-known graph coloring problem (denoted as PLSCOL), which is a typical grouping problem. The viability of the proposed PLSCOL algorithm is verified by comparing our computational results with state-of-the-art results. Finally, we investigate the key ingredients of the proposed approach to gain a deep understanding of their impacts on the performance of the algorithm.

— **Chapter 4** integrates the concept of the opposition-based learning technique into the well-known memetic algorithm framework for solving the maximum diversity problem. We first give a brief introduction of opposition-based learning and memetic search, and then present our proposed Opposition-Based Memetic Algorithm (OBMA). After a detailed description of each component of the proposed OBMA algorithm, we present experimental results and comparisons with the state-of-the-art algorithms in the literature to show the benefit of the opposition-based learning and discuss the effectiveness of some key components.

— **Chapter 5** combines the frequent pattern mining technique with heuristic algorithm for solving the quadratic assignment problem. We first provide some background about mining and heuristic. Then, we present a detailed description of the proposed Frequent Pattern-Based Search (FPBS) which aims to use a data mining procedure to mine frequent patterns from a set of high-quality solutions collected from previous search, and the mined frequent patterns are then employed to build starting solutions that are improved by an optimization procedure. Computational studies of the proposed FPBS are conducted on 21 hardest benchmark instances. Finally, some experimental analysis are made to investigate the effectiveness of some key algorithm components.

1

Background

This introductory chapter provides the background for designing heuristic algorithms for Combinatorial Optimization Problems (COPs). We start with a discussion of combinatorial optimization and introduce three well-known COPs as studied examples in the whole thesis, including the graph coloring problem, maximum diversity problem and quadratic assignment problem. Following this, we give a brief overview of metaheuristics and hybrid metaheuristics in the literature.

Contents

1.1 Combinatorial optimization

1.1.1 Basic notations and definitions

Combinatorial problems arise in many areas of computer science and other disciplines in which computational methods are applied, such as artificial intelligence, operations research, and bioinformatics. Combinatorial problems typically involve finding groupings, orderings, or assignments of a discrete set of objects which satisfy certain constraints and optimize an objective function. These elements are generally modelled by means of a combinatorial structure and represented through a vector of decision variables which can assume values within a finite set. The potential solution of a combinatorial problem is usually a value assignment to the variables that meets some specified constraints. Combinatorial optimization is an important branch of optimization. Its domain is optimization problems where the set of feasible solutions is discrete or can be reduced to a discrete one, and the goal is to find the best possible solution.

We now give a formal definition of *Combinatorial Optimization Problems (COPs)*. In general, a COP can be either a *minimization* or a *maximization* problem. Without loss of generality, we consider a minimization problem. However, the adaptation for handling maximization problems is straightforward.

Definition 1. *[Blum and Roli, 2003] A COP instance $I = (\Omega, f)$ is defined by a set of variables $X = \{x_1, \ldots, x_n\}$ and corresponding domains $D_1, \ldots, D_n$, a set of constraints among variables, and an objective function f to be minimized, where $f : D_1 \times \ldots \times D_n \rightarrow \mathbb{R}^+$.*

A feasible solution of instance I is $S = \{(x_1, v_1), \ldots, (x_n, v_n)\} | v_i \in D_i, S$ *satisfies all the constraints*$\}$, and the set of all feasible solutions is denoted as Ω. Ω is usually called a *search space* and each element $S \in \Omega$ is a *candidate solution* of I. To solve the optimization problem associated with instance I, one has to find a solution $S^* \in \Omega$ with minimum objective function value, i.e., $f(S^*) \leqslant f(S), \forall S \in \Omega$. S^* is called a *global optimum* of I.

1.1.2 Studied problems

This thesis considers three important categories of combinatorial optimization problems:

— **Grouping problems** aim to partition a set of objects into a collection of mutually disjoint subsets according to some specific criterion and constraints. Some representative examples include the *graph coloring problem*, capacitated clustering problem, maximally diverse grouping problem and bin packing problem.

— **Subset selection problems** try to find a subset of a given set so that a given set of constraints is satisfied. Many combinatorial search problems can be naturally solved as subset selection problems, such as the critical node problems, winner determination problem, *maximum diversity problem* and other diversity and dispersion problems.

— **Permutation problems** represent a category of COPs whose solutions are based on permutations, the true meaning of these permutations can be different in different cases. Well-known permutation problems include, for instance, the traveling salesman problem, flow shop scheduling problem, linear ordering problem and *quadratic assignment problem*.

In the following, we introduce three example problems which will be used throughout this thesis to illustrate specific issues concerning the solution of COPs using learning-based heuristic approaches. These problems are classic $\mathcal{NP}$-hard problems, including the graph coloring problem, maximum diversity problem and quadratic assignment problem. They fall into three representative categories of COPs: grouping problems, subset selection problems and permutation problems, respectively. We begin our introduction with the graph coloring problem.

Example of a grouping problem: Graph coloring problem

The *Graph Coloring Problem (GCP)* is one of the most studied COPs [Garey and Johnson, 1979]. Given an undirected graph $G = (V, E)$, where V is the set of $|V| = n$ vertices and E is the set of $|E| = m$ edges, a *k-coloring* of G is a partition of V into k mutually disjoint color classes such that two vertices linked by an edge must belong to two different color classes. The decision version of the GCP attempts to answer whether for a given graph G and an integer k, a k-coloring exists (k-GCP for short), while the optimization version of the GCP tries to find the minimum number k such that a k-coloring exists. This minimum value of k required for a legal coloring is called the *chromatic number* $\chi(G)$ of G.

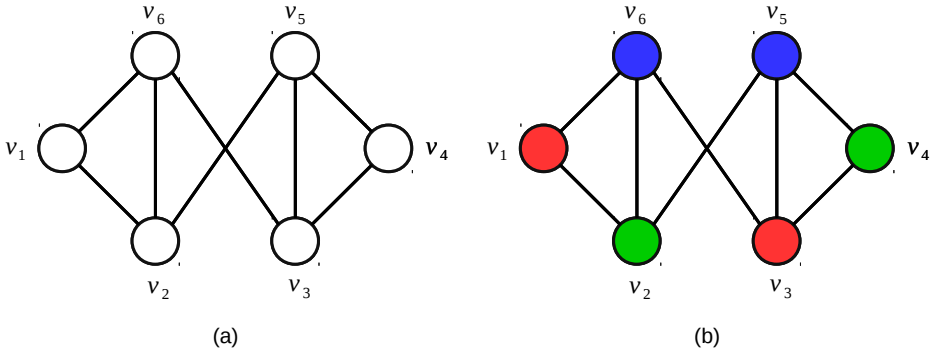

Figure 1.1: A GCP example. (a) An undirected graph G with six vertices. (b) The 3-coloring for the G.

The leftmost of Figure 1.1 shows an undirected graph G with six vertices $v_1, \ldots, v_6$. A graph coloring configuration is to assign each vertex one color, and a legal 3-coloring is shown in the rightmost of Figure 1.1. In this example, vertices v_1 and v_3 are colored by red color, v_2 and v_4 are colored by green color, and v_5 and v_6 are colored by blue color. This 3-coloring is the optimal coloring of G because the chromatic number for this graph is also 3. We can clearly observe that there are two cliques (i.e., a fully connected sub-graph) of three vertices, i.e., clique $\{v_1, v_2, v_6\}$ and clique $\{v_3, v_4, v_5\}$. It means that 3 is a lower bound of of G because any valid coloring of a clique of size k requires at least k different colors.

Note that a solution of a given GCP instance usually does not depend on the the particular numbering of the colors, because all permutations of a legal k-coloring are isomorphic solutions. Alternatively to the formulation as an assignment problem, the GCP can be represented as a grouping problem, in which a k-coloring corresponds to a grouping of the set of vertices into k groups such that no two adjacent vertices u and v belong to the same group. For example, the 3-coloring shown in the rightmost of Figure 1.1 can also be represented as a grouping, i.e., $\{v_1, v_3\}$, $\{v_2, v_4\}$ and $\{v_5, v_6\}$

In general, the GCP can be approximated by solving a series of k-GCP (with decreasing k) as follows [Galinier *et al.*, 2013]. For a given G and a given k, graph coloring algorithms usually try to solve k-GCP by seeking a legal k-coloring. If such a coloring is successfully found, k decreases to $k - 1$ and solves the new k-GCP again. The process repeats until no legal k-coloring can be reached. In this case, the smallest k for which a legal k-coloring has been found represents an approximation (upper bound) of the chromatic number of G. This general solution approach has been used in many coloring algorithms, and is also adopted in our work.

k-GCP is a very popular $\mathcal{NP}$-complete COP in graph theory [Garey and Johnson, 1979] and has attracted much attention in the literature. GCP arises naturally in a wide variety of real-world applications, such as register allocation [Chaitin, 1982], timetabling [Burke *et al.*, 1994; de Werra, 1985], frequency assignment [Gamst, 1986; Hale, 1980; Hao *et al.*, 1998], and scheduling [Leighton, 1979; Zufferey *et al.*, 2008]. It was one of the three target problems of several international competitions including the well-known Second DIMACS Implementation Challenge on Maximum Clique, Graph Coloring, and Satisfiability.

Example of a subset selection problem: Maximum diversity problem

Given a set N of n elements where any pair of elements are separated by a distance, and an integer number m with $m < n$, the *Maximum Diversity Problem (MDP)* aims to select a subset S of m elements from N in such a way that the sum of pairwise distances between any two elements in S is maximized. Let $N = \{e_1, e_2, \ldots, e_n\}$ be the given set of elements and d_{ij} be the distance between element e_i and element e_j ($d_{ij} = d_{ji}$). Formally, MDP can be formulated as the following quadratic binary problem [Kuo *et al.*, 1993]:

$$\max \quad f(x) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_i x_j \tag{1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_i = m \tag{1.2}$$

$$x_i \in \{0,1\}; \quad i = 1, \ldots, n \tag{1.3}$$

where the binary variable $x_i = 1$ if element e_i is selected; and $x_i = 0$ otherwise. Equation (1.2) ensures that a solution S exactly contains m elements.

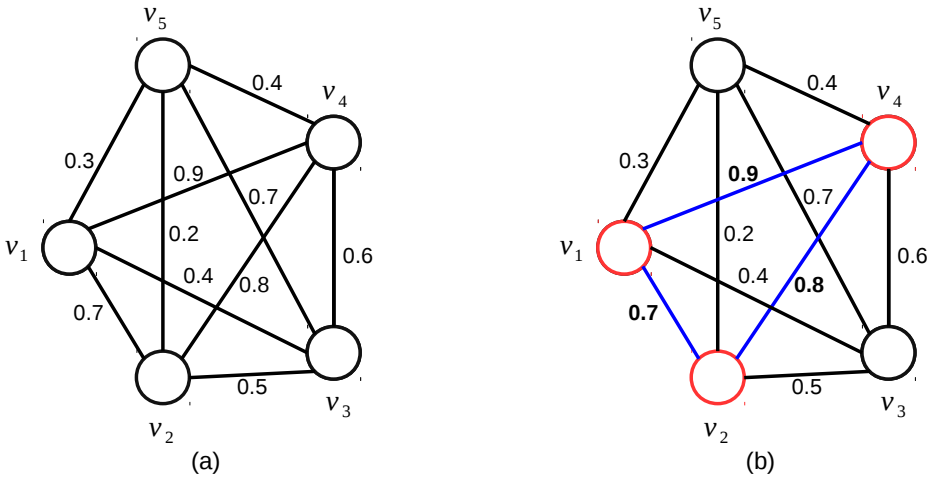

Figure 1.2: A MDP example. (a) A MDP instance of size 5 and $m = 3$. (b) An optimal solution for the instance (i.e., a subset $S = \{v_1, v_2, v_4\}$) and its cost can be computed as $d_{12} + d_{14} + d_{24} = 0.7 + 0.9 + 0.8 = 2.4$.

Figure 1.2 shows a MDP example. There are in total five elements $N = \{v_1, v_2, v_3, v_4, v_5\}$, and each pair of elements are separated by a distance, as shown in the leftmost of Figure 1.2. The rightmost of Figure 1.2 shows a selected subset $S = \{v_1, v_2, v_4\}$, and we color all selected elements with red color. This subset is also an optimal solution of maximum cost 2.4.

MDP is known to be $\mathcal{NP}$-hard and has a high computational complexity [Ghosh, 1996]. Besides the theoretical significance as a difficult combinatorial problem, MDP also proves to be useful model to formulate a variety of practical applications including location of undesirable or mutually completing facilities [Erkut and Neuman, 1991], genetic engineering [Martí *et al.*, 2013], decision analysis with multiple objectives [Palubeckis, 2007], composing jury panels [Lozano *et al.*, 2011], product design [Glover *et al.*, 1998], medical and social sciences [Kuo *et al.*, 1993]. More details on the applications of MDP can be found in [Martí *et al.*, 2013].

Example of a permutation problem: Quadratic assignment problem

The *Quadratic Assignment Problem (QAP)* is a well-known $\mathcal{NP}$-hard COP. It has been subject of a great number of research efforts. QAP was originally introduced by Koopmans and Beckman [Koopmans and Beckmann, 1957] in 1957 to model the locations of indivisible economic activities such as capital equipment. QAP aims to determine a minimal cost assignment of n facilities to n locations, given a flow a_{ij} from facility i to facility j for all $i, j \in \{1, \ldots, n\}$ and a distance b_{uv} between locations u and v for all $u, v \in \{1, \ldots, n\}$. Let Ω denotes the set of all possible permutations $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$, then QAP can mathematically be formulated as follows.

$$\min_{\pi \in \Omega} f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} \tag{1.4}$$

where a and b are the flow and distance matrices respectively, and $\pi \in \Omega$ is a solution and $\pi(i)$ represents the location chosen for facility i. The optimization objective is to find a permutation π^* in Ω such that the sum of the products of the flow and distance matrices is minimized, i.e., $f(\pi^*) \leqslant f(\pi), \forall \pi \in \Omega$.

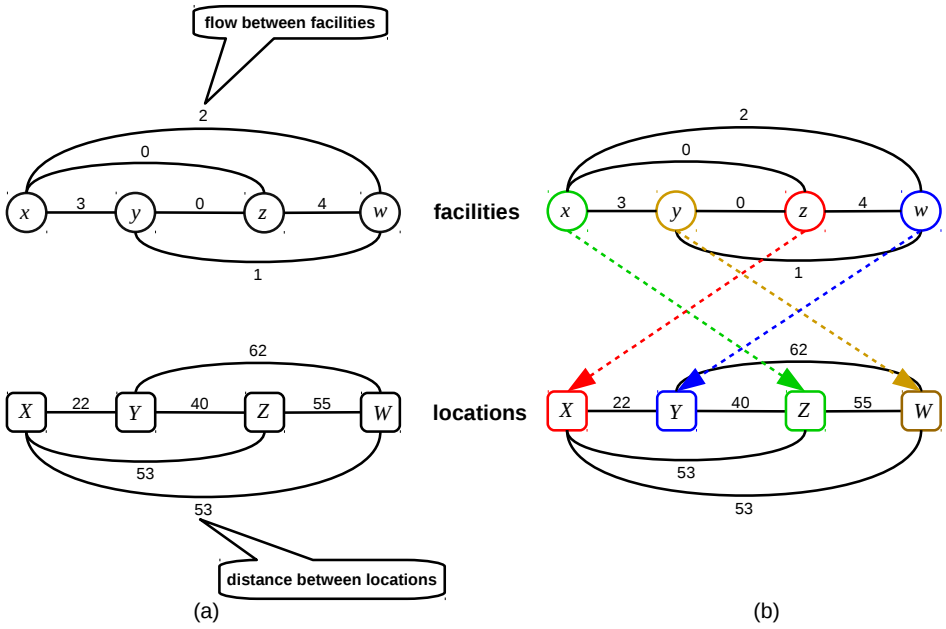

Figure 1.3: A QAP example. (a) A QAP instance of size 4. (b) An optimal assignment for the instance $(x \rightarrow Z, y \rightarrow W, z \rightarrow X, w \rightarrow Y)$ and its assignment cost can be computed as $a_{12} \cdot b_{34} + a_{13} \cdot b_{31} + a_{14} \cdot b_{32} + a_{23} \cdot b_{41} + a_{24} \cdot b_{42} + a_{34} \cdot b_{12} = 3 \cdot 55 + 0 \cdot 53 + 2 \cdot 40 + 0 \cdot 53 + 1 \cdot 62 + 4 \cdot 22 = 395$.

As a typical example of QAP, consider a facility location problem given in Figure 1.3. There are four available locations $\{X, Y, Z, W\}$ and four facilities $\{x, y, z, w\}$ to locate, the line between two facilities indicates that there is a required flow, while the line between two locations means the distance between them, as shown in leftmost of Figure 1.3. The objective of the problem is to assign four facilities to four locations in such a way as to minimize the total sum of the pair assignment costs. The assignment cost for a pair of facilities is the product of the flow between the facilities and the distance between the locations of the facilities. One possible assignment is shown in the rightmost of Figure 1.3: facility z is assigned to location X, facility w is assigned to location Y, facility x is assigned to location Z, and facility y is assigned to location W. This assignment can be written as a permutation $\{z, w, x, y\}$. In fact, the assignment shown in the rightmost of Figure 1.3 is an optimal assignment.

Besides the facility location problem, QAP can model a number of other real-world problems [Drezner *et al.*, 2005; Loiola *et al.*, 2007; Duman and Or, 2007] such as electrical circuit wiring/routing, transportation engineering, parallel and distributed computing, image processing and analysis of chemical reactions for

organic compounds. Reviews on some significant applications of QAP can be found in [Pardalos *et al.*, 1994; Duman and Or, 2007]. In addition, many classic $\mathcal{NP}$-hard COPs, such as traveling salesman problem, maximum clique problem, bin packing problem and graph partitioning problems, can also be formulated as QAPs [Loiola *et al.*, 2007].

1.2 Solution approaches for COPs

It is well-known that the great majority of complex real-world problems, when modeled as optimization problem, belongs to the category of $\mathcal{NP}$-hard problems. These $\mathcal{NP}$-hard problems arise in many areas of industrial applications, such as telecommunication, computational biology, transportation and logistics, engineering design, scheduling and planning.

Due to the practical importance of COPs, many algorithms have been proposed in the literature to tackle them. These algorithms can be roughly classified as either *exact* or *approximate* methods (see Figure 1.4). Exact algorithms are guaranteed to find the optimal solution of a COP. A survey on exact algorithms for $\mathcal{NP}$-hard problems is provided in [Woeginger, 2003]. Unfortunately, most of combinatorial search problems are $\mathcal{NP}$-hard, and exact algorithms need an exponential worst-case run time to explore the search space. Moreover, in many practical applications, it is often enough to obtain a high-quality (or sub-optimal) solution. Therefore, researchers often prefer to use approximate algorithms, especially *heuristic algorithms*. Heuristic algorithms usually sacrifice the guarantee of finding optimal solutions for the sake of getting high-quality solutions in polynomial-time.

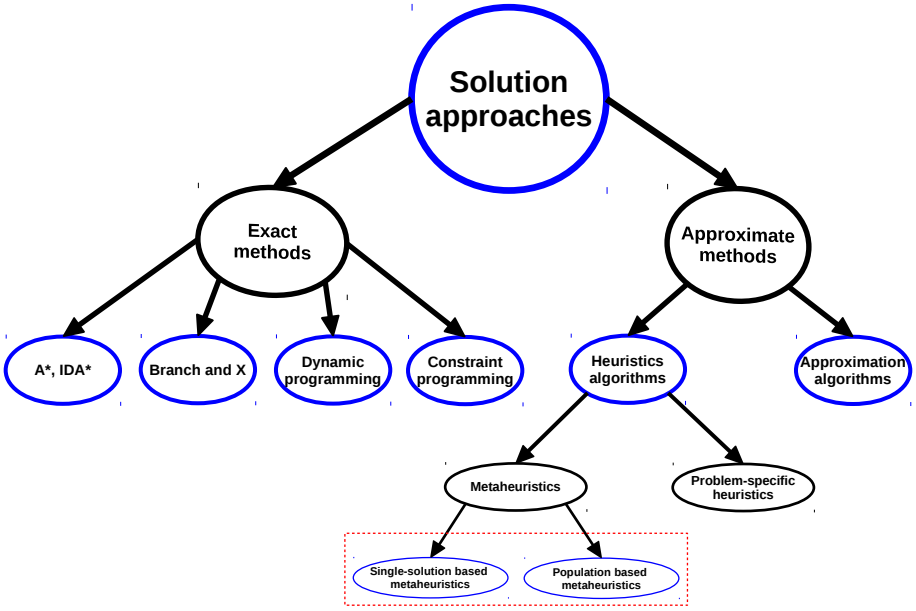

Figure 1.4: A brief taxonomy of solution approaches for COPs.

1.2.1 Metaheuristics

As opposed to exact algorithms, which guarantee to find an optimal solution of the problem, heuristic algorithms only attempt to yield a good, but not necessarily optimal solution. The field of heuristic optimization is a rapidly growing research area. A great number of heuristic algorithms have been proposed in the literature [Blum and Roli, 2003; Blum *et al.*, 2011; Sörensen and Glover, 2013]. Many heuristic algorithms are specific and problem-dependent. In contrast to problem-specific heuristics, *metaheuristics* are problem-independent algorithms [Glover, 1986]. Metaheuristics may be considered as upper level general methodologies that can be used as a guiding strategy in designing underlying heuristics to solve specific

optimization problems. In the literature, a clear definition of metaheuristic is still lacking, or it could be argued that it is still disputed. In our case, we adopt the following definition:

> "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method. The family of metaheuristics includes, but is not limited to, adaptive memory procedures, tabu search, ant systems, greedy randomized adaptive search, variable neighborhood search, neural networks, simulated annealing, and their hybrids." [Voß *et al.*, 2012].

A large number of metaheuristics have been proposed in the literature [Blum and Roli, 2003; Blum *et al.*, 2011; Sörensen and Glover, 2013]. They can be categorized according to different classification criteria [Blum and Roli, 2003], such as nature inspired versus non-nature inspired, memory usage versus memory-less, dynamic versus static objective function, one versus various neighborhood structures, and single-solution based search and population-based search. In the following, we focus on conducting a brief overview of metaheuristics by organizing them into two categories: *single-solution based metaheuristics* and *population-based metaheuristics*.

Single-solution based metaheuristics

The main characteristic of single-solution based metaheuristics is that a single solution (and search trajectory) is considered at a time. *Local Search (LS)* approaches fall into the category of single-solution based metaheuristics. Local search metaheuristics usually start from an initial solution and try to improve the current solution by iteratively performing local changes (called moves) [Hoos and Stützle, 2004]. The set of solutions that can be obtained by applying a single move to a given solution is called the neighborhood of that candidate solution. At each iteration, a potential solution from the neighborhood of the current solution is selected to work as the new candidate solution. The simplest local search algorithm is the greedy *Hill-Climbing (HC)*. HC accepts only neighboring solutions with the same or better cost than the current solution. This method can quickly find a local optimum, but the obtained local optimum is often of poor quality. A solution whose neighborhood does not contain any better solutions is called a local optimum (as opposed to a global optimum). When the search walks into a local optimum, the heuristic needs a perturbation strategy to escape to other regions of the search space.

The simplest strategy to escape to other regions is either to start the search again from a new starting solution or to make a relatively large change (called perturbation) to the current solution. These two strategies respectively result in two metaheuristics known as *Multi-start Local Search (MLS)* [Martí, 2003] and *Iterated Local Search (ILS)* [Lourenço *et al.*, 2003]. Recently, two improved ILS approaches are proposed, i.e., *Breakout Local Search (BLS)* [Benlic and Hao, 2012; Benlic and Hao, 2013; Benlic *et al.*, 2017] and *Three-Phase Search (TPS)* [Lai and Hao, 2016; Zhou and Hao, 2017b]. BLS distinguishes itself from the conventional ILS approach by the following two aspects. First, multiple types of perturbations are used in BLS, which are triggered according to the search states, achieving variable levels of diversification. Second, the local optimal solution returned by the local search procedure is always accepted as the new starting solution in BLS regardless of its quality, which completely eliminates the acceptance criterion component of ILS [Benlic and Hao, 2012]. TPS follows and generalizes the basic ILS scheme. It iterates through three distinctive and sequential search phases. Starting from an initial solution, a descent-based neighborhood search procedure is first employed to find a local optimal solution. Then, a local optima exploring phase is triggered with the purpose of discovering nearby local optima of better quality. When the search stagnates in the current search zone, TPS turns into a diversified perturbation phase, which strongly modifies the current solution to jump into a new search region. The process iteratively runs the above three phases until a given stopping condition is met. Compared to BLS, TPS further divides the perturbation phase into a local optima exploring phase (to discover more local optima within a given region) and a diversified perturbation

phase (to displace the search to a new and distant search region). Both BLS and TPS have been successfully applied to solve several hard COPs. A detailed comparison and discussion among algorithms ILS, BLS and TPS is provided in our paper [Zhou and Hao, 2017b].

Changing the type of neighborhood is also a widely-used strategy to escape from the local optimum. Metaheuristics that use this strategy are commonly called *Variable Neighborhood Search (VNS)* [Mladenović and Hansen, 1997]. VNS systematically exploits the idea of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. The rationale behinds this strategy is that a local optimum relative to a specific neighborhood can often be improved by performing local search with a different neighborhood. The success of VNS depends on three facts: (i). a local optimum with respect to one neighborhood structure is not necessarily a local optimum for another one; (ii). a global optimum is a local optimum with respect to all possible neighborhood structures; (iii). for many problems local optima with respect to one or several neighborhoods are relatively close to each other. Besides the basic VNS, many extensions have also been proposed in the literature, such as *Skewed VNS (SVNS)* and *Variable Neighborhood Decomposition Search (VNDS)* [Hansen and Mladenović, 2005].

Besides two kinds of strategies mentioned above, some algorithms use memory structures to guide the search to find good solutions more quickly. Different memory structures may be used to explicitly remember different aspects about the trajectory that the algorithm has previously experienced through the search and different strategies may be devised to use this information to drive the search to promising search areas [Glover and Laguna, 1993]. A well-known example is *Tabu Search (TS)* [Glover, 1990; Glover and Laguna, 1993; Glover and Laguna, 1997]. Widely-used memory structures include the tabu list that records the last encountered solutions (or some attributes of them) and forbids these solutions (or attributes) from being visited again as long as they are on the list. The length of the tabu list is called the *tabu tenure*. Frequency memory records how often certain attributes have been encountered in solutions on the search trajectory, which allows the search to avoid visiting solutions that display the most often encountered attributes or to visit solutions with attributes seldom encountered. Such memory can also include an evaluative component that allows moves to be influenced by the quality of solutions previously encountered. Other memory structures such as an elite set of the best solutions encountered so far are also common [Sörensen and Glover, 2013]. Another example of the use of memory can be found in a metaheuristic called *Guided Local Search (GLS)* [Voudouris and Tsang, 1999]. GLS introduces an augmented objective function that includes a penalty factor for each potential element. When trapped in a local optimum, GLS increases the penalty factor for all elements of the current solution, making other elements (and therefore other moves) more attractive and allowing the search to escape from the local optimum [Voudouris and Tsang, 1999].

Unlike most other local search approaches, *Simulated Annealing (SA)* uses a random move strategy, emulating the annealing process of a crystalline solid [Kirkpatrick *et al.*, 1983]. The fundamental idea of SA is to probabilistically allow moves resulting in solutions of worse quality than the current solution in order to escape from local optima. The probability to perform such a move is continuously decreased during the search. SA is commonly recognized as the oldest among the metaheuristics and surely one of the first algorithms that had an explicit strategy to escape from local optima. In addition to simulated annealing, a number of other similar search methods have also been proposed in the literature, such as *Threshold Accepting (TA)* [Dueck and Scheuer, 1990] and *Great Deluge Algorithm (GDA)* [Dueck, 1993].

Greedy Random Adaptive Search Procedure (GRASP) [Feo and Resende, 1995] is a single-solution based metaheuristic, which combines constructive heuristics and local search. GRASP uses randomization to overcome the drawback of purely greedy algorithms by introducing some randomness to the selection process. Specifically, GRASP is composed of two phases: solution construction and solution improvement. The construction phase builds a feasible solution, whose neighborhood is investigated until a local optima is found during the local search phase. These two phases are repeated until a termination condition is met.

Population-based metaheuristics

Unlike single-solution based metaheuristics, at each generation, population-based metaheuristics handle a set of solutions rather than a single solution. The idea behind this category of metaheuristics is that good solutions can be found by exchanging solution features between two or more (usually high-quality) solutions. The most widely-studied population based methods in combinatorial optimization are evolutionary computation methods and swarm intelligence methods.

Genetic Algorithm (GA) was originally proposed in [Holland, 1975] and was inspired by Darwin's theory about evolution [Davis, 1991]. The driving force behind most GAs is selection and recombination (or crossover). Selection guarantees that, in most cases, high-quality solutions in the population are selected for recombination, usually by biasing the probability of each solution to be selected towards its objective function value. Recombination aims to use specialized operators to recombine the features of two or more solutions into new offspring solutions. The new offspring solutions are then inserted into the population or discarded according to a specific criterion. In many cases, mutation operator is also applied which causes local changes of offspring solution produced by recombination operator. Starting from an initial population of individuals (generated randomly or with a constructive algorithm), GAs try to improve the quality of the individuals by making the population evolve. Specifically, GAs iterate the selection, recombination, mutation, and reinsertion phases until a stopping condition is met, and return the best solution ever met in the population [Calégari *et al.*, 1999].

Unlike GA, *Path Relinking (PR)* [Glover *et al.*, 2004] operates on a small set of solutions and employs diversification strategies of the form proposed in tabu search, which gives precedence to strategies learning based on adaptive memory, with limited recourse to randomization. PR tries to create combinations of solutions by a process of generating paths between selected high-quality solutions. The approach was originally suggested in [Glover, 1989] and then formalized and named as path relinking in [Glover and Laguna, 1993]. Paths consist of elementary moves such as the ones used in local search metaheuristics. The moves on a path transform one solution (called the initiating solution) to a second solution (called the guiding solution) in the search space. Contrary to local search approaches, PR employs a move strategy that chooses the move to execute based on the fact that this move will bring the solution closer to the guiding solution. Besides the classic between-form of PR, a beyond-form of PR (called exterior path relinking) is also proposed in recent years [Glover, 2014]. The exterior path relinking shares a similar idea with the opposition-based memetic search proposed in [Zhou *et al.*, 2017c].

Ant Colony Optimization (ACO) [Dorigo *et al.*, 1996] is another category of population-based metaheuristics. ACO was introduced in the early 1990s as a novel technique for solving hard COPs [Dorigo *et al.*, 1996; Dorigo and Blum, 2005]. The inspiring source of ACO is the foraging behavior of real ants. The central component of an ACO algorithm is a parametrized probabilistic model, which is called pheromone model. Artificial ants incrementally construct solutions by adding opportunely defined solution components to a partial solution under consideration. In general, the ACO algorithm attempts to solve an optimization problem by repeating the following two steps: (i) candidate solutions are constructed using a pheromone model, that is, a parametrized probability distribution over the solution space; (ii) the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high-quality solutions [Dorigo and Blum, 2005].

1.2.2 Hybrid metaheuristics

Besides the aforementioned two categories of metaheuristics, hybrid metaheuristic is also very popular in combinatorial optimization [Blum *et al.*, 2011]. Hybrid metaheuristics are algorithms for combinatorial optimization that result from a combination of algorithmic components originating from different optimization methods, such as metaheuristics, mathematical programming, constraint programming and machine learning [Talbi, 2016]. The hybridizations in the literature can be roughly divided into the following four categories:

— Combining metaheuristics with complementary metaheuristics;

— Combining metaheuristics with constraint programming;

— Combining metaheuristics with mathematical programming;

— Combining metaheuristics with machine learning [1].

Each kind of combination represents a category of hybrid metaheuristics. These hybrid optimization algorithms have been applied to find the best solutions of many COPs in science and industry. In the following, we will discuss these categories of hybrid metaheuristics, respectively.

Combining metaheuristics with complementary metaheuristics

Single-solution based metaheuristics (e.g., SA and TS) manipulate and transform a single solution during the search, whereas population-based metaheuristics operate on a set of candidate solutions in a population. These two kinds of metaheuristics have complementary characteristics. i.e., the former is good at intensifying the search in local regions, while the later allows a better diversification in the whole search space. The combination of the single-solution based metaheuristic and the population-based metaheuristic makes use of advantages of each metaheuristic and results in a new hybrid metaheuristic that is more powerful than any of its composing metaheuristic. That is, the resulting hybrid metaheuristic could offer a good balance between exploitation and exploration, thus assuring a high search performance.

Memetic Algorithm (MA) [Moscato and others, 1989] is a representative example, and it is widely recognized as the most successful class of hybrid metaheuristics. MA combines the population-based metaheuristic with local search metaheuristic. Indeed, it is generally believed that the population-based search framework offers more facilities for exploration while neighborhood search provides more capabilities for exploitation [Hao, 2012]. The effectiveness of MAs have been verified on many classic $\mathcal{NP}$-hard problems, such as graph coloring, graph partitioning and quadratic knapsack problem [Hao, 2012]. We also apply memetic algorithm framework in our work. For example, we propose an opposition-based memetic search for the maximum diversity problem [Zhou *et al.*, 2017c], and a memetic search algorithm for identifying critical nodes in sparse graphs [Zhou *et al.*, 2017e].

In contrast to the use of local search algorithms within population-based algorithms (e.g., MA), recent years have also witnessed the development of some algorithms that result from the enhancement of metaheuristics based on local search with concept from population-based approaches. An example is the Population-based Iterated Local Search (PILS) algorithm introduced in [Thierens, 2004]. PILS is a population extension of the ILS metaheuristic. The goal of the PILS is to improve a single solution by running an ILS algorithm. In addition to ILS, PILS also keeps a small population of neighboring solutions. These solutions are employed to focus the ILS process to the common search region between the current solution and of the individual in the population.

PR is a major enhancement to the basic GRASP procedure, leading to significant improvements both in terms of solution quality and time. The use of PR within a GRASP algorithm (named GRASP-PR), as an intensification strategy applied to each local optimum, was first proposed in [Laguna and Marti, 1999]. Some extensions of the GRASP-PR have been also proposed in recent years. The hybridization of PR and GRASP has successfully applied to many applications, such as 2-path network design problem, p-median problem, the Steiner problem, the job-shop scheduling problem, quadratic assignment problem and max-cut problem. A detailed review of applications and advances of GRASP-PR is provided in [Resendel and Ribeiro, 2005].

Combining metaheuristics with constraint programming

Metaheuristics and *Constraint Programming (CP)* are two different problem solving techniques. CP is based on feasibility (finding a feasible solution) rather than optimization (finding an optimal solution) and

1. All the three hybrid metaheuristics proposed in this thesis belong to this category

focuses on the constraints and variables instead on the objective function, while the metaheuristics are good at optimizing objective functions instead of finding feasible solutions. Since they are two complementary techniques, it is natural to design a hybrid metaheuristic which makes use of the power of strong constraint propagation techniques and the explorative power and performance of metaheuristic.

The hybridizations between metaheuristics and CP techniques have been applied to successfully solve many optimization problems [Focacci *et al.*, 2004]. On the one hand, CP can be integrated into metaheuristic as a sub-procedure. For example, CP-based Large Neighborhood Search (LNS) which represents a class of problem solving techniques in which local search uses CP for exploring a very large neighborhood. LNS aims to combine the advantage of a large neighborhood, that usually enhances the explorative capabilities of local search, with an exhaustive CP exploration that is faster than enumeration, especially when most problem variables are already assigned [Shaw, 1998]. On the other hand, metaheuristic can also be used in CP algorithms. For instance, ACO-driven CP is a CP solver driven by a learning branching heuristic [Di Gaspero *et al.*, 2013]. The literature on hybrid methods which combine metaheuristics with CP is quite rich ranging from theory of algorithms to applications. Detailed surveys of hybrid algorithms combing metaheuristics and CP is provided in [Focacci *et al.*, 2004; Di Gaspero, 2015].

Combining metaheuristics with mathematical programming

The hybridization of metaheuristics and *Mathematical Programming (MP)* is known as *matheuristic* [Maniezzo *et al.*, 2009]. Metaheuristics and MP are also two complementary solution techniques. MP techniques are known to be time and memory consuming, and they are not applicable to large instances of difficult optimization problems. For these large instances, researchers often prefer heuristic and metaheuristic algorithms that are able to efficiently find a good or sub-optimal solution. The hybridization between metaheuristic and MP is able to make use of their individual benefits. On the one hand, MP techniques can be applied to improve the effectiveness of heuristic algorithms (i.e., finding better solutions). On the other hand, metaheuristics can also be employed to design more efficient exact algorithms by finding optimal solutions in shorter computational time.

The main MP techniques can be roughly classified as follows: (i) enumerative methods, such as Branch and Bound (B&B) and Dynamic Programming (DP); (ii) relaxation techniques, such as Lagrangian relaxation; (iii) decomposition methods, such as the Bender's decomposition; and (iv) cutting plan and pricing algorithms. The combinations of MP techniques and metaheuristics have applied to successfully solve many hard optimization problems. For example, Fu and Hao [Fu and Hao, 2015] proposed a highly effective dynamic programming driven memetic algorithm for the Steiner tree problem with revenues, budget and hop constraints. Based on DP technique, an estimation criterion is designed to identify and discard a large number of useless neighboring solutions, which significantly speeds up the local optimization procedure. Detailed surveys and classification on combining metaheuristics with mathematical programming can be found in [Puchinger and Raidl, 2005; Jourdan *et al.*, 2009; Maniezzo *et al.*, 2009; Talbi, 2016].

Combining metaheuristics with machine learning

Using *Machine Learning (ML)* techniques to help metaheuristics is becoming increasingly popular in combinatorial optimization. From **Chapter 3** to **Chapter 5**, we propose three hybrid metaheuristics based on different learning techniques. These three hybrid metaheuristics belong to this category. Since the hybridization between metaheuristics and machine learning techniques falls into the topic of *learning-driven heuristic optimization*, we will discuss it in more detail in the next chapter.

1.3 Chapter conclusion

In this chapter, we first introduced some basic concepts of combinatorial optimization problems (COPs). Then, we presented three representative $\mathcal{NP}$-hard problems, namely graph coloring problem, maximum diversity problem and quadratic assignment problem. These problems are case studies for verifying our proposed approaches in this thesis. Due to the practical importance of COPs, a great number of efforts have been made to design effective solution approaches. These algorithms can be roughly classified as exact algorithms and heuristic algorithms. For small instances, exact algorithms are usually applied to find optimal solutions in bounded computational time, while for large large instances, researchers prefer to solve them approximately by heuristic algorithms in polynomial-time. Many heuristic algorithms have been proposed in the literature. We summarized them as two categories: single-solution based metaheuristics and population-based metaheuristics, and discussed the main characteristics and some representative algorithms of each category. Besides these traditional metaheuristics, combinations of metaheuristics algorithms and other problem solving techniques have attracted increasing attention from the research community. We also conducted a brief overview of hybrid metaheuristics, such as combining metaheuristics with complementary metaheuristics, constraint programming, mathematical programming and machine learning techniques.

2

Learning-Driven Heuristic Optimization

This chapter serves mainly as an introduction into the topic of learning-driven heuristic optimization. We start with the background of learning-driven heuristic optimization. Then, we briefly overview the machine learning and data mining techniques. Related surveys on the learning-driven optimization are also discussed in detail. Finally, we systematically survey the literature on using machine learning to help heuristic algorithms, classifying them into four categories according to the purposes of applying machine learning techniques.

Contents

2.1 Introduction

The rapid increase of dimensions and complexity of real-world problems makes it more difficult to find optimal solutions by traditional optimization algorithms. To deal with the challenge, many intelligent and sophisticated heuristic algorithms have been proposed in the literature. One interesting direction is to analyze and enhance the heuristic algorithms by means of machine learning. Machine learning techniques, working from the data collected from the heuristic search process, can help to discover interesting relationships between the search space and the objective function, thus the following heuristic search may benefit from these uncovered relationships.

In the literature, machine learning techniques have been successfully applied to analyse and improve optimization algorithms. Learning lies at the core of artificial intelligence. Based on learning mechanism, many new concepts have been proposed in optimization, such as *learning from failures*, *conflict-directed clause learning*, *conflict-directed constraint learning*, *nogood learning*, *learning while optimization*, *learning from opposite* and *learning from problem solving*. Specifically for combinatorial optimization, related works have been studied under the umbrella of reactive search optimization, hyper-heuristics, automated algorithm configuration and algorithm selection.

— **Reactive Search Optimization (RSO)** [Battiti *et al.*, 2008] advocates the integration of machine learning techniques into search heuristics for solving complex optimization problems. The machine learning components act on the top of the search heuristic, thus the algorithm is able to self-tune its parameter values during the search process.

— **Hyper-heuristic** [Burke *et al.*, 2010] is a high-level heuristic search method that seeks to automate, often by incorporation of machine learning techniques, the process of selecting, combining, generating or adapting several low-level heuristics (or components of such heuristics) to efficiently solve computational search problems.

— **Algorithm Configuration (AC)** [Hutter *et al.*, 2009] aims to determine a well-performing parameter configuration of a given algorithm across a given set of instances. Algorithm configuration is also known as *parameter tuning*. It has been shown that the problem of tuning a metaheuristic can be described and solved as a machine learning problem [Birattari, 2009].

— **Algorithm Selection (AS)** [Kotthoff, 2014] is concerned with selecting the best algorithm to solve a given problem instance on a case-by-case basis. Almost all contemporary algorithm selection approaches employ machine learning to learn the performance mapping from problem instances to algorithms using features extracted from the instances. It is worth noting that the task of algorithm selection is also known as **meta-learning** [Lemke *et al.*, 2015] in the field of machine learning. The meta-learning focuses on selecting the base-learning method that is most likely to perform well for a specific problem.

In this chapter, we concentrate on investigating the literature which uses machine learning techniques to help heuristic search for COPs, especially for single-objective optimization problems. In order to present the literature review, we collected and selected the representative references which are closely related to this research field, by considering their quality, popularity, and the cover of different aspects of the topic surveyed.

The rest of the chapter is organized as follows. In the next section, we present a brief overview of machine learning and data mining. Section 2.3 summarizes the main surveys on learning-driven heuristic optimization in the literature. Section 2.4 is devoted to reviewing learning-driven heuristic algorithms in combinatorial optimization. Finally, conclusions are provided in Section 2.5.

2.2 Machine learning and data mining

In this section, we first introduce machine learning and data mining followed by a brief overview of common learning tasks and their corresponding algorithms.

2.2.1 Basic concepts

Machine learning and data mining are methods for extracting useful information from data. Given a set of examples, for instance, they try to discover what patterns appear frequently in the data or what function can discriminate positive from negative examples.

Data mining

Data Mining (DM), also popularly referred to as *Knowledge Discovery from Data (KDD)*, is defined as the process of discovering patterns in data [Witten *et al.*, 2016]. DM is a critical step of the KDD process, as shown in Figure 2.1. The process may be automatic or semi-automatic. The patterns discovered must be meaningful in that they lead to some advantage, such as economic advantage. The applied intelligent methods can be machine learning and statistic methods.

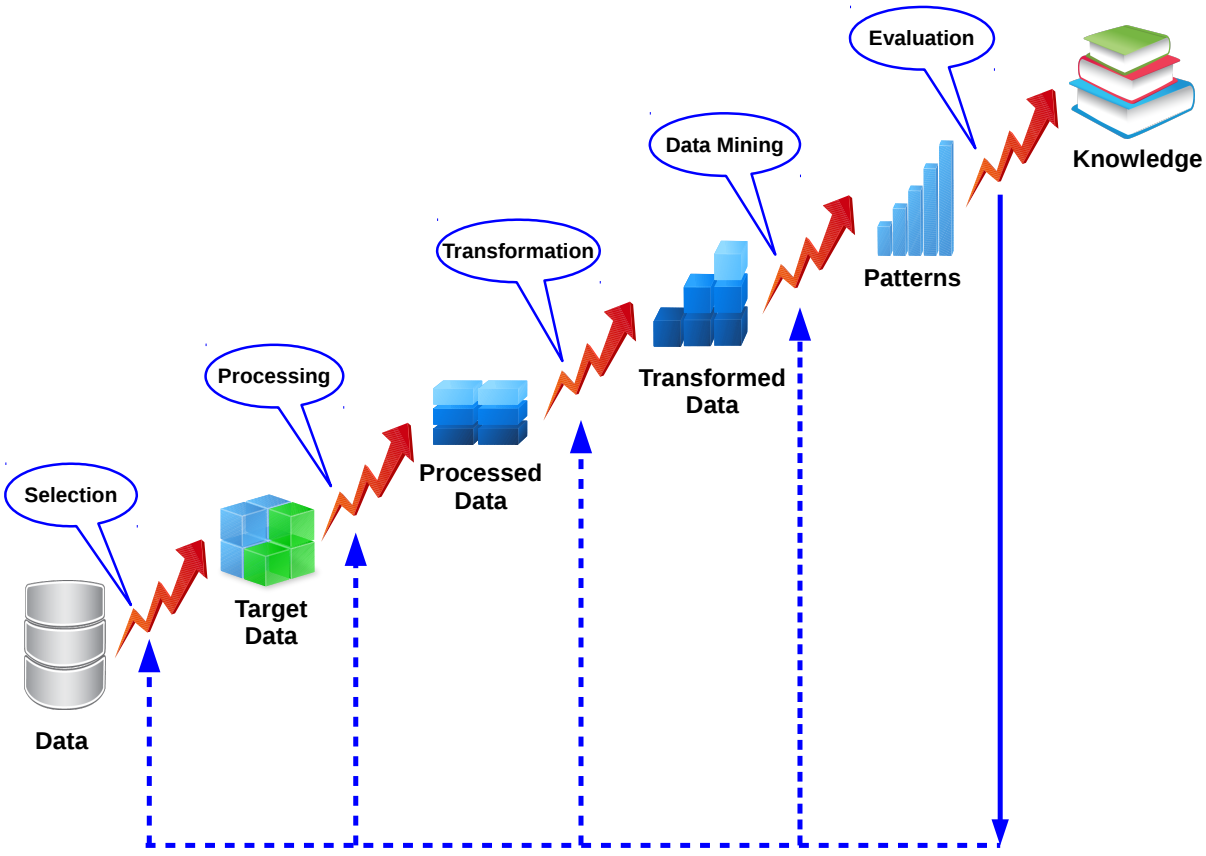

Figure 2.1: The general process of knowledge discovery.

Machine learning

Machine Learning (ML) is one of the important means for automatic knowledge acquisition. ML can be roughly defined as computational methods using experience to improve performance or to make accurate predictions. The "experience" refers to the past information available to the learner, which typically takes the form of electronic data collected and made available for analysis. Since the success of a learning algorithm greatly depends on the data used, ML is inherently related to DM and statistics methods. More generally, learning techniques are data-driven methods combining fundamental concepts in computer science with ideas from statistics learning, probability and optimization.

2.2.2 A brief overview of common learning tasks in machine learning

Learning tasks

There are a lot of tasks that machine learning tries to address. Correspondingly, a large number of algorithms have been proposed for solving each task, as shown in Figure 2.2. The reader interested in machine learning and data mining can refer to [Han *et al.*, 2011; Mohri *et al.*, 2012; Witten *et al.*, 2016]. In the following, only several representative learning tasks are briefly described.

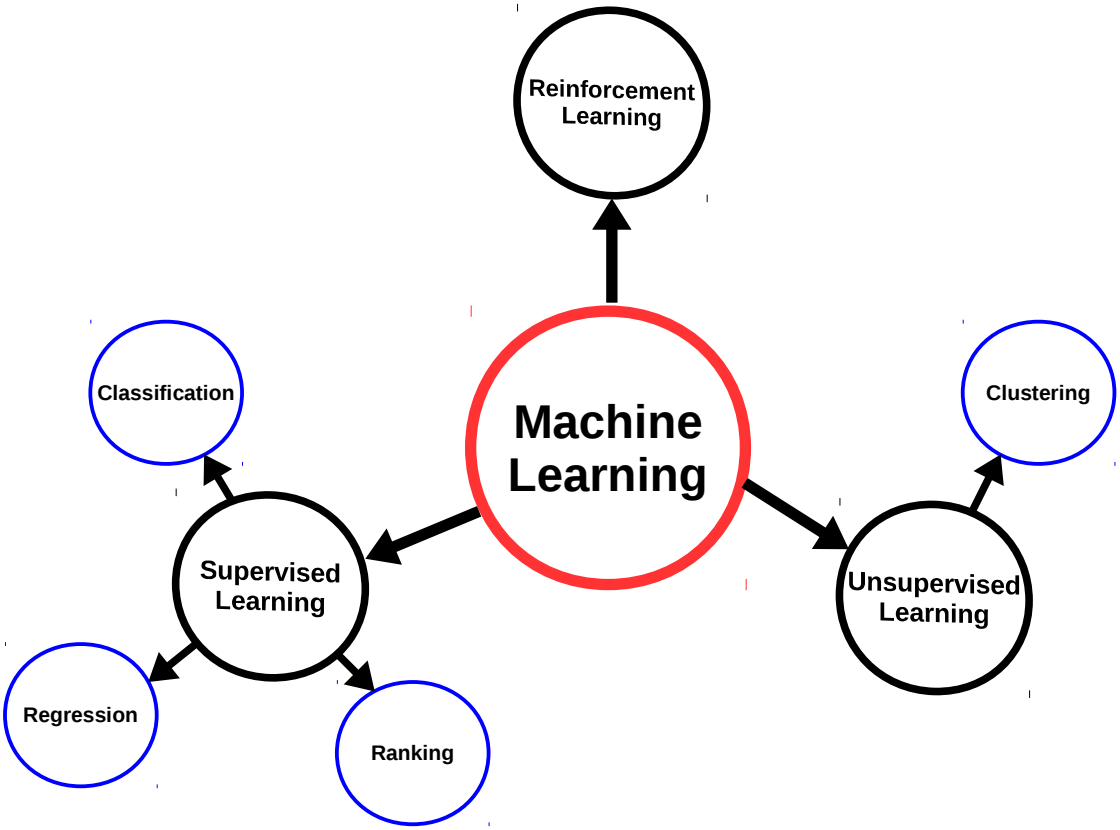

Figure 2.2: An overview of main learning tasks in machine learning.

— **Classification** aims to assign an instance to one of the predetermined classes. The number of classes in such tasks is often relatively small, but can be very large in some difficult tasks and even unbounded as in text classification and speech recognition. The well-known classification algorithms include *Support Vector Machine (SVM)*, *Naive Bayes (NB)*, *Bayesian Network (NB)*, *Decision Tree (DT)*, *k-Nearest Neighbor (KNN)*, *Multilayer Perceptron (MLP)*, and ensemble algorithms such as Adaboost and *Random Forest (RF)* [Han *et al.*, 2011].

— **Regression** is used to determine the relation between a set of input continuous variables and a continuous output variable. The common regression tasks mainly include linear regression, and more complex regression such as generalized linear regression, kernel ridge regression, support vector regression, and Bayesian regression. Both regression and classification are related to prediction, where regression predicts a real-value, whereas classification tries to predict a possible class. This is the key difference between regression and classification. Indeed, regression is a supervised learning task as well as classification [Han *et al.*, 2011].

— **Clustering** is the process of grouping a set of items into multiple groups or clusters so that items within a cluster have high similarity, but are very dissimilar to items in other cluster. The similarity and dissimilarity are usually defined based on the feature values of the items, and the number of

clusters is usually not known in advance. Clustering as an important learning task has its application in many areas such as business intelligence, Web search and biology. Clustering is the most representative unsupervised learning task. The well-known clustering algorithms include *k-means* and hierarchical clustering [Xu and Wunsch, 2005].

— **Ranking** is a learning task that tries to order items according to some criterion. A widely studied ranking problem is the ranking of retrieval results of a search engine, i.e., learning to rank [Cao *et al.*, 2007]. Given a query x and a set of documents $\mathcal{D}$, learning to rank tries to find a ranking of the documents in $\mathcal{D}$ that corresponds to their relevance with respect to x. So far, a number of different ranking problems have been introduced in the literature, though a commonly accepted terminology has not yet been established. Fürnkranz and Hüllermeier [Fürnkranz and Hüllermeier, 2010] proposed a categorization of ranking problems into object ranking, instance ranking and label ranking. The learning task of ranking naturally arises in many real-world applications, including recommendation systems, search engine design, natural language processing and bioinformatics. Not surprisingly, a large number of ranking algorithms have been proposed in the literature, where the well-known ranking algorithms include PageRank, ranking with SVMs, RankBoost, bipartite ranking [Mohri *et al.*, 2012].

Besides the above learning tasks, there is a widely-studied learning task known as **Association analysis**. Association analysis aims to discover interesting relationships hidden in large data sets. The uncovered relationships can be represented in the forms of association rules or sets of frequent patterns (e.g., itemsets, subsequences or substructure). Association analysis was originally applied to the market basket analysis [Agrawal *et al.*, 1993]. More specifically, it is used to analyze customer buying habits by discovering associations between the different items that customers place in their shopping baskets. The mined associations can further help retailers develop marketing strategies and plan their shelf space. The *Apriori* algorithm [Agrawal and Srikant, 1994] is the first association rule mining algorithm that uses the support-based pruning to systematically control the exponential growth of candidate itemsets. Other well-known mining algorithms include *FP-growth* and *FPmax** algorithm. In addition, many variants of frequent pattern mining have been designed for advanced data types. A detailed survey on frequent pattern mining can be found in [Aggarwal *et al.*, 2014]. It worth noting that association analysis has been used to solve many learning tasks, including classification, clustering and ranking [Tan and others, 2006].

The field of machine learning studies learning algorithms, which specify how the changes in the learner's behavior depend on the input signals received and on feedback signals from the environment. Machine learning can be roughly divided into three main categories based on the feedback available during the learning:

— **Supervised learning**, where the algorithm receives a set of labeled examples as training data and makes predictions for all unseen examples. This is the most widely-studied learning scenarios associated with classification, regression and ranking problems.

— **Unsupervised learning**, where the algorithm exclusively receives unlabeled training data. Usually, there is no labeled example available in the training data, so it is difficult to quantitatively assess the performance of an algorithm. Clustering is a representative example of unsupervised learning problems.

— **Reinforcement learning** is concerned with how an agent learns while interacting with an unknown environment in order to maximize its reward. The objective of the agent is to maximize its rewards and thus to determine the best course of actions to achieve that objective. There is no long-term reward signal provided by the environment, and the agent has to keep the balance between exploring unknown states and actions to gain more information about the environment and the rewards, and exploiting the information already collected to optimize its reward.

2.3 Related surveys

Machine learning and heuristic search are two different topics in artificial intelligence and operation research, respectively. They have been developed relatively independently. However, in last decade, researchers had an increasing interest in the interaction of these two domains. Several surveys on the synergies between data mining and operations research have been made in the literature (see Table 2.1). This section is devoted to summarizing and comparing these related surveys.

Table 2.1: Related surveys on using machine learning and data mining for heuristic optimization.

Survey	comment on the context *
[Baluja *et al.*, 2000]	ML helps large-scale optimization
[Jourdan *et al.*, 2006]	DM helps metaheuristic
[Santos *et al.*, 2008]	DM helps metaheuristic
[Olafsson *et al.*, 2008]	OR helps DM, and DM helps OR
[Osei-Bryson and Rayward-Smith, 2009]	OR helps DM, and DM helps OR
[Meisel and Mattfeld, 2010]	OR helps DM, and DM helps OR
[Corne *et al.*, 2012]	OR helps DM, and DM helps OR (multi-objective optimization)
[Talbi, 2016]	MP helps metaheuristic, CP helps metaheuristic, and ML helps metaheuristic
[Martins *et al.*, 2016]	DM helps metaheuristic
[Calvet *et al.*, 2017]	ML helps metaheuristic, and metaheuristic helps ML

* Operations Research (OR), Mathematical Programming (MP), Constraint Programming (CP), Machine Learning (ML) and Data Mining (DM).

The first survey on applying machine learning to large-scale optimization is made in [Baluja *et al.*, 2000]. This survey summarized 14 studies, and categorized them according to the different goals of the used learning technique, including (i) understanding search spaces; (ii) algorithm selection and tuning; (iii) learning generative models of solutions; and (iv) learning evaluation functions. This survey provides a coherent overview of some of the first steps in applying machine learning to large-scale optimization.

Jourdan et al. [Jourdan *et al.*, 2006] conducted a short survey on using data mining techniques to help metaheuristics in combinatorial optimization. The survey classified the references according to the localization of the knowledge integration, including (i) speeding-up metaheuristics during the evaluation; (ii) setting parameter values of the metaheuristics; (iii) generating initial solution or initial population; (iv) managing the population; (v) managing the operators and so on. This survey provides a quick comprehensive picture of the interest of combining data mining techniques and metaheuristics.

More specifically, a survey on the applications of the hybridization of the Greedy Random Adaptive Search Procedure (GRASP) with Data Mining (DM-GRASP) was made in [Santos *et al.*, 2008]. DM-GRASP is hybrid metaheuristic which combines the GRASP with a data mining process. This survey summarized three specific applications of the DM-GRASP heuristic, including set packing problem and maximum diversity problem, server replication for reliable multicast. The survey shows that the incorporation of a data-mining process into the original GRASP could improve the search process through the exploration of most promising search region. Furthermore, a detailed and recent survey of DM-GRASP is provided in [Martins *et al.*, 2016]. Besides the GRASP, in this survey several other metaheuristics, such as path relinking, iterated local search, variable neighborhood search, are successfully hybridized with a data mining procedure for solving $\mathcal{NP}$-hard problems. An interesting extension of DM-GRASP is also proposed and denoted as MDM-GRASP. Compared to the original DM-GRASP, MDM-GRASP tries data mining procedure multiple times instead of only once. This survey shows that both memoryless heuristics and memory-based heuristics can benefit from the use of data mining by obtaining better solutions in shorter computational time.

Also, four surveys [Olafsson *et al.*, 2008; Osei-Bryson and Rayward-Smith, 2009; Meisel and Mattfeld, 2010; Corne *et al.*, 2012] that aim to study the synergies of Operations Research (OR) and Data Mining (DM) are conducted in the literature. That is, the integration of optimization techniques into data mining, and the application of data mining techniques to help the optimization algorithms. These four surveys are summarized as follows:

— The survey [Olafsson *et al.*, 2008] provided an overview of the intersection of OR and DM. This survey aims to illustrate the range of interactions between the aforementioned two fields, and gives some detailed examples to explain the synergies.

— The survey [Osei-Bryson and Rayward-Smith, 2009] summarized nine papers of a special issue in Journal of the Operation Research Society. The papers in this issue can also be divided into two major categories: (i) OR to DM (six papers): the use of insights and techniques from OR to design effective DM algorithms; (ii) DM to OR (three papers): the use of insights or techniques from DM to more effectively address decision-making problems in OR.

— The survey [Meisel and Mattfeld, 2010] identified the synergies of OR and DM. More specifically, in this survey, authors defined three categories of synergies: (i) OR to increase DM efficiency; (ii) DM to increase OR effectiveness by replacement; (iii) DM to increase OR effectiveness by refinement, and illustrated each of them by examples.

— The survey [Corne *et al.*, 2012] studied the synergies between OR and DM. That is how OR and DM can benefit each other. However, a particular emphasis is placed on the field of the multi-objective optimization.

Recently, Talbi [Talbi, 2016] conducted a survey on hybrid metaheuristics. This survey summarized four different hybridizations: (i) combining metaheuristics with complementary metaheuristics; (ii) combining metaheuristics with mathematical programming approaches; (iii) combining metaheuristics with constraint programming approaches; and (iv) combining metaheuristics with machine learning and data mining techniques. Specially, we are interested in the hybridization between metaheuristics and machine learning techniques (see Section 5 of [Talbi, 2016]). The hybridization schemes between metaheuristics and machine learning techniques can be further divided into four categories:

— Lower-level relay hybrid scheme in which the knowledge extracted during the search may serve to change dynamically at run time the values of some parameters in the traditional single-solution based metaheuristics, greedy or multi-start metaheuristics.

— Low-level teamwork hybrid scheme in which the knowledge extracted during the search is incorporated into the recombination operators for the generation of new solutions.

— High-level relay hybrid scheme in which a priori knowledge is first extracted from the target optimization problem. Then, this knowledge is integrated into the metaheuristic for a more efficient search.

— High-level teamwork hybrid scheme in which a dynamically acquired knowledge is extracted in parallel during the search in cooperation with a metaheuristic.

Calvet et al. [Calvet *et al.*, 2017] also investigated the existing literature on the combination of metaheurisics with machine learning. The combinations of using machine learning to enhance metaheuristics and using metaheuristics to improve machine learning are investigated, respectively. Specially for the first combination, authors further divided the existing works into specifically-located hybridizations (where machine learning is applied in a specific procedure) and global hybridizations (i.e., machine learning has a higher effect on the metaheuristic design). In addition, a novel hybrid approach named learnheuristic is also proposed for solving dynamic combinatorial optimization problems.

During the last two decades, in the literature, many related works are devoted to completely or partially studying the synergies between operation research and data mining. However, there is still a lack of survey that focuses on using machine learning techniques to help heuristic search. Therefore, we conduct a brief literature review on using machine learning techniques to help heuristic search in the next section.

2.4 Machine learning driven heuristic search

The hybridization of machine learning techniques with heuristic algorithms is an emerging research field in the artificial intelligence and operation research communities. In the literature, a great number of

efforts have been made to hybridize heuristics with machine learning techniques. In contrast with the view of using heuristic algorithms to enhance machine learning techniques, this work focuses on making a survey of researches that use machine learning techniques to help heuristic search.

The combinations of machine learning techniques and heuristic algorithms have been applied to successfully solve many COPs. Machine learning techniques can be hybridized with heuristic algorithms in following two directions:

— **Online methods** are trained using information collected during the current execution of the search procedure. For example, to find patterns useful in enhancing the construction phase of GRASP [Santos *et al.*, 2008].

— **Off-line methods** aims to identify interesting structures and patterns shared by the entire class of problems. For example, to recommend the best metaheuristic for a new TSP instance by using a meta-learning approach based on label ranking algorithms. [Kanda *et al.*, 2016].

Also, learning-driven heuristic optimization algorithms can be categorized according to the purpose of introducing learning techniques. In general, heuristic search can benefit from the applications of machine learning techniques in four aspects:

— improve the quality of the obtained solutions,

— speed up the search to find the best solution,

— optimize the algorithm parameters,

— select heuristic algorithms.

In fact, some of the above-categories have been studied under different terms. For example, the study of using machine learning techniques to optimize the parameters of heuristic algorithm is directly related to the *Reactive Search Optimization* and *Algorithm Configuration*. Also, the category of using machine learning techniques to select the best heuristic algorithm has been widely studied under the umbrella of *Algorithm Selection* and *Hyper-heuristics*. Compared to hyper-heuristics, algorithm selection aims to select the best heuristic algorithm according to instance-specific features instead of depending upon the current problem state or search stage. A great deal of effort has been made for reactive search optimization, algorithm configuration, algorithm selection and hyper-heuristic respectively. The corresponding surveys of each term are also available in the literature [Battiti *et al.*, 2008; Burke *et al.*, 2013; Kotthoff, 2014]. Therefore, our following investigation will focus on the categories of using machine learning techniques to improve the quality of the obtained solutions and to speed up the search to find the best solution.

2.4.1 Improving the quality of the obtained results

The quality of solutions obtained by heuristic algorithm can be improved by incorporating knowledge into the search. Various machine learning techniques have successfully combined with heuristic algorithms for improving the quality of the obtained solutions. A brief summary of these works is provided in Table 2.2. From this table, we observe that there is an increasing research effort made on using machine learning techniques to improve the solution quality. Many different machine learning techniques have successfully enhanced different optimization algorithms. In what follows, we review and organize them according to the applied machine learning techniques.

— **Classification**: Cadenas et al. [Cadenas *et al.*, 2009] proposed a centralized hybrid metaheuristic cooperative strategy to solve optimization problems, in which knowledge is incorporated through a set of fuzzy rules and models obtained from a fuzzy decision tree applied to the records of the results returned by individual metaheuristics. Samorani and Laguna [Samorani and Laguna, 2012] developed a general purpose data mining driven neighborhood search approach that attempts to learn the guiding constraints to lead the search to escape from local optima. Computational results on the constrained task allocation problem and the matrix bandwidth minimization problem show that adding these guiding constraints to a simple tabu search improves the quality of the solutions found.

Table 2.2: A summary of works on using machine learning to improve the quality of the obtained solutions.

Reference	metaheuristic *	machine learning technique	combinatorial search problem
[Boyan and Moore, 1998]	LS	reinforcement learning	bin-packing, channel routing, satisfiability and etc
[Telelis and Stamatopoulos, 2001]	constructive heuristic	instance-based learning and kernel regression	knapsack and set partitioning problems
[Gersmann and Hammer, 2005]	simple greedy strategy	support vector machine and reinforcement learning	resource constraint project scheduling problem
[de Lima Júnior *et al.*, 2007]	GRASP and GA	reinforcement learning	symmetrical traveling salesman problem
[Barreto *et al.*, 2007]	sequential heuristic	hierarchical and non-hierarchical clustering	capacitated location-routing problem
[Ventresca and Tizhoosh, 2008]	PBIL	opposition-based learning	traveling salesman problem
[Cadenas *et al.*, 2009]	TS, SA and GA	fuzzy decision tree and fuzzy rules	0-1 knapsack problem
[Ergezer and Simon, 2011]	biogeography-based optimization	opposition-based learning	graph coloring and traveling salesman problems
[Wang *et al.*, 2012]	TS and EDA	clustering	maximum diversity problem
[Samorani and Laguna, 2012]	TS	hyperplan-based classification	constrained task allocation and bandwidth minimization problems
[Jędrzejowicz and Ratajczak-Ropel, 2014]	multi-agents optimization	reinforcement learning	resource constrained project scheduling problem
[Pinto *et al.*, 2015]	PSO and SA	regression model	space allocation in the retail industry
[Rojas-Morales *et al.*, 2016]	ant based algorithm	opposition-based learning	multidimensional knapsack problem
[Arin and Rabadi, 2016]	Meta-RaPS and EDA	reinforcement learning	0-1 multidimensional knapsack problem
[García *et al.*, 2017]	Cuckoo search and Black hole metaheuristic	k-means	multidimensional knapsack problem
[Toffolo *et al.*, 2018]	LS	learning automaton	swap-body vehicle routing problem

* Meta-heuristic for Randomized Priority Search (Meta-RaPS), Greedy Random Adaptive Search Procedure (GRASP), Local Search (LS), Tabu Search (TS), Simulated Annealing (SA), Genetic Algorithm (GA), Memetic Algorithm (MA), Population-Based Incremental Learning (PBIL), Particle Swarm Optimization (PSO), Estimation of Distribution Algorithm (EDA)

— **Regression**: Telelis et al. [Telelis and Stamatopoulos, 2001] proposed a heuristic methodology for solving the knapsack and the set partitioning problems, which employs instance-based learning and function approximation through kernel regression, for guiding any constructive search to promising regions of the search space, as far as optimality is concerned. Pinto et al. [Pinto *et al.*, 2015] combined regression models and metaheuristics to optimize space allocation in the retail industry. A metaheuristic is used to search different hypothesis of space allocations for multiple product categories, guided by the predictions made by regression models that estimate the sales for each category based on the assigned space.

— **Clustering**: Barreto et al. [Barreto *et al.*, 2007] integrated several hierarchical and non-hierarchical clustering techniques (with several proximity functions) into a sequential heuristic algorithm for solving the capacitated location routing problem. Wang et al. [Wang *et al.*, 2012] proposed a learning-based heuristic LTS_EDA for the maximum diversity problem, which applies k-means clustering and estimation of distribution algorithm to extract useful information from the search history of tabu search in order to drive the search to promising search region. García et al. [García *et al.*, 2017] presented a k-means transition ranking framework to solve the multidimensional knapsack problem. The proposed framework binaries two different continuous population-based metaheuristics (i.e., Cuckoo search and Black hole) by a k-means technique.

— **Reinforcement learning**: Boyan and Moore [Boyan and Moore, 1998] applied reinforcement learning techniques to learn an evaluation functions which predicts the outcome of a local search algorithm, such as hill-climbing or WALKSAT, as a function of state features along its search trajectories. The learning evaluation function is then used to bias future search trajectories toward better solutions. Gersmann and Hammer [Gersmann and Hammer, 2005] investigated the capability of reinforcement learning to improve a simple greedy strategy for solving the resource constraint project scheduling problem. de Lima Júnior et al. [de Lima Júnior *et al.*, 2007] applied Q-learning to the constructive phase of GRASP and to generate the initial population of a genetic algorithm for the symmetrical traveling salesman problem. Jedrzejowicz and Ratajczak-Ropel [Jędrzejowicz and Ratajczak-Ropel, 2014] proposed the A-Team with reinforcement learning for solving the resource constrained project scheduling problem. Instead of a static strategy, reinforcement learning is used to supervise the interactions between optimization agents and the common memory. Arin and Rabadi [Arin and Rabadi, 2016] integrated Estimation of Distribution Algorithm (EDA) and Q-learning into the *Metaheuristic for Randomized Priority Search (Meta-RaPS)* for the 0-1 multidimensional knapsack problem, and respectively denoted as Meta-RaPS EDA and Meta-RaPS Q-Learning. Experimental results show that they are very competitive compared to the state-of-the-art algorithms, and especially that Meta-RaPS EDA appears to perform better than Meta-RaPS Q-Learning. Toffolo et al. [Toffolo *et al.*, 2018] proposed a stochastic local search with both general and dedicated heuristic components, i.e., a subproblem optimization scheme and a learning automation. The incorporated learning automation is used to determine the probabilities of selecting each of the available neighborhoods.

— **Opposition-based learning**: Ventresca and Tizhoosh [Ventresca and Tizhoosh, 2008] proposed a diversity maintaining population-based incremental learning algorithm for solving the traveling salesman problem, where the concept of opposition was used to control the amount of diversity within a given sample population. Ergezer and Simon [Ergezer and Simon, 2011] hybridized open-path opposition and circular opposition with biogeography-based opposition for solving the graph coloring problem and traveling salesman problem. Rojas-Morales et al. [Rojas-Morales *et al.*, 2016] incorporated an opposite learning phase to Ant Knapsack that is an ant based algorithm for the multi-dimensional knapsack problem, for discarding regions of the search space. This opposite knowledge is then used by Ant Knapsack for solving the original problem.

2.4.2 Speeding up the heuristic search

To speed up the search, machine learning techniques can help heuristic algorithm from two different levels, i.e., reducing the cost of evaluating the objective function and reducing the size of search space.

In some cases, the objective function can be very expensive to compute, thus reducing the cost of evaluating the objective function will be able to speed up the whole search. To achieve it, heuristic algorithms could benefit from machine learning in two aspects. On the one hand, machine learning techniques can be used to build approximate models of the objective function. For example, Guo et al. [Guo *et al.*, 2017] proposed a Hybrid Evolutionary Algorithm (HEA) for two-stage capacitated facility location problems, which incorporates an approximating fitness evaluation approach based on the extreme learning machine to alleviate the computational burden of HEA. On the other hand, machine learning technique can also be employed to reduce the number of complete evaluations during the search. For example, Rasheed and Hirsh [Rasheed and Hirsh, 2000] proposed an improving genetic-algorithm based optimization using informed genetic operators. The idea is to generate several purely random candidates at every step of initial population generation or crossover or mutation, then rank these random candidates using some reduced models. Only the best random candidate is taken as the final outcome for initialization, crossover or mutation. Empirical results demonstrate that the proposed method can significantly speed up the genetic algorithm.

Besides the effort to reduce the cost of evaluating the objective function, heuristic algorithms can also benefit from machine learning techniques by reducing the search space. Specifically, machine learning techniques can be used to find interesting and useful properties of the candidate solutions. Uncovering information can be used to construct a promising candidate solution which is close to the optimal solution, thus reducing the size of the search space. Association analysis technique is the most widely-used machine learn technique, and has achieved great success on many combinatorial optimization problems. Ribeiro et al. [Ribeiro *et al.*, 2004; Ribeiro *et al.*, 2006] first hybridized GRASP with data mining techniques (DM-GRASP) for solving the set packing problem. DM-GRASP organizes its search process into two sequential phases, and incorporates an association rule mining at the second phase. That is, after executing a predefined number of GRASP iterations in the first phase, the data mining process extracts patterns from an elite set of solution (collected from the first phase) which will guide the following iterations.

DM-GRASP has been applied to solve several problems including the maximum diversity problem [Santos *et al.*, 2005], the server replication for reliable multicast problem [Santos *et al.*, 2006b], and the p-median problem [Plastino *et al.*, 2011]. A survey on some significant applications of DM-GRASP can be found in [Santos *et al.*, 2008]. An interesting extension of DM-GRASP is to execute the data mining procedure multiple times instead of only once [Plastino *et al.*, 2014; Guerine *et al.*, 2016; Martins *et al.*, 2016]. Compared to DM-GRASP where the data mining call occurs once at the midway of the whole search process, the multi-mining version performs the mining task when the elite set stagnates. The same idea has been explored recently by hybridizing data mining and GRASP enhanced with path-relinking [Barbalho *et al.*, 2013] or variable neighborhood descent [Guerine *et al.*, 2016].

In addition to GRASP, data mining has also been hybridized with other metaheuristics like evolutionary algorithms. To improve the performance of an evolutionary algorithm applied to an oil collecting vehicle routing problem, a hybrid algorithm (GADMLS) combining genetic algorithm, local search and data mining

was proposed in [Santos *et al.*, 2006a]. Another hybrid approach (GAAR) that uses a data mining module to guide an evolutionary algorithm was presented in [Raschip *et al.*, 2015b; Raschip *et al.*, 2015a] to solve the constraint satisfaction problem. Besides the standard components of a genetic algorithm, a data mining module is added to find association rules (between variables and values) from an archive of best individuals found in the previous generations.

Apart from GRASP and evolutionary algorithms, it has been shown that other heuristics can also benefit from the incorporation of a data mining procedure. For example, a data mining approach was used to reduce the search space of local search algorithms, i.e., extracting variable associations from the instance to be solved in order to identify promising pairs of flipping variables in the neighborhood search, and verified it on the set covering and set partitioning problems [Umetani, 2017]. A preliminary experimental results of this paper was presented in [Umetani, 2015]. Another example is the hybridization of neighborhood search with data mining techniques for solving the p-median problem [Reddy *et al.*, 2012]. Also for the p-median problem, a data mining procedure was integrated into a multistart hybrid heuristic [Martins *et al.*, 2014], which combines elements of different traditional metaheuristics (e.g., local search) and uses path-relinking as a memory-based intensification mechanism. Finally, the widely-used iterated local search method was also hybridized with data mining for solving the set covering with pairs problem [Martins *et al.*, 2016].

2.4.3 Optimizing the algorithm parameters

The optimization of an algorithm's performance by setting its (typically few and numerical) parameters is known as *parameter tuning*. Parameter tuning is a crucial issue both in the scientific research and in the practical use of heuristics. The study showed that, to solve an optimization problem, only 10% of the total time is spent on heuristic algorithm design and test, and the remaining 90% is consumed by the parameter configurations [Adenso-Diaz and Laguna, 2006]. This is particularly true when designing heuristics algorithms for solving combinatorial optimization problems. Given a heuristic algorithm for solving a problem instance, we usually have following two observations. On the one hand, for an instance, the performance of a heuristic algorithm greatly depends on its parameter configurations. For example, tabu search often performs differently with different tabu tenures. On the other hand, different problem instances usually require different parameter configurations for a heuristic algorithm to find good solutions. The importance of fine-tuning heuristic algorithms has also been emphasized in the literature.

> "The selection of parameter values that drive heuristics is itself a scientific endeavor and deserves more attention than it has received in the operations research literature. This is an area where the scientific method and statistical analysis could and should be employed". [Barr *et al.*, 1995]

Control and tuning of heuristic algorithms parameters have been widely studied during the last decades especially in the field of combinatorial optimization. Machine learning techniques can be applied to accelerate the optimization performance by tuning the parameters of heuristic algorithms automatically [Baluja *et al.*, 2000]. The possible parameters include domain-specific terms, such as the coefficients of extra objective function; generic parameters of the heuristics, such as the cooling-rate coefficient in simulated annealing; and even high-level parameters, such as the which heuristic is selected in hyper-heuristics.

The parameter tuning of metaheuristics allows greater flexibility and robustness but requires a careful initialization, since different parameter values have a great influence on the efficiency and effectiveness of the search. Talbi [Talbi, 2009] classified the approaches for the parameter tuning of metaheuristics into two categories:

— **Offline tuning**: the parameter values are defined before the execution of heuristic algorithms. For example, Al-Duoli and Rabadi [Al-Duoli and Rabadi, 2014] hybridized an Inductive Decision Tree (IDT) with the Meta-RaPS for solving the capacitated vehicle routing problem. This hybridization tries to use an IDT to perform on-line tuning of the parameters in Meta-RaPS. The integrated IDT is used to find a favorable parameter by using the information collected during the construction and

improvement phases of Meta-RaPS, this uncovering parameter is then used in future Meta-RaPS iterations.

— **Online tuning**: the parameter values are controlled and updated in a dynamic or adaptive way through the execution of heuristic algorithms. For instance, Lessmann et al. [Lessmann *et al.*, 2011] designed a data mining based approach to tune the parameters of a particle swarm optimization algorithm. This hybrid approach employs regression models to learn suitable regression parameter values from past moves of the metaheuristic in an online fashion.

As we mentioned above, the application of machine learning to parameter tuning of a heuristic algorithm has been studied under the umbrella of *Reactive Search Optimization (RSO)* [Zhang, 2010]. RSO advocates the integration of subsymbolic machine learning techniques into heuristics for tuning control parameters in an online way. For example, dos Santos et al. [dos Santos *et al.*, 2014] proposed a reactive search approach using reinforcement learning, more specifically the Q-learning algorithm for the self-tuning of the implemented algorithm, applied to the symmetric traveling salesman problem. Benlic et al. [Benlic *et al.*, 2017] proposed an improved Breakout local search for solving the vertex separator problem by incorporating a parameter control mechanism that draws upon ideas from reinforcement learning theory to reach an interdependent decision on the number and on the type of perturbation moves.

Besides RSO, the problem of parameter tuning is also studied as an *Algorithm Configuration (AC)* [Hutter *et al.*, 2009] problem. AC aims to choose the best parameter configuration for solving a problem with an algorithm. Given a parameterized algorithm $\mathcal{A}$ with possible parameter settings $\mathcal{C}$, a set of training problem instances $\mathcal{T}$, and a performance metric $f : \mathcal{T} \times \mathcal{C} \rightarrow \mathcal{R}$, the algorithm configuration problem is to find a parameter configuration $c \in \mathcal{C}$ that minimizes f across the instances in $\mathcal{T}$ [Hutter *et al.*, 2009]. A comprehensive literature summary of algorithm configuration is available at website `http://aclib.net/acbib/`. Also, a survey on algorithm configuration and parameter tuning procedures is provided in [Hoos, 2012].

2.4.4 Selecting heuristic algorithms

The task of selecting heuristic algorithm based on instance features is known as *Algorithm Selection (AS)*. The algorithm selection problem was initially proposed in [Rice, 1976]. It aims to learn a mapping from instance space X to algorithm space P. Figure 2.3 shows the general process of algorithm selection. The selection model is often implemented with machine learning methods. Based on gathered training data (i.e., instance features and corresponding performance data), AS learns a machine learning model that maps from instance features to a well-performing algorithm. The philosophy behind the algorithm selection is that no single algorithm could achieve the best performance on all instances, and each algorithm performs well only on a particular instance or a class of instances. Therefore, selecting the most suitable algorithm for each particular problem instance has the potential to significantly boost the performance in practice.

The algorithm selection has received a lot of attention in area of combinatorial optimization that deal with solving hard combinatorial problems. Researchers have recognised that using algorithm selection techniques can provide significant performance improvements with relatively little effort. An established approach to solve the algorithm selection problem is to use machine learning techniques [Fink, 1998]. Almost all contemporary algorithm selection approaches employ machine learning to learn the performance mapping from problem instances to algorithms using features extracted from instances [Kotthoff, 2014]. During the last decades, a number of machine learning techniques have been used to perform algorithm selection, including classification (such as Bayesian network, decision tree, random forest, k-nearest neighbors, support vector machine, decision tree, multilayer perceptrons, naive Bayes), clustering, regression, ranking and case-based reasoning. A systematic comparison and evaluation of the different machine learning techniques in algorithm selection for combinatorial search problems is available in [Kotthoff *et al.*, 2012]. A detailed survey on algorithm selection for combinatorial search problems is available in [Kotthoff, 2014; Kotthoff, 2016], and a comprehensive literature summary of algorithm selection is also available at website `https://larskotthoff.github.io/assurvey/`.

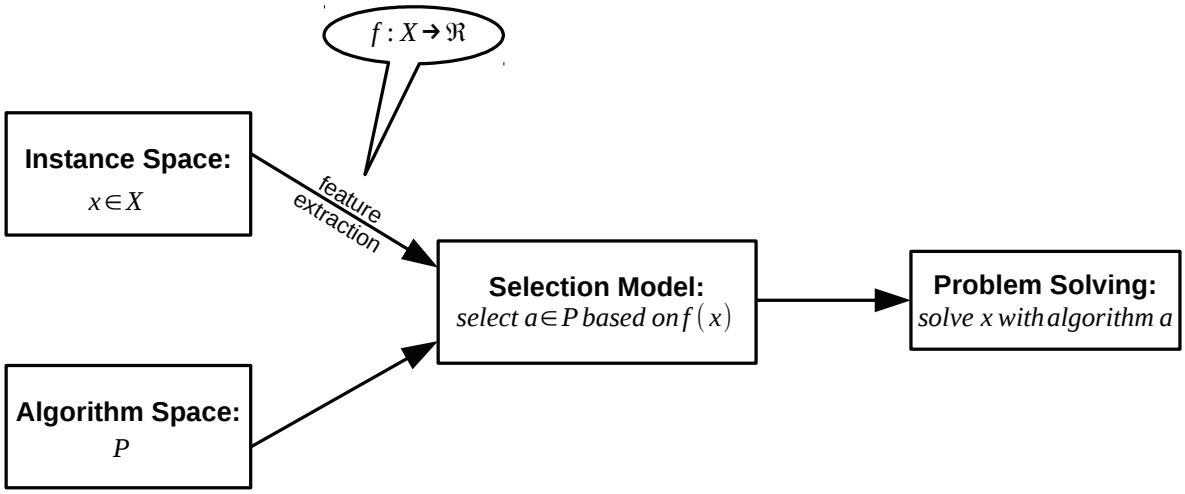

Figure 2.3: A general description of algorithm selection.

The selection of the best algorithm for a given task has been studied in a sub-area of machine learning known as *meta-learning* [Lemke *et al.*, 2015]. Meta-learning aims to deal with selecting/generating algorithms tailored to the problem in machine learning. The area does that by benefiting from previous runs of each algorithm on different datasets. Hence, while a base learner accumulates experience over a specific problem, meta-learners accumulate experience from the performance of the learner in different applications. The meta-learning approaches have also been successfully applied to select the best meta-heuristics for the combinatorial optimization problems. For example, Kanda et al. [Kanda *et al.*, 2016] used label ranking algorithms like k-nearest neighbor, decision tree and multilayer perceptron to induce a meta-learning model able to associate properties of the traveling salesman problem instances with the optimization performance of metaheuristics.

Besides meta-learning, algorithm selection is also highly related to *hyper-heuristics*. The term hyper-heuristics is relatively new, and it was originally used in [Cowling *et al.*, 2001] to describe *heuristics to choose heuristics* in the area of combinatorial optimization. In this case, a hyper-heuristic was defined as a high-level approach that, given a problem instance and a number of heuristics, selects and applies an appropriate heuristic at each decision point. The definition of hyper-heuristics has been recently extended to refer to a search method or learning mechanism for selecting or generating heuristics to solve computational search problem. Hyper-heuristics can be divided into two categories: heuristic selection and heuristic generation. Compared to problem-specific heuristics and metaheuristics, hyper-heuristics operate on a search space of heuristics rather than directly on the search space of solutions. A great number of machine learning techniques have been successfully used as the high-level strategy in hyper-heuristics, such as reinforcement learning, case-based reasoning and learning classifier systems. A detailed survey of hyper-heuristics can be found in [Burke *et al.*, 2013]. For example, Burke et al. [Burke *et al.*, 2002] proposed a hyper-heuristic method using Case-Based Reasoning (CBR) for solving course timetabling problem. The basic idea behind this method is that we maintain a case base of information about the most successful heuristics for a range of previous timetabling problems to predict the best heuristic for the new problem in hand using the previous knowledge. Knowledge discovery techniques are used to carry out the training on the CBR system to improve the system performance on the prediction.

The algorithm selection problems can be solved by meta-learning and hyper-heuristic approaches. An investigation on meta-learning and hyper-heuristics approaches used for solve algorithm selection problems is provided in [Cruz-Reyes *et al.*, 2012].

2.5 Chapter conclusion

In this chapter, we presented a brief overview of machine learning-driven heuristic optimization. Based on the purposes of applying machine learning techniques, we divide the existing learning-driven heuristic optimization algorithms into four categories: using machine learning to improve the solution quality, using machine learning to speed up the heuristic search, using machine learning to optimize algorithm parameters, and using machine learning to select heuristics.

3

Probability Learning based Local Search for Graph Coloring Problem [1]

Grouping problems aim to partition a set of items into multiple mutually disjoint subsets according to some specific criterion and constraints. Grouping problems cover a large class of important combinatorial optimization problems that are computationally difficult in general. In this chapter, we propose a general solution approach for grouping problems, i.e., *Probability learning based Local Search (PLS)*, which combines a probability learning scheme with an optimization procedure. The viability of the proposed approach is verified on a well-known representative grouping problem (i.e., *Graph Coloring Problem (GCP)*), and the corresponding algorithm is denoted as PLSCOL. Experimental studies on popular DIMACS benchmark graphs indicate that PLSCOL achieves competitive performances compared to a number of well-known coloring algorithms. This chapter is based on three articles [Zhou *et al.*, 2016], [Zhou *et al.*, 2017b] and [Zhou *et al.*, 2017a].

Contents

1. **Instead of the term "reinforcement learning" originally used in [Zhou *et al.*, 2016], in this thesis, we adopt the more explicit and appropriate term "probability learning" lately used in [Zhou *et al.*, 2017a].**

3.1 Introduction

Grouping Problems (GPs) aim to partition a set of items into a collection of mutually disjoint subsets according to some specific criterion and constraints. Grouping problems naturally arise in numerous domains. Well-known grouping problems include, for instance, *Graph Coloring Problem (GCP)* [Garey and Johnson, 1979; Galinier and Hao, 1999; Lewis, 2009; Elhag and Özcan, 2015], timetabling [Lewis and Paechter, 2007; Elhag and Özcan, 2015], bin packing [Falkenauer, 1998; Quiroz-Castellanos *et al.*, 2015], scheduling [Kashan *et al.*, 2013] and clustering [Agustı *et al.*, 2012]. Formally, given a set V of n distinct items, the task of a grouping problem is to partition the items of set V into k different groups g_i $(i = 1, \ldots, k)$ (k can be fixed or variable), such that $\cup_{i=1}^{k} g_i = V$ and $g_i \cap g_j = \emptyset, i \neq j$ while taking into account some specific constraints and optimization objective. For instance, the graph coloring problem is to partition the vertices of a given graph into a minimum number of k color classes such that adjacent vertices must be put into different color classes.

According to whether the number of groups k is fixed in advance, grouping problems can be divided into constant grouping problems or variable grouping problems [Kashan *et al.*, 2013]. The number of groups k is a fixed value in problems such as identical or non-identical parallel-machines scheduling problem, while in other settings, k is variable and the goal is to find a feasible grouping with a minimum number of groups, such as the bin packing problem and graph coloring problem. Grouping problems can also be classified according to the types of the groups. A grouping problem with identical groups means that all groups have similar characteristics, thus naming of the groups is irrelevant. Aforementioned examples such as identical parallel-machines scheduling, bin-packing and graph coloring belong to this category. In other grouping problems where the groups have different characteristics, hence, swapping items between two groups will result in a new grouping. This is the case for the non-identical parallel-machines scheduling problem.

Many grouping problems, including the examples mentioned above are $\mathcal{NP}$-hard, thus computationally challenging. A number of heuristic approaches for grouping problems, in particular based on genetic algorithms, have been proposed in the literature with varying degrees of success [Falkenauer, 1998; Galinier and Hao, 1999; Quiroz-Castellanos *et al.*, 2015]. These approaches are rather complex since they are population-based and often hybridized with other search methods like local optimization.

In this chapter, we present the *Probability learning based Local Search (PLS)* approach for grouping problems, which combines the probability learning scheme with an optimization procedure. Our proposed PLS approach tries to learn a generative model of solutions. For a grouping problem with its k groups, we associate to an item a probability vector with respect to each possible group and determine the group of the item according to the probability vector. Once all items are assigned to their groups, a grouping solution is generated. Then, the optimization procedure is invoked to improve this solution until a local optimum is attained. Afterwards, the probability vector of each item is updated by comparing the group of the item in the starting solution and in the attained local optimum solution. If an item stays in its original group, then we reward the selected group of the item, otherwise we penalize the original group and compensate the new group (i.e., expected group). There are two key issues that need to be considered, i.e., how do we select a suitable group for each item according to the probability vector, and how do we smooth the probabilities to avoid potential search traps. To handle these issues, we design two strategies: a hybrid group selection strategy that uses a noise probability to switch between random selection and greedy selection; and a probability smoothing mechanism able to forget old decisions.

To evaluate the viability of the proposed PLS method, we use the well-known GCP as a case study. We show computational experiments on DIMACS benchmark graphs. Computational results demonstrate that the proposed PLSCOL approach, despite its simplicity, achieves competitive performances on most tested instances compared to many existing algorithms. With an analysis of three important issues of PLSCOL, we show the effectiveness of combining probability learning and descent-based local search. We also assess the contribution of the probability smoothing technique to the performance of PLSCOL.

The rest of the chapter is organized as follows. In the next section, we present the proposed probability learning based local search approach. Section 3.3 demonstrates a specific application of the general PLS

approach to solve the graph coloring problem. Section 3.4 shows extensive computational results and comparisons on the DIMACS challenge benchmark instances. Section 3.5 is devoted to conducting some interesting experimental investigations of the proposed approach. Finally, conclusions and some potential research directions are provided in Section 3.6.

3.2 Probability learning based local search

Grouping problems aim to partition a set of items into k disjoint groups according to some imperative constraints and an optimization criterion. For our PLS approach, we suppose that the number of groups k is given in advance. Note that such an assumption is not necessarily restrictive. In fact, to handle a grouping problem with variable k, one can repetitively run PLS with different k values. We will illustrate this approach on the graph coloring problem in Section 3.3.

3.2.1 Main scheme

The proposed *Probability learning based Local Search (PLS)* is a general framework designed for solving grouping problems. Generally, a grouping problem aims to group a set of given items into a fixed or variable number of groups while respecting some specific requirements. The basic idea of PLS is to iterate through a group selection phase (to generate a starting solution S according to a *probability matrix* P (see Figure 3.2) that indicates for each item its chance to belong to each group), an optimization phase (to obtain an improved solution S' from S), and a probability learning phase.

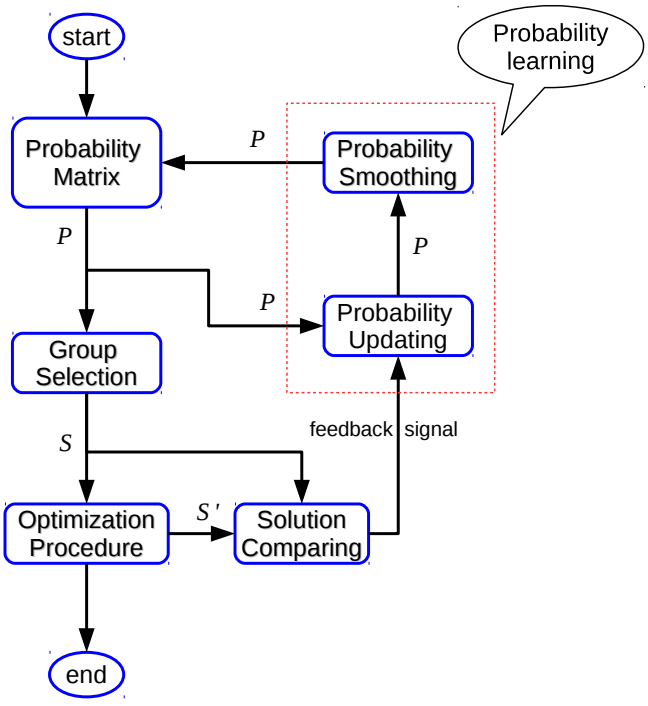

Figure 3.1: A schematic diagram of PLS for grouping problems. From a starting solution generated according to the probability matrix P, PLS iteratively runs until its meets its stop condition (see Sections 3.2.2-3.2.5 for more details).

The schematic diagram of PLS for grouping problems is depicted in Figure 3.1 while its algorithmic pseudo-code is provided in Algorithm 1. To apply PLS to a grouping problem, several procedures need to be specified. The *GroupSelection* procedure (line 5) generates a starting solution S. The *Optimization* procedure (line 6) aims to obtain a local optimum S' from the starting solution S. During the probability

learning phase (lines 10-11), *ProbabilityUpdating* updates the probability matrix P by comparing the starting solution S and its improved solution S', while *ProbabilitySmoothing* erases old decisions that become useless and may even mislead the search. The modifications of the probability matrix rely on information about group changes of the items. Using the updated probability matrix, a new starting solution is built for the next round of the *Optimization* procedure (this is also called an iteration).

Algorithm 1: Pseudo-code of our PLS for grouping problems.

Input: Grouping problem instance I and the number of groups k
Output: The best solution S^* found so far

```
1  // I is supposed to be a minimization problem
2  begin
3      P = [p_ij = 1/k]_{i=1,2,...,n,j=1,2,...,k}
4      while stopping condition not reached do
5          S ← GroupSelection(P);                          /* Section 3.2.2 */
6          S' ← Optimization(S);                           /* Section 3.2.3 */
7          // update the best solution found so far
8          if f(S') < f(S*) then
9              S* ← S'
10         P ← ProbabilityUpdating(P, S, S');              /* Section 3.2.4 */
11         P ← ProbabilitySmoothing(P);                    /* Section 3.2.5 */
```

We first define a probability matrix P of size $n \times k$ (n is the number of items and k is the number of groups, see Figure 3.2 for an example). An element p_{ij} denotes the probability that the i-th item v_i selects the j-th group g_j as its group. Therefore, the i-th row of the probability matrix defines the probability vector of the i-th item and is denoted by p_i. At the beginning, all the probability values in the probability matrix are set as $1/k$. It means that all items select a group from the available k groups with equal probability.

	g_1	g_2	$\cdots$	g_k
v_1	p_{11}	p_{12}	$\cdots$	p_{1k}
v_2	p_{21}	p_{22}	$\cdots$	p_{2k}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
v_n	p_{n1}	p_{n2}	$\cdots$	p_{nk}

Figure 3.2: Probability matrix P for n items and k groups.

At each iteration, each item v_i, $i \in \{1, 2, ..., n\}$ selects one suitable group g_j, $j \in \{1, 2, ..., k\}$ by applying a group selection strategy (Section 3.2.2) based on its probability vector p_i. Once all the items are assigned to their groups, a grouping solution S is obtained. Then, this solution is improved by a local optimization procedure to attain a local optimum denoted by S' (Section 3.2.3). During the probability learning phase, we try to determine that whether each item moves from its original group to a new group in S' or it still stays in its original group of S by comparing the starting solution S and the improved solution

S'. Then PLS adjusts the probability matrix accordingly by the following rule. If an item stayed in its original group, the selected group (called *correct group*) is rewarded for the item; if the item moved to a new group in S', the discarded group (called *incorrect group*) is penalized and the new group (called *expected group*) for the item is compensated (Section 3.2.4).

Next, we apply a probability smoothing technique to smooth each item's probability vector (Section 3.2.5). Hereafter, PLS iteratively runs until a predefined stop condition is reached (e.g., a legal solution is found or the number of iterations without improvement exceeds a maximum allowable value). In the following subsections, the four key components of our PLS approach are presented in detail.

3.2.2 Group selection procedure

At each iteration of PLS, each item v_i needs to select a group g_j from the k available groups according to its probability vector p_i. We consider four possible group selection strategies:

— Random selection: the item selects its group at random (regardless of its probability vector). As this selection strategy does not use any useful information collected from the search history, it is expected that this strategy would not perform well.

— Greedy selection: the item always selects the group g_j such that the associated probability p_{ij} has the maximum value. This strategy is intuitively reasonable, but may cause the algorithm to be trapped rapidly.

— Roulette wheel selection: the item selects its group based on its probability vector and the chance for the item to select group g_j is proportional to the probability p_{ij}. Thus a group with a large (small) probability has more (less) chance to be selected.

— Hybrid selection: this strategy combines the random selection and greedy selection strategies in a probabilistic way; with a noise probability ω, random selection is applied; with probability $1 - \omega$, greedy selection is applied.

As we show in Section 3.5.2, the group selection strategy greatly affects the performance of the PLS approach. After experimenting the above strategies, we adopted the hybrid selection strategy which combines randomness and greediness which are controlled by the noise probability ω. The purpose of selecting a group with maximum probability (greedy selection) is to make an attempt to correctly select the group for an item that is most often falsified at a local optimum. Selecting such a group for this item may help the search to escape from the current trap. On the other hand, using the noise probability has the advantage of flexibility by switching back and forth between greediness and randomness. Also, this allows the algorithm to occasionally move away from being too greedy. This hybrid group selection strategy proves to be better than the roulette wheel selection strategy, as confirmed by the experiments of Section 3.5.2.

3.2.3 Optimization procedure

Even if any optimization procedure can be used to improve a given starting grouping solution, for the reason of simplicity, we employ a simple and fast *Descent-Based Local Search (DB-LS)* procedure in this chapter. To explore the search space, DB-LS iteratively makes transitions from the incumbent solution to a neighboring solution according to a given neighborhood relation such that each transition leads to a better solution. This iterative improvement process continues until no improved solution exists in the neighborhood in which case the incumbent solution corresponds to a local optimum with respect to the neighborhood.

Let Ω denote the search space of the given grouping problem. Let $N : \Omega \rightarrow 2^{\Omega}$ be the neighborhood relation which associates to each solution $S \in \Omega$ a subset of solutions $N(S) \subset \Omega$ (i.e., $N(S)$ is the set of neighboring solutions of S). Typically, given a solution S, a neighboring solution can be obtained by moving an item of S from its current group to another group. Let $f : \Omega \rightarrow \mathbb{R}$ be the evaluation (or cost)

function which measures the quality or cost of each grouping solution. The pseudo code of Algorithm 2 displays the general DB-LS procedure.

Algorithm 2: Pseudo-code of **DB-LS** procedure

Input: An initial candidate solution S
Output: The local optimum solution attained S^*

```
begin
    S* ← S
    while S is not a local optimum do
        choose a neighbor S' of S such that f(S') < f(S)
        S ← S'
    S* ← S
```

DB-LS can find a local optimum quickly. However, the local optimal solution discovered is generally of poor quality. It is fully possible to improve the performance of PLS by replacing the DB-LS with a more powerful improvement algorithm, such as tabu search.

3.2.4 Probability updating procedure

In PLS, we make the assumption that, if the item stays in its original group after the DB-LS, then the item has selected the right group in the original solution, otherwise its new group in the improved solution would be the right group. This assumption can be considered to be reasonable because the DB-LS procedure is driven by its cost function and each transition from the current solution to a new (neighboring) solution is performed only when the transition leads to an improvement. The problem of selecting the most appropriate group for each item is a dynamic process. Through the interactions with the unknown environment, PLS evolves and gradually finds the optimal or a suboptimal solution of the problem.

At iteration t, we firstly generate a grouping solution S based on the current probability matrix P (see Section 3.2.1). In other words, each item selects one suitable group from the k available groups based on its probability vector (with the group selection strategy of Section 3.2.2). Then solution S is improved by the DB-LS procedure, leading to an improved solution S'. Now, for each item v_i, we compare its groups in S and S'. If the item stays in its original group (say g_u), we reward the selected group g_u (called correct group) and update its probability vector p_i at next iteration $t+1$ according to Equation (3.1):

$$p_{ij}(t+1) = \begin{cases} \alpha + (1-\alpha)p_{ij}(t) & j = u \\ (1-\alpha)p_{ij}(t) & \text{otherwise.} \end{cases} \tag{3.1}$$

where α $(0 < \alpha < 1)$ is a reward factor. When item v_i moves from its original group g_u of solution S to a new group (say $g_v, v \neq u$) of the improved solution S', we penalize the discarded group g_u (called *incorrect group*), compensate the new group g_v (called *expected group*) and finally update its probability vector p_i at next iteration $t+1$ according to Equation (3.2):

$$p_{ij}(t+1) = \begin{cases} (1-\gamma)(1-\beta)p_{ij}(t) & j = u \\ \gamma + (1-\gamma)\frac{\beta}{k-1} + (1-\gamma)(1-\beta)p_{ij}(t) & j = v \\ (1-\gamma)\frac{\beta}{k-1} + (1-\gamma)(1-\beta)p_{ij}(t) & \text{otherwise.} \end{cases} \tag{3.2}$$

where β $(0 < \beta < 1)$ and γ $(0 < \gamma < 1)$ are a penalization factor and compensation factor respectively. The update of the complete probability matrix P is bounded by $O(n \times k)$ in terms of time complexity.

It is necessary to note that our learning scheme is different from general reinforcement learning schemes such as linear reward-penalty, linear reward-inaction and linear reward-ϵ-penalty [Sutton and Barto, 1998]. The philosophy of these schemes is to increase the probability of selecting an action in the event of success and decrease it when receiving a failed signal. Unlike these general schemes, our learning scheme not only rewards the correct group and penalizes the incorrect group, but also compensates the expected group.

3.2.5 Probability smoothing procedure

The intuition behind the probability smoothing technique is that old decisions that were made long time ago are no longer helpful and may mislead the current search. Therefore, these aged decisions should be considered less important than the recent ones. In addition, all items are required to correctly select their suitable groups in order to produce a legal grouping solution. It is not enough that only a part of items can correctly select their groups. Based on these two reasons, we introduce a probability smoothing technique to reduce the group probabilities periodically.

Our probability smoothing strategy is inspired by forgetting mechanisms in smoothing techniques in clause weighting local search algorithms for *Satisfiability (SAT)* [Hutter *et al.*, 2002; Ishtaiwi *et al.*, 2005]. Based on the way that weights are smoothed or forgotten, there are four available forgetting or smoothing techniques for the *Minimum Vertex Cover (MVC)* and SAT:

— Decrease one from all clause weights which are greater than one such as PAWS [Thornton *et al.*, 2004].

— Pull all clause weights to their mean value using the formula $w_i = \rho \cdot w_i + (1 - \rho) \cdot \overline{w_i}$ like ESG [Schuurmans *et al.*, 2001] and SAPS [Hutter *et al.*, 2002].

— Transfer weights from neighboring satisfied clauses to unsatisfied ones like DDWF [Ishtaiwi *et al.*, 2005].

— Reduce all edge weights using the formula $w_i = \lfloor \rho \cdot w_i \rfloor$ when the average weight achieves a threshold like NuMVC [Cai *et al.*, 2013].

The probability smoothing strategy adopted in our PLS approach works as follows (see Algorithm 3). For an item, each possible group is associated with a value between 0 and 1 as its probability, and each group probability is initialized as $1/k$. At each iteration, we adjust the probability vector based on the obtained feedback information (i.e., reward, penalize or compensate a group). Once the probability of a group in a probability vector achieves a given threshold (i.e., p_0), it is reduced by multiplying a smoothing coefficient (i.e., $\rho < 1$) to forget some earlier decisions. It is obvious that the smoothing technique used in PLS is different from the above-mentioned four techniques. To the best of our knowledge, this is the first time a smoothing technique is introduced into local search algorithms for grouping problems.

Algorithm 3: Pseudo-code of the probability smoothing procedure

Input: Probability matrix P at iteration t, smoothing probability p_0 and smoothing coefficient ρ
Output: Probability matrix P at iteration $t + 1$

```
1 begin
2     for i = 1 to n do
3         p_iw ← max{p_ij, j = 1, 2, ..., k}
4         if p_iw > p_0 then
5             for j = 1 to k do
6                 if j = w then
7                     p_ij(t + 1) ← ρ · p_ij(t)
8                 else
9                     p_ij(t + 1) ← (1−ρ)/(k−1) · p_iw(t) + p_ij(t)
```

3.3 PLS applied to graph coloring problem

This section presents an application of the proposed PLS method to the well-known *Graph Coloring Problem (GCP)* which is a typical grouping problem. Following many coloring algorithms [Galinier *et al.*,

2013; Galinier and Hao, 1999; Galinier and Hertz, 2006; Hertz and de Werra, 1987; Lü and Hao, 2010; Malaguti *et al.*, 2008; Porumbel *et al.*, 2010a], we approximate GCP by solving a series of k-coloring problems. For a given graph and a fixed number of k colors, we try to find a legal k-coloring. If a legal k-coloring is found, we set $k = k - 1$ and try to solve the new k-coloring problem. We repeat this process until no legal k-coloring can be found, in which case we return $k + 1$ (for which a legal coloring has been found) as an approximation (upper bound) of the chromatic number of the given graph.

3.3.1 Related work

Given the $\mathcal{NP}$-hardness of GCP, exact algorithms are usually effective only for solving small or easy graphs. In fact, some graphs with as few as 150 vertices cannot be solved optimally by any exact algorithm [Malaguti *et al.*, 2011; Zhou *et al.*, 2014]. To deal with large and difficult graphs, heuristic algorithms are preferred to solve the problem approximately. Comprehensive reviews of the graph coloring algorithms can be found in [Galinier *et al.*, 2013; Galinier and Hertz, 2006; Malaguti and Toth, 2010]. In what follows, we focus on some representative heuristic-based coloring algorithms.

— **Constructive approaches** generally construct the color groups by iteratively adding a vertex at one time to a color group until a complete coloring is reached. At each iteration, there are two steps: the next vertex to be colored is chosen at first, and then this vertex is assigned to a color group. DSATUR [Brélaz, 1979] and RLF [Leighton, 1979] are two well-known greedy algorithms which employ refined rules to dynamically determine the next vertex to color. These greedy heuristic algorithms are usually fast, but they tend to need much more colors than the chromatic number to color a graph. Consequently, they are often used as initialization procedures in hybrid algorithms.

— **Local search approaches** start from an initial solution and try to improve the coloring by performing local changes. In particular, tabu search is known as one of the most popular local search method for GCP [Hertz and de Werra, 1987]. It is often used as a subroutine in hybrid algorithms, such as the hybrid evolutionary algorithm [Fleurent and Ferland, 1996; Galinier and Hao, 1999; Lü and Hao, 2010; Moalic and Gondran, 2015; Porumbel *et al.*, 2010a]. However, local search algorithms are often substantially limited by the fact that they do not exploit enough global information (e.g., solution symmetry), and cannot compete with hybrid population-based algorithms. A thorough survey of local search algorithms for graph coloring can be found in [Galinier and Hertz, 2006].

— **Population-based hybrid approaches** work with multiple solutions that can be manipulated by some selection and recombination operators. To maintain the population diversity which is critical to avoid a premature convergence, population-based algorithms usually integrate dedicated diversity preservation mechanisms which require the computation of a suitable distance metric between solutions [Hao and Wu, 2012]. For example, hybrid algorithms [Lü and Hao, 2010; Moalic and Gondran, 2015; Porumbel *et al.*, 2010a; Titiloye and Crispin, 2011] are among the most effective approaches for graph coloring, which have reported the best solutions on most of the difficult DIMACS instances. Nevertheless, population-based hybrid algorithms have the disadvantage of being much more complex in design and more sophisticated in implementation. Moreover, their success greatly depends on the use of a meaningful recombination operator, an effective local optimization procedure and a mechanism for maintaining population diversity.

— "**Reduce and solve**" **approaches** combines a preprocessing phase and a coloring phase. The preprocessing phase identifies and extracts some vertices (typically independent sets) from the original graph to obtain a reduced graph, while the subsequent coloring phase determines a proper coloring for the reduced graph. Empirical results showed that "reduce-and-solve" approaches achieve a remarkable performance on some large and very large graphs [Hao and Wu, 2012; Wu and Hao, 2013]. "Reduce and solve" approaches are designed for solving large graphs and are less suitable for small and medium-scale graphs. Moreover, their success depends on the vertices extraction procedure and the underlying coloring algorithm.

— **Other approaches** which cannot be classified into the previous categories include for instance a method that encodes GCP as a boolean satisfiability problem [Bouhmala and Granmo, 2008], a modified cuckoo algorithm [Mahmoudi and Lotfi, 2015], a grouping hyper-heuristic algorithm [Elhag and Özcan, 2015] and a multi-agent based distributed algorithm [Sghir *et al.*, 2015].

3.3.2 PLSCOL for GCP

We first define the search space and the evaluation function used by the PLSCOL algorithm. For a given graph $G = (V, E)$ with k available colors, the search space Ω contains all possible (both legal or illegal) k-colorings (candidate solutions). A candidate solution in Ω can be represented by $S = \{g_1, g_2, \ldots, g_k\}$ such that g_i is the group of vertices receiving color i. The evaluation function $f(S)$ is used to count the conflicting edges induced by S.

$$f(S) = \sum_{\{u,v\} \in E} \delta(u, v) \tag{3.3}$$

where $\delta(u, v) = 1$, if $u \in g_i, v \in g_j$ and $i = j$, and otherwise $\delta(u, v) = 0$. Thus, a candidate solution S is a legal k-coloring when $f(S) = 0$. The objective of PLSCOL is to minimize f, i.e., the number of conflicting edges to find a legal k-coloring in the search space.

Algorithm 4 presents the PLSCOL algorithm. Initially, by setting the probability $p_{ij} = 1/k, i \in \{1, \ldots, n\}, j \in \{1, \ldots, k\}$, each group is selected uniformly by each item (line 3). A starting solution S is produced by the hybrid selection strategy based on the current probability matrix P (line 5). The tabu search procedure is used to improve the starting solution S to a new solution S' (line 6). The *GroupMatching* procedure is then applied to find a maximum weight matching between the starting solution S and its improved solution S' (line 10), followed by the *ProbabilityUpdating* procedure (to update the probability matrix P according to the matching results, line 11), and the *ProbabilitySmoothing* procedure (to forget some earlier decisions, line 12).

Algorithm 4: Pseudo-code of **PLSCOL** for k-coloring

Input: Instance G = (V, E) and number of available colors k.
Output: the best k-coloring S^* found so far

```
1  begin
2      // initialize the probability matrix P
3      P = [p_ij = 1/k]_{i=1,2,...,n, j=1,2,...,k}
4      while stopping condition not reached do
5          S ← GroupSelection(P);                    /* generate a starting solution */
6          S' ← TabuSearch(S);                       /* find a local optimal solution */
7          // update the best solution found
8          if f(S') < f(S*) then
9              S* ← S'
10         feedback ← GroupMatching(S, S');          /* make a maximum matching */
11         P ← ProbabilityUpdating(feedback, P);     /* learn a new probability matrix */
12         P ← ProbabilitySmoothing(P);              /* conduct a probability smoothing */
```

Compared with the general PLS method (see Algorithm 1), the PLSCOL algorithm introduces two improvements. Considering the specific feature of GCP where color groups are interchangeable, we call the group matching procedure to find a group-to-group correspondence between a starting solution and its improved solution. Also, instead of using the descent-based local search to search for a legal coloring, we adopt a more elaborated coloring algorithm (i.e., the well-known tabu search procedure). In the following, we will explain these two improvements respectively.

Tabu Search Procedure

The used optimization procedure is an improved version of the tabu search coloring procedure originally proposed in [Hertz and de Werra, 1987]. As described in [Dorne and Hao, 1998; Galinier and Hao, 1999], the improved tabu coloring procedure (TabuCol) integrates three enhancements.

1. Instead of considering a *sample* of neighboring colorings, it considers all neighboring colorings induced by the set of conflicting vertices.
2. It adopts a dynamic tabu tenure which is defined as a function of the number of conflicting vertices and a random number.
3. It employs dedicated data structures to ensure a fast and incremental evaluation of neighboring solutions.

Given a conflicting k-coloring S, the "one-move" neighborhood $N(S)$ consists of all solutions produced by moving a conflicting vertex v_i from its original group g_u to a new group g_v $(u \neq v)$. TabuCol picks at each iteration a best neighbor $\hat{S} \in N(S)$ according to the evaluation function f given by Equation (3.3) such that either $\hat{S}$ is a best solution not forbidden by the tabu list or is better than the best solution found so far S^* (i.e., $f(\hat{S}) < f(S^*)$). Ties are broken at random. Once a move is made, e.g., a conflicting vertex v_i is moved from group g_u to group g_v, the vertex v_i is forbidden to go back to group g_u in the following l iterations (l is called tabu tenure). The tabu tenure l is dynamically determined by $l = \mu \times f(S) + \textit{Random(A)}$, where *Random(A)* returns a random integer in $\{0, \ldots, A-1\}$ [Dorne and Hao, 1998; Galinier and Hao, 1999]. In our algorithm, we set $\mu = 1.2$ and $A = 10$. The TabuCol process stops when the number of iterations without improving S^* reaches a predefined value I_{max}, which is set to be 10^5.

For an efficient implementation of the TabuCol procedure, we use an incremental technique [Fleurent and Ferland, 1996; Galinier and Hao, 1999] to maintain and update the move gains $\Delta f = \gamma_{i,u} - \gamma_{i,v}$ for each possible candidate move (i.e., displacing vertex v_i from group g_v to group g_u).

Group Matching Procedure

As a particular grouping problem, GCP is characterized by the fact that the numberings of the groups in a solution are irrelevant. For instance, for a graph with five vertices {a,b,c,d,e}, suppose that we are given a solution (coloring) that assigns a color to {a,b,c} and another color to {d,e}. It should be clear that what really characterizes this coloring is not the naming/numbering of the colors used, but is the fact that some vertices belong to the same group and some other vertices cannot belong to the same group due to the coloring constraints. As such, we can name {a,b,c} by 'group 1' and {d,e} by 'group 2' or reversely name {a,b,c} by 'group 2' and {d,e} by 'group 1', these two different group namings represent the same coloring. This symmetric feature of colorings is known to represent a real difficulty for many coloring algorithms [Galinier and Hao, 1999; Porumbel *et al.*, 2010b].

In the generic PLS approach, the learning phase was based on a direct comparison of the groups between the starting solution and its improved solution, by checking for each vertex its starting group number and the new group number. This approach has the advantage of easy implementation and remains meaningful within the PLS method. Indeed, PLS does not assume any particular feature (e.g., symmetry of solutions in the case of GCP) of the given grouping problem. Moreover, when a strict descent-based local search is used to obtain an improved solution (i.e., local optimum), only a small portion of the vertices change their groups. It suffices to compare the groups of the two solutions to identify the vertices that changed their group. However, the situation is considerably different when tabu search (or another optimization algorithm) is used to obtain an improved coloring, since many vertices can change their groups during the search process. This difficulty is further accentuated for GCP due to its symmetric feature of colorings, making it irrelevant to compare the group numbers between the starting and improved solutions.

To illustrate this point, consider the example of Figure 3.3. From the same starting 8-coloring $S = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8\}$ with $f(S) = 568$, the descent-based local search ends with an improved (but always illegal) 8-coloring S'_{ds} with $f(S'_{ds}) = 10$, while the tabu search procedure successfully obtains an

improved 8-coloring S'_{ts} with $f(S'_{ts}) = 0$. In this figure, the new groups in the two improved solutions are marked by red color and underlined.

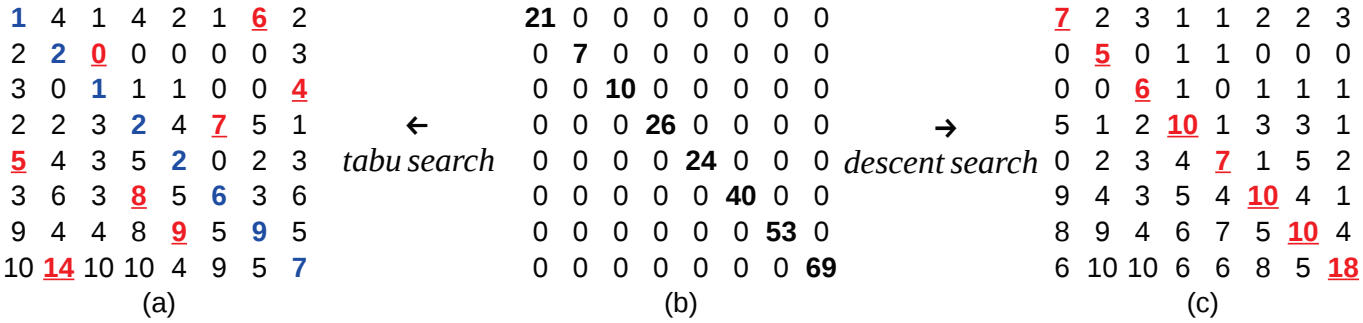

Figure 3.3: Distribution of vertices of solutions on instance DSJC250.1. The value on i-th row and j-th column represents the number of vertices whose color have changed from color i to color j. (a) An improved solution S'_{ts} (with $f(S'_{ts}) = 0$) obtained by tabu search. (b) A starting solution S (with $f(S) = 568$). (c) An improved solution S'_{ds} with $f(S'_{ds}) = 10$ obtained by descent-based local search.

By comparing the distributions of vertices of the two improved solutions from descent-based search and tabu search, we observe a clear one-to-one correspondence between the starting solution S and its improved solution S'_{ds} obtained by descent-based search, i.e., $g_1 \leftrightarrow g'_1, g_2 \leftrightarrow g'_2, g_3 \leftrightarrow g'_3, g_4 \leftrightarrow g'_4, g_5 \leftrightarrow g'_5, g_6 \leftrightarrow g'_6, g_7 \leftrightarrow g'_7, g_8 \leftrightarrow g'_8$. However, it is difficult to find such a relationship between S and S'_{ts} obtained by tabu search. Indeed, from the same starting solution, much more changes have been made by tabu search. For example, there are $|g_1| = 21$ vertices colored by color 1 (g_1) in the starting solution S; after improving to S'_{ds} by descent-based search, $|g'_1| = 7$ vertices keep their original color, the remaining 2, 3, 1, 1, 2, 2, 3 vertices respectively changed to new colors g'_2, g'_3, g'_4, g'_5, g'_6, g'_7, g'_8. When S is improved to S'_{ts} by tabu search, only $|g'_1| = 1$ vertex of these 21 vertices keeps its original color, the remaining 4, 1, 4, 2, 1, 6, 2 vertices have moved to color groups g'_2, g'_3, g'_4, g'_5, g'_6, g'_7, g'_8.

Now if we examine the groups of vertices in S and S'_{ts} regardless of the numbering of the groups, we can identify the following group-to-group correspondence between S and S'_{ts}: $g_1 \leftrightarrow g'_7, g_2 \leftrightarrow g'_3, g_3 \leftrightarrow g'_8, g_4 \leftrightarrow g'_6, g_5 \leftrightarrow g'_1, g_6 \leftrightarrow g'_4, g_7 \leftrightarrow g'_5, g_8 \leftrightarrow g'_2$. Indeed, this correspondence can be achieved by finding a maximum weight matching between the two compared solutions as follows.

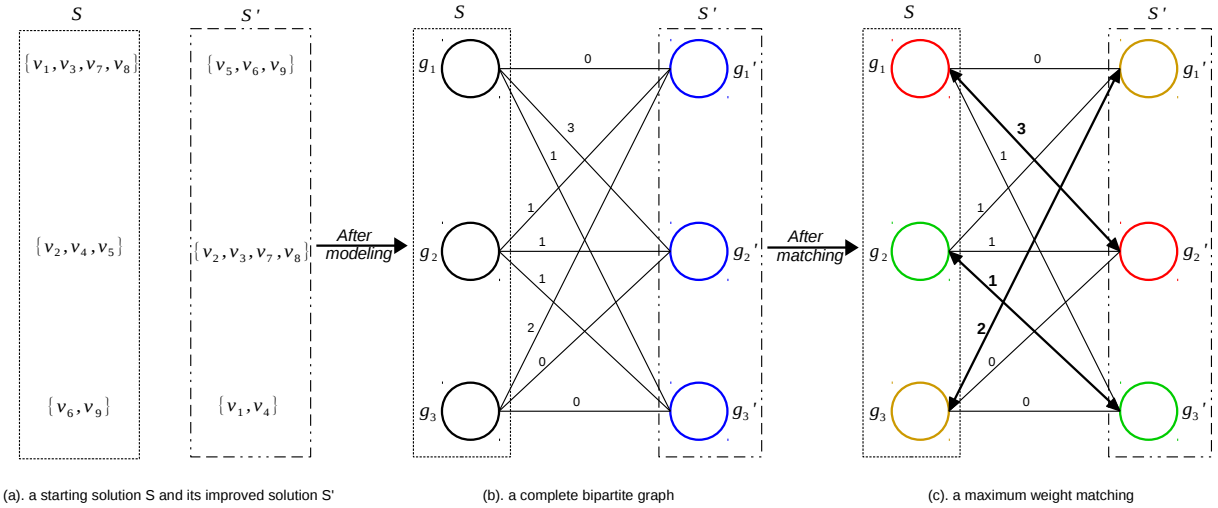

Figure 3.4: (a) A starting solutions S and its improved solution S'. (b) A complete bipartite graph with the weights between two groups $\omega_{g_i,g'_j} = |g_i \cap g'_j|$. (c) The corresponding maximum weight complete matching with the maximum weight of 6.

Specifically, to identify such a relationship between a starting solution S and its improved solution S',

we sequentially match each group of S with each group of S'. Then we build a complete bipartite graph $G = (V_1, V_2, E)$, where V_1 and V_2 represent respectively the k groups of solution S and S'. Each edge $(g_i, g'_j) \in E$ is associated with a weight $w_{g_i,g'_j} = |g_i \cap g'_j|$, which is defined as the number of common vertices in group g_i of S and g'_j of S'. Based on the bipartite graph, we can find a maximum weight matching with the well-known *Hungarian algorithm* [Kuhn, 1955] in $O(k^3)$. Figure 3.4 shows an illustrative example of matching two solutions $S = \{(v_1, v_3, v_7, v_8), (v_2, v_4, v_5), (v_6, v_9)\}$ and $S' = \{(v_5, v_6, v_9), (v_2, v_3, v_7, v_8), (v_1, v_4)\}$. The group matching procedure finds the following one-to-one group relation between these two solutions: $g_1 \leftrightarrow g'_2$, $g_2 \leftrightarrow g'_3$ and $g_3 \leftrightarrow g'_1$.

3.4 Computational results

3.4.1 Benchmark instances

This section is dedicated to an extensive experimental evaluation of the PLSCOL algorithm using the well-known DIMACS challenge benchmark instances [2]. These instances have been widely used in the literature for assessing the performances of graph coloring algorithms. They belong to the following six types:

1. **Standard random graphs** denoted as "DSJCn.x", where n is the number of vertices and 0.x is the density. They are generated in such a way that the probability of an edge being present between two given vertices equals the density.
2. **Geometric random graphs** named as "DSJRn.x " and "Rn.x". They are produced by choosing randomly n points in the unit square, which define the vertices of the graph, by joining two vertices with an edge, if the two related points at a distance less than x from each other. Graphs with letter c denotes the complement of a geometric random graph.
3. **Leighton graphs** named as "len_χx", are of density below 0.25, where n is the number of vertices, χ is the chromatic number, and $x \in \{a, b, c, d\}$ is a letter to indicate different graphs with the similar parameter settings.
4. **"Quasi-random" flat graphs** denoted as "flatn_χ_δ", where n is the number of vertices, χ is the chromatic number, and δ is a flatness parameter giving the maximal allowed difference between the degrees of two vertices.
5. **Scheduling graphs** include two scheduling graphs school1 and school1_nsh.
6. **Latin square graph** represents a latin square graph latin_square_10.

These instances can be roughly divided into two categories: easy graphs and difficult graphs according to the classification in [Galinier *et al.*, 2013; Galinier *et al.*, 2008]. Let k^* be $\chi(G)$ (if known) or the smallest number of colors for which a legal coloring has been found by at least one coloring algorithm. Then easy graphs G are those that can be colored with k^* colors by a basic coloring algorithm like DSATUR [Brélaz, 1979] (thus by numerous algorithms). Otherwise, if a k^*-coloring can only be achieved by a few advanced coloring algorithms, the graph will be qualified as difficult. Since easy graphs do not represent any challenge for PLSCOL (and many reference algorithms), we mainly focus our tests on difficult graphs.

3.4.2 Experimental settings

The PLSCOL algorithm [3] was implemented in C++, and complied using GNU g++ on an Intel E5-2760 with 2.8 GHz and 2GB RAM under Linux operating system. For our experiments, each instance was solved 10 times independently. Each execution was terminated when the maximum allowable running time of 5

2. ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/
3. The code of the PLSCOL algorithm is available upon request.

CPU hour is reached or a legal k-coloring is found. We used the same parameters setting to solve all tested instances except for the penalization factor β. Parameter β is sensitive to the structures of GCP instances, and it mainly affects the running time to find the best k-coloring (see Section 3.5.5 for an analysis of this parameter). Accordingly, we adopted values for β from $[0.05, 0.45]$ to obtain the reported results on the set of 20 difficult instances. Table 3.1 gives the description and setting of each parameter.

Table 3.1: Parameter settings of PLSCOL algorithm.

Parameter	description	value
ω	noise probability	0.200
α	reward factor for correct group	0.100
β	penalization factor for incorrect group	[0.05, 0.45]
γ	compensation factor for expected group	0.300
ρ	smoothing coefficient	0.500
p_0	smoothing threshold	0.995

3.4.3 Comparison with its simple version PLS

We first assess the performance of the proposed PLSCOL algorithm with the generic PLS (i.e., descent-based search and no group matching procedure). This experiment aims to demonstrate the usefulness of the two enhancements, i.e., the group matching procedure and the tabu coloring procedure. Both algorithms were run 10 times on each instance, each run being given a CPU time of 5 hours.

The comparative results between PLSCOL and PLS are summarized in Table 3.2. Columns 1-2 indicate the instance name, its chromatic number χ when it is known or the best-known result reported in the literature (k^*). For each algorithm, we report the smallest number of colors (k) used by the algorithm, the rate of successful runs out of 10 runs ($\#succ$), the average number of moves ($\#iter$) and the average running time in seconds ($time(s)$). Better results (with a smaller k) between the two algorithms are indicated in bold. When both algorithms achieve the same result in terms of number of colors used, we underline the better results in terms of number of iterations.

From Table 3.2, we observe that PLSCOL significantly outperforms PLS, achieving better solutions for 13 out of 20 instances and equal solutions on the remaining instances. Among the 7 instances where both algorithms reach the same results, PLSCOL performs better in terms of success rate and average running time in 5 cases. In summary, this study indicates that the group matching procedure and the tabu coloring procedure of PLSCOL significantly boosts the performance of the generic PLS approach for solving the graph coloring problem.

3.4.4 Comparison with other state-of-the-art algorithms

This section shows a comparison of PLSCOL with 10 state-of-the-art coloring algorithms in the literature. For this comparison, we focus on the criterion of solution quality in terms of the smallest number of colors used to find a legal coloring. In fact, despite the extremely vast literature on graph coloring, there is no uniform experimental condition to assess a coloring algorithm. This is mainly because the difficult DIMACS instances are really challenging. Indeed, the best-known solutions for these instances can be found only by few most powerful algorithms which were implemented with different programming languages and executed under various computing platforms and different stopping conditions (maximum allowed generations, maximum allowed fitness evaluations, maximum allowed iterations or still maximum allowed cut off time). In any case, a run time of several hours or even several days was typically applied (see e.g., [Lü and Hao, 2010; Malaguti *et al.*, 2008; Moalic and Gondran, 2015; Porumbel *et al.*, 2010a; Titiloye and Crispin, 2011; Titiloye and Crispin, 2012]). For this reason, it would be quite difficult to compare CPU times of the compared algorithms. Following the common practice of reporting comparative

Table 3.2: Comparative results of PLSCOL and PLS on the difficult DIMACS graphs.

		PLSCOL				PLS			
Instance	χ/k^*	k	#succ	#iter	time(s)	k	#succ	#iter	time(s)
DSJC250.5	?/28	**28**	10/10	4.0×10^5	4	29	10/10	2.8×10^6	91
DSJC500.1	?/12	**12**	07/10	7.5×10^6	43	13	10/10	5.6×10^5	17
DSJC500.5	?/47	**48**	03/10	7.9×10^7	1786	50	05/10	1.9×10^7	1714
DSJC500.9	?/126	**126**	10/10	2.4×10^7	747	129	03/10	5.0×10^8	9859
DSJC1000.1	?/20	**20**	01/10	2.9×10^8	3694	21	10/10	1.4×10^7	1223
DSJC1000.5	?/83	**87**	10/10	2.7×10^7	1419	94	07/10	3.8×10^8	9455
DSJC1000.9	?/222	**223**	05/10	3.1×10^8	12094	233	03/10	2.5×10^8	12602
DSJR500.1c	?/85	**85**	10/10	3.2×10^7	386	**85**	01/10	4.6×10^6	700
DSJR500.5	?/122	**126**	08/10	7.3×10^7	1860	**126**	01/10	1.8×10^8	3428
le450_15c	15/15	**15**	07/10	2.8×10^8	1718	**15**	07/10	9.5×10^6	308
le450_15d	15/15	**15**	03/10	2.8×10^8	2499	**15**	04/10	5.8×10^8	211
le450_25c	25/25	**25**	10/10	2.0×10^8	1296	26	07/10	4.7×10^6	181
le450_25d	25/25	**25**	10/10	2.2×10^8	1704	26	04/10	1.1×10^7	438
flat300_26_0	26/26	**26**	10/10	4.9×10^6	195	**26**	09/10	9.4×10^6	450
flat300_28_0	28/28	**30**	10/10	1.5×10^7	233	32	09/10	4.4×10^6	173
flat1000_76_0	76/81	**86**	01/10	1.1×10^8	5301	89	01/10	4.5×10^7	11609
R250.5	?/65	**66**	10/10	1.1×10^8	705	**66**	04/10	7.0×10^8	6603
R1000.1c	?/98	**98**	10/10	9.1×10^6	256	100	02/10	5.3×10^8	16139
R1000.5	?/234	**254**	04/10	3.7×10^7	7818	261	02/10	3.7×10^7	9015
latin_square_10	?/97	**99**	08/10	6.7×10^7	2005	**99**	02/10	3.3×10^7	12947

results in the coloring literature, we use the best solution (i.e., the smallest number of used colors) for this comparative study. Since the experimental conditions of the compared algorithms are not equivalent, the comparison is just intended to show the relative performance of the proposed algorithm, which should be interpreted with caution.

The reference algorithms and their corresponding experimental environment are displayed as follows:

1. Iterated local search algorithm (IGrAl) [Caramia and Dell'Olmo, 2008] (a 2.8 GHz Pentium 4 processor and a cut off time of 1 hour).
2. Variable space search algorithm (VSS) [Hertz *et al.*, 2008] (a 2.0 GHz Pentium 4 processor and a cut off time of 10 hours).
3. Local search algorithm using partial solutions (Partial) [Blöchliger and Zufferey, 2008] (a 2.0 GHz Pentium 4 and a time limit of 10 hours together with a limit of $2*10^9$ iterations without improvement).
4. Hybrid evolutionary algorithm (HEA) [Galinier and Hao, 1999] (the processor used is not available for this oldest algorithm and the results were obtained with different parameter settings).
5. Adaptive memory algorithm (AMA) [Galinier *et al.*, 2008] (the processor applied is not available and the results were obtained with different parameter settings).
6. Two-phase evolutionary algorithm (MMT) [Malaguti *et al.*, 2008] (a 2.4 GHz Pentium processor and a cut off time of 6000 or 40000 seconds).
7. Evolutionary algorithm with diversity guarantee (Evo-Div) [Porumbel *et al.*, 2010a] (a 2.8 GHz Xeon processor and a cut off time of 12 hours).
8. Memetic algorithm (MACOL or MA) [Lü and Hao, 2010] (a 3.4 GHz processor and a cut off time of 5 hours).
9. Quantum annealing algorithm (QACOL or QA) [Titiloye and Crispin, 2011; Titiloye and Crispin, 2012] (a 3.0 GHz Intel processor and a cut off time of 5 hours).
10. The newest memetic algorithm (HEAD) [Moalic and Gondran, 2015] (a 3.1 GHz Intel Xeon processor with 4 cores and a cut off time of 0.5 hour).

These reference algorithms belong to two categories: local search algorithms and population-based algorithms. Table 3.3 presents the comparative results of PLSCOL with these state-of-the-art algorithms. In this table, column 2 recalls the chromatic number or best-known value χ/k^* in the literature. Column 3 shows the best results of PLSCOL (k_{best}). The following columns are the best results obtained by the reference algorithms, which are extracted from the literature.

Table 3.3: Comparative results of PLSCOL and 10 state-of-the-art algorithms on the difficult DIMACS graphs.

		local search algorithms				population-based algorithms						
		PLSCOL	IGrAl	VSS	Partial	HEA	AMA	MMT	Evo-Div	MA	QA	HEAD
Instance	χ/k^*	k	2008	2008	2008	1999	2008	2008	2010	2010	2011	2015
DSJC250.5	?/28	**28**	29	*	*	*	**28**	**28**	*	**28**	**28**	**28**
DSJC500.1	?/12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
DSJC500.5	?/47	48	50	48	48	48	48	48	48	48	48	**47**
DSJC500.9	?/126	**126**	129	**126**	127	**126**	**126**	127	**126**	**126**	**126**	**126**
DSJC1000.1	?/20	**20**	22	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
DSJC1000.5	?/82	87	94	86	89	83	84	84	83	83	83	**82**
DSJC1000.9	?/222	223	239	224	226	224	224	225	223	223	**222**	**222**
DSJR500.1c	?/85	**85**	**85**	**85**	**85**	*	86	**85**	**85**	**85**	**85**	**85**
DSJR500.5	?/122	126	126	125	125	*	127	**122**	**122**	**122**	**122**	*
le450_15c	15/15	**15**	16	**15**	**15**	**15**	**15**	**15**	*	**15**	**15**	*
le450_15d	15/15	**15**	16	**15**	**15**	**15**	**15**	**15**	*	**15**	**15**	*
le450_25c	25/25	**25**	27	**25**	25	26	26	**25**	**25**	**25**	**25**	**25**
le450_25d	25/25	**25**	27	**25**	25	26	26	**25**	**25**	**25**	**25**	**25**
flat300_26_0	26/26	**26**	*	*	*	*	**26**	**26**	*	**26**	*	*
flat300_28_0	28/28	30	*	**28**	**28**	31	31	31	31	29	31	31
flat1000_76_0	76/81	86	*	85	87	83	84	83	82	82	82	**81**
R250.5	?/65	66	*	*	66	*	*	**65**	**65**	**65**	**65**	**65**
R1000.1c	?/98	**98**	*	*	*	*	*	**98**	**98**	**98**	**98**	**98**
R1000.5	?/234	254	*	*	248	*	*	**234**	238	245	238	245
latin_square_10	?/97	99	100	*	*	*	104	101	100	99	**98**	*

Table 3.4: Comparative results of PLSCOL and other local search algorithms to find optimal 25-coloring on instance le450_25c and le450_25d.

	le450_25c			le450_25d		
Instance	#succ	#iter	time(h)	#succ	#iter	time(h)
PLSCOL	10/10	2.0×10^8	< 1	10/10	2.2×10^8	< 1
VSS	9/10	1.6×10^9	5	6/10	2.2×10^9	6
Partial	2/5	2.0×10^9	10	3/5	2.0×10^9	10
TS-Div	4/10	7.7×10^8	11	2/10	1.2×10^9	19
TS-Int	10/10	3.4×10^9	10	10/10	6.5×10^9	25

As we observe from Table 3.3, PLSCOL competes very favorably with the three reference local search algorithms except for instance flat300_28_0. The best result for this instance was reached only by one algorithm [Blöchliger and Zufferey, 2008]. To the best of our knowledge, for difficult instances le450_25c and le450_25d, PLSCOL is the first local search algorithm which achieves the optimal 25-coloring within 1 hour. A more detailed comparison on these two instances is shown in Table 3.4, including two additional advanced tabu search algorithms (TS-Div and TS-Int [Porumbel *et al.*, 2010b]). Although these three local search algorithms can also obtain the optimal 25-coloring, they need much more running times and more iterations with a lower success rate. It is important to mention that no other local search method was able to find the optimal solution of le450_25c and le450_25d more efficiently than PLSCOL.

When comparing with the seven complex population-based hybrid algorithms, we observe that PLSCOL also remains competitive, and even obtains better results in some cases. In order to facilitate comparisons, we divide these seven reference algorithms into two groups. PLSCOL achieves at least three better solutions compared with the first three algorithms (HEA, AMA and MMT). For example, PLSCOL saves at least one color for DSJC1000.9, flat300_28_0 and latin_square_10. If we compare PLSCOL with the last four algorithms (Evo-Div, MACOL, QACOL and HEA), one finds that PLSCOL is also competitive. For example, though PLSCOL and the last four algorithms reach the same 126-coloring for DSJC500.9, PLSCOL uses less time and iterations to achieve this result (even if timing information was not shown in the table). The same observation applies for other instances, such as DSJR500.1c and R1000.1c. Moreover, compared with Evo-Div and QACOL, PLSCOL also saves one color and finds the 30-coloring for instance flat300_28_0 even if MACOL achieves a better performance for this instance.

On the other hand, PLSCOL performs worse than the population-based algorithms for several instances, such as DSJC1000.5, flat1000_76_0 and R1000.5. However, this is not a real surprise given that the coloring procedure of PLSCOL (i.e., TabuCol) is extremely simple compared to these highly complex population-based algorithms which integrate various search strategies (solution recombination, advanced population management, local optimization). In this sense, the results achieved by PLSCOL are remarkable and demonstrates that its probability learning scheme greatly boosts the performance of the rather simple tabu coloring algorithm.

Table 3.5: Comparative results of PLSCOL and MACOL on easy DIMACS graphs.

		MACOL				PLSCOL			
Instance	χ/k^*	k	#succ	#iter	time(m)	k	#succ	#iter	time(m)
DSJC125.1	?/5	**5**	10/10	1.4×10^5	1	**5**	10/10	4.8×10^3	< 1
DSJC125.5	?/17	**17**	10/10	4.8×10^4	3	**17**	10/10	6.3×10^4	< 1
DSJC125.9	?/44	**44**	10/10	2.4×10^6	4	**44**	10/10	3.0×10^3	< 1
DSJC250.1	?/8	**8**	10/10	6.9×10^5	2	**8**	10/10	6.4×10^5	< 1
DSJC250.9	?/72	**72**	10/10	5.5×10^6	3	**72**	10/10	2.6×10^5	< 1
R125.1	?/5	**5**	10/10	3.7×10^5	2	**5**	10/10	3.6×10^1	< 1
R125.1c	?/46	**46**	10/10	2.8×10^6	5	**46**	10/10	1.6×10^6	< 1
R125.5	?/36	**36**	10/10	3.2×10^4	1	**36**	10/10	8.0×10^5	< 1
R250.1	?/8	**8**	10/10	1.5×10^6	5	**8**	10/10	1.7×10^3	< 1
R250.1c	?/64	**64**	10/10	2.8×10^6	4	**64**	10/10	8.9×10^6	1
DSJR500.1	?/12	**12**	10/10	3.3×10^5	4	**12**	10/10	1.6×10^3	< 1
R1000.1	?/20	**20**	10/10	2.9×10^5	2	**20**	10/10	1.9×10^3	< 1
le450_15a	15/15	**15**	10/10	2.7×10^5	2	**15**	10/10	1.3×10^5	< 1
le450_15b	15/15	**15**	10/10	3.5×10^5	2	**15**	10/10	7.4×10^4	< 1
le450_25a	25/25	**25**	10/10	1.8×10^5	4	**25**	10/10	4.4×10^2	< 1
le450_25b	25/25	**25**	10/10	2.8×10^6	3	**25**	10/10	3.5×10^2	< 1
school1	?/14	**14**	10/10	8.8×10^5	6	**14**	10/10	9.3×10^2	< 1
school1_nsh	?/14	**14**	10/10	7.3×10^5	1	**14**	10/10	5.6×10^3	< 1
flat300_20_0	20/20	**20**	10/10	1.7×10^6	4	**20**	10/10	1.6×10^3	< 1

Finally, we show that for easy instances, PLSCOL can attain the best-known solution more quickly. We illustrate this by comparing PLSCOL with MACOL [Lü and Hao, 2010] which is one of the most powerful population-based coloring algorithms in the literature on 19 easy DIMACS instances (see Table 3.5). The results of MACOL were obtained on a PC with 3.4 GHz CPU and 2G RAM (which is slightly faster than our PC with 2.8 GHz and 2G RAM). Table 3.5 indicates that both PLSCOL and MACOL can easily find the best-known results k^* with a 100% success rate. However, PLSCOL uses less time (at most 1 minute) to find its best results while MACOL needs 1 to 5 minutes to achieve the same results. Moreoevr, PLSCOL needs much less iterations (as underlined) compared to MACOL on all instances except DSJC125.5, R125.5 and R250.1c for which PLSCOL still requires a shorter run time.

3.5 Experimental analysis

In this section, we perform additional experiments to gain some understanding of the proposed PLS approach including the benefit of probability smoothing technique, comparisons among different group selection strategies, and the benefits of the probability learning scheme and group matching procedure. We also analyze the impact of the penalization factor β over the performance of PLSCOL.

3.5.1 Benefit of the probability smoothing technique

To study the effectiveness of the probability smoothing technique used in PLS (with descent-based local search and no group matching procedure), we compare PLS with its alternative algorithm PLS_1, which is obtained from PLS by adjusting the probability updating scheme. More specifically, PLS_1 works in the same way as PLS, but it does not use the probability smoothing strategy, that is, line 11 in Algorithm 1 is removed. For this experiment, by following [Galinier and Hao, 1999], we use *running profiles* to observe the change of evaluation function f over the number of iterations. Running profiles provide interesting information about the convergence of the studied algorithms.

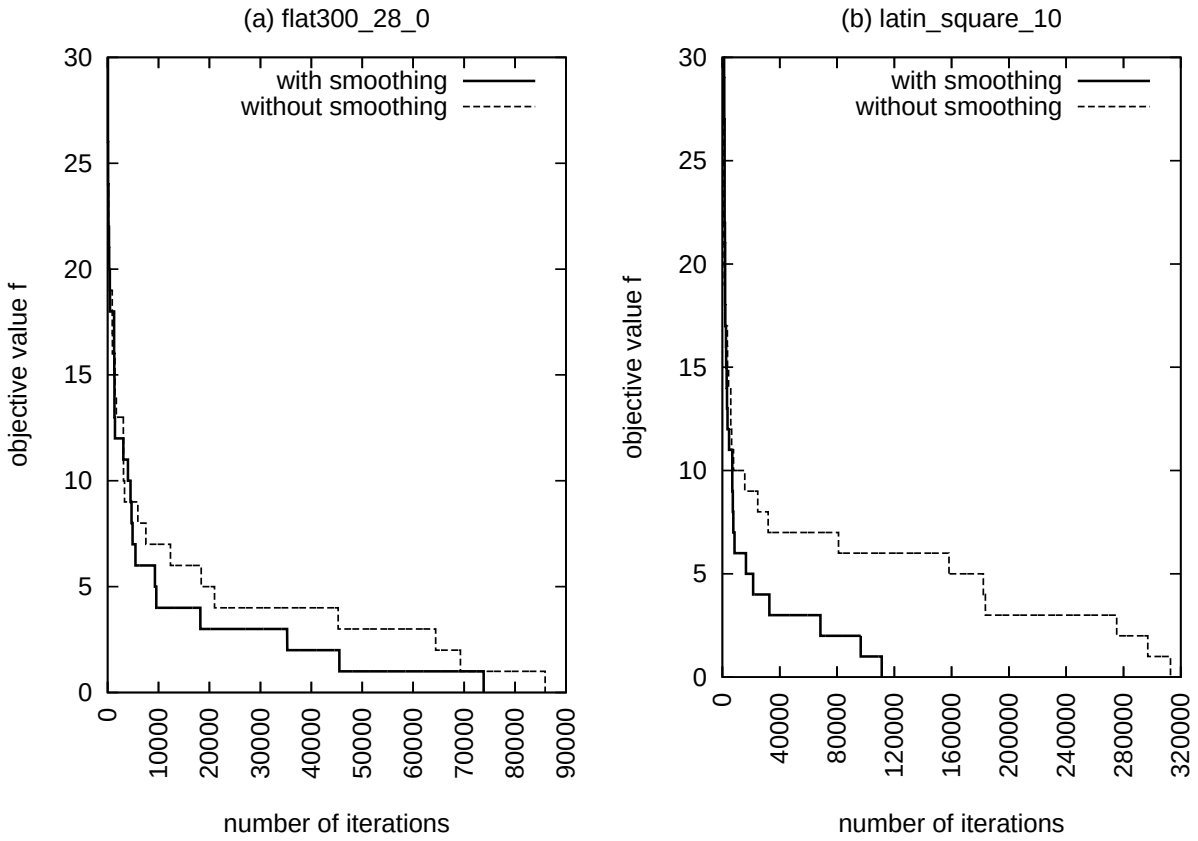

Figure 3.5: Running profile of PLS (with smoothing) and PLS_1 (without smoothing) on instances flat300_28_0 and latin_square_10.

The running profiles of PLS and PLS_1 are shown in Figure 3.5 on two selected instances: Figure 3.5(a) for flat300_28_0 ($k = 32$), and Figure 3.5(b) for latin_square_10 ($k = 101$). We observe that though both algorithms successfully obtain a legal k-coloring, PLS converges to the best solution more quickly than PLS_1, i.e., the objective value f of PLS decreases more quickly than that of PLS_1. Consequently, PLS needs less iterations to attain a legal solution. This experiment illustrated the benefit of using probability smoothing technique in PLS.

3.5.2 Comparison of different group selection strategies

The group selection strategy plays an important role in PLS. At each iteration, each vertex selects a suitable group based on the group selection strategy to produce a new solution for the next round of the descent-based local search optimization. In this section, we show an analysis of the group selection strategies to confirm the interest of the adopted hybrid strategy which combines random and greedy strategies.

The investigation was carried out between PLS and its variant PLS_2, which is obtained from PLS by means of replacing the hybrid group selection strategy with the roulette wheel selection strategy. In the experiment, each instance was tested 20 times independently with different random seeds. The number of successful runs, the average number of iterations and the average running time of successful runs are reported.

Table 3.6: Comparative performance of PLS and PLS_2.

	PLS_2			PLS		
Instance	k_1(#hit)	#iter	time(s)	k_2(#hit)	#iter	time(s)
le450_25c	26(0/20)	-	-	26(13/20)	4.7×10^6	181.39
	27(20/20)	7.0×10^5	26.86	27(20/20)	1.5×10^6	61.13
DSJR500.1	12(0/20)	-	-	12(20/20)	7.8×10^4	1.91
	13(20/20)	2.0×10^6	50.42	13(20/20)	3.0×10^3	0.10
DSJR500.1c	85(0/20)	-	-	85(02/20)	4.6×10^6	699.63
	86(0/20)	-	-	86(20/20)	3.6×10^6	529.47
	87(20/20)	3.2×10^6	361.97	87(20/20)	6.9×10^5	108.12
DSJC1000.1	21(09/20)	2.0×10^7	1508.48	21(20/20)	1.4×10^7	1223.18
	22(20/20)	6.0×10^5	41.82	22(20/20)	8.0×10^5	64.77

Table 3.6 show comparative results of PLS (with its hybrid group selection strategy) with PLS_2 (with the roulette wheel selection strategy) for the chosen instances. The results indicate that PLS significantly outperforms PLS_2 in terms of the best k value and the number of successful running times. For example, on instance DSJR500.1c, PLS colors this graph with 85 colors, while PLS_2 needs more colors ($k = 87$) to color it. A similar observation can be found on instance le450_25c, for which PLS obtains a legal 26-coloring, while PLS_2 only obtains a 27-coloring. Furthermore, when they need the same number of colors to color a graph DSJC1000.1, PLS achieves it with a higher success rate compared to PLS_2. This experiment confirms the interest of the adopted hybrid selection strategy.

3.5.3 Benefit of the probability learning scheme

We compared the performance of the PLSCOL (with tabu search and group matching procedure) algorithm with a variant (denoted by $PLSCOL_1$) where the probability learning component is disabled. At each iteration, instead of generating a new initial solution with the probability matrix, $PLSCOL_1$ generates a new solution at random (i.e., line 5 of Algorithm 4 is replaced by the random generation procedure) and then uses the tabu search procedure to improve the initial solution. In other words, $PLSCOL_1$ repetitively restarts the TabuCol procedure. We ran $PLSCOL_1$ under the same stopping condition as before – $PLSCOL_1$ stops if a legal k-coloring is found or the maximum allowed run time of 5 CPU hours is reached. Each instance was solved 10 times.

Table 3.7 summarizes the computational statistics of PLSCOL and $PLSCOL_1$. Columns 1-2 indicates the name of each instance, its chromatic number χ when it is known or the best-known result reported in the literature (k^*). For each algorithm, we report the following information: the smallest number of colors (k) used by the algorithm, the frequency of successful runs out of 10 runs ($\#succ$), the average number of generations ($\#gen$), the average number of tabu search iterations (or moves) ($\#iter$) and the average running time in seconds. The lowest k values between the two algorithms are in bold (a smaller value indicates a better performance in terms of solution quality). When the two algorithms achieve the same

Table 3.7: Comparative results of PLSCOL and PLSCOL$_1$ on the difficult DIMACS graphs.

		PLSCOL					PLSCOL$_1$				
Instance	χ/k^*	k	#succ	#gen	#iter	time(s)	k	#succ	#gen	#iter	time(s)
DSJC250.5	?/28	**28**	10/10	3	4.0×10^5	4	**28**	10/10	58	1.1×10^7	102
DSJC500.1	?/12	**12**	07/10	69	7.5×10^6	43	**12**	10/10	8936	1.9×10^9	12808
DSJC500.5	?/47	**48**	03/10	761	7.9×10^7	1786	49	06/10	1122	2.5×10^8	5543
DSJC500.9	?/126	**126**	10/10	187	2.4×10^7	747	127	10/10	362	9.2×10^7	2704
DSJC1000.1	?/20	**20**	01/10	369	2.9×10^8	3694	21	10/10	2	3.7×10^5	4
DSJC1000.5	?/83	**87**	10/10	203	2.7×10^7	1419	89	02/10	492	1.4×10^8	7031
DSJC1000.9	?/222	**223**	05/10	2886	3.1×10^8	12094	229	05/10	270	1.0×10^8	9237
DSJR500.1c	?/85	**85**	10/10	317	3.2×10^7	386	**85**	10/10	554	8.3×10^7	1825
DSJR500.5	?/122	**126**	08/10	464	7.3×10^7	1860	127	01/10	2,701	4.3×10^8	8592
le450_15c	15/15	**15**	07/10	2883	2.8×10^8	1718	**15**	10/10	155	2.1×10^7	238
le450_15d	15/15	**15**	03/10	2787	2.8×10^8	2499	**15**	10/10	766	1.1×10^8	1314
le450_25c	25/25	**25**	10/10	1968	2.0×10^8	1296	26	10/10	1	8.1×10^4	1
le450_25d	25/25	**25**	10/10	2110	2.2×10^8	1704	26	10/10	1	1.1×10^5	2
flat300_26_0	26/26	**26**	10/10	49	4.9×10^6	195	**26**	10/10	31	5.1×10^6	254
flat300_28_0	28/28	**30**	10/10	147	1.5×10^7	233	31	10/10	95	1.9×10^7	242
flat1000_76_0	76/81	**86**	01/10	908	1.1×10^8	5301	89	02/10	339	9.1×10^7	3709
R250.5	?/65	**66**	10/10	865	1.1×10^8	705	**66**	01/10	1793	2.3×10^8	2038
R1000.1c	?/98	**98**	10/10	88	9.1×10^6	256	**98**	10/10	110	2.0×10^7	702
R1000.5	?/234	**254**	04/10	189	3.7×10^7	7818	260	10/10	1	3.1×10^5	124
latin_square_10	?/97	**99**	08/10	666	6.7×10^7	2005	103	10/10	444	9.7×10^7	7769

result in terms of number of colors used, we underlined the smallest number of iterations iterations (which indicates a better performance in terms of computational efficiency).

From Table 3.7, it can be seen that for the 20 instances, PLSCOL obtains a better solution for 12 instances and an equal solution for the 8 remaining instances. With the help of the learning scheme, the improvement of PLSCOL over PLSCOL$_1$ is quite significant for several very difficult graphs [4] like DSJC1000.9 (-6 colors), at flat1000_76_0 (-3 colors), R1000.5 (-6 colors) and latin_square_10 (-4 colors). Finally, we observe that for the 8 instances where both algorithms achieve an equal result in terms of colors used, PLSCOL requires less iterations and less computing time than PLSCOL$_1$ in 6 cases.

This comparative study between PLSCOL and PLSCOL$_1$ demonstrates the effectiveness of the probability learning scheme used in our PLSCOL algorithm.

3.5.4 Benefit of group matching procedure

We now investigate the usefulness of the group matching procedure, which is used to provide feedback information to the probability learning component. For this purpose, we create PLSCOL$_2$, which is a PLSCOL variant where we replace line 10 of Algorithm 4 with the simple group identifying procedure. We ran PLSCOL$_2$ under the same stopping condition as before – PLSCOL$_2$ stops if a legal k-coloring is found or the maximum allowed run time of 5 CPU hours is reached. Each tested instance was solved 10 times.

Table 3.8 displays the comparative performance between our PLSCOL and PLSCOL$_2$. In the table, we compare these two algorithms based on same indicators adopted in Table 3.7. From this table, we clearly observe that PLSCOL significantly outperforms PLSCOL$_2$, achieving the better objective values on 11 out of 20 instances and the equal objective values on the remaining 9 instances. Moreover, for these 9 instances where both algorithms achieve same best objective value, PLSCOL needs less iterations and less computing time than PLSCOL$_2$ on 7 instances (as indicated by underline in Table 3.8). These observations confirm the benefit of group matching procedure applied in our PLSCOL algorithm.

4. For these instances, even gaining one color could be difficult, since when k is close to χ, finding a legal coloring is usually much harder for k than for $k+1$.

Table 3.8: Comparative results of PLSCOL and PLSCOL$_2$ on the difficult DIMACS graphs.

		PLSCOL					PLSCOL$_2$				
Instance	χ/k^*	k	#succ	#gen	#iter	time(s)	k	#succ	#gen	#iter	time(s)
DSJC250.5	?/28	**28**	10/10	3	4.0×10^5	4	**28**	10/10	103	6.7×10^7	1026
DSJC500.1	?/12	**12**	07/10	69	7.5×10^6	43	**12**	10/10	5	5.5×10^6	59
DSJC500.5	?/47	**48**	03/10	761	7.9×10^7	1786	50	10/10	15	6.2×10^6	154
DSJC500.9	?/126	**126**	10/10	187	2.4×10^7	747	**126**	06/10	931	2.7×10^8	9314
DSJC1000.1	?/20	**20**	01/10	369	2.9×10^8	3694	21	10/10	0	8.9×10^4	1
DSJC1000.5	?/83	**87**	10/10	203	2.7×10^7	1419	89	09/10	407	6.9×10^7	4070
DSJC1000.9	?/222	**223**	05/10	2886	3.1×10^8	12094	224	02/10	826	1.0×10^8	8265
DSJR500.1c	?/85	**85**	10/10	317	3.2×10^7	386	**85**	10/10	60	5.5×10^7	600
DSJR500.5	?/122	**126**	08/10	464	7.3×10^7	1860	**126**	07/10	753	3.9×10^8	7534
le450_15c	15/15	**15**	07/10	2883	2.8×10^8	1718	**15**	02/10	874	1.5×10^9	8744
le450_15d	15/15	**15**	03/10	2787	2.8×10^8	2499	16	10/10	0	3.3×10^5	3
le450_25c	25/25	**25**	10/10	1968	2.0×10^8	1296	26	10/10	0	1.2×10^5	1
le450_25d	25/25	**25**	10/10	2110	2.2×10^8	1704	26	10/10	1	1.3×10^5	2
flat300_26_0	26/26	**26**	10/10	49	4.9×10^6	195	**26**	07/10	603	1.4×10^8	6034
flat300_28_0	28/28	**30**	10/10	147	1.5×10^7	233	31	10/10	410	2.4×10^8	4101
flat1000_76_0	76/81	**86**	01/10	908	1.1×10^8	5301	87	01/10	321	3.9×10^7	3212
R250.5	?/65	**66**	10/10	865	1.1×10^8	705	**66**	10/10	151	2.0×10^8	1516
R1000.1c	?/98	**98**	10/10	88	9.1×10^6	256	**98**	10/10	18	3.8×10^6	181
R1000.5	?/234	**254**	04/10	189	3.7×10^7	7818	255	05/10	241	1.6×10^7	2425
latin_square_10	?/97	**99**	08/10	666	6.7×10^7	2005	100	06/10	648	2.4×10^8	6480

3.5.5 Effect of the penalization factor β

The penalization factor β is used to determine the new probability vector when an item selects an incorrect group. Preliminary results suggest that the performance of PLSCOL is more sensitive to β than the other two parameters α and γ. In the following, we report a detailed analysis of β on six selected instances. To avoid the influence caused by random numbers, we set the random seed to 1 in the experiments. For each parameter setting and each instance, we conduct an independent experiment with a maximum allowed time limit of 5 CPU hours. We set other parameters to their default values (as shown in Table 3.1) and only vary the parameter β.

Table 3.9: Effect of penalization factor β on the running time (s) of PLSCOL.

β	DSJC250.5	DSJC500.9	DSJR500.1c	R1000.1c	DSJC1000.9	flat1000_76_0
0.45	13.10	2055.03	**128.40**	88.50	*	*
0.40	17.65	367.57	139.73	229.88	5884.32	7474.21
0.35	13.10	1096.23	471.30	**34.25**	**1158.67**	3154.03
0.30	17.57	2467.98	242.40	2644.97	*6133.87*	*7519.52*
0.25	13.28	903.89	*6120.46*	*8383.67*	7122.16	1035.84
0.20	13.26	*10149.23*	1412.54	164.01	3008.86	*
0.15	*17.66*	8765.75	1519.88	55.96	4015.04	7163.64
0.10	**11.48**	**316.26**	1232.88	113.23	2815.88	1244.57
0.05	17.14	*	5753.18	181.73	4349.16	**995.73**
Δ_{max}	6.18	9832.97	5992.06	8349.42	4975.20	6523.79

Table 3.9 shows the impact of β on the performance of PLSCOL with nine different values for β, $\beta \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45\}$. In this table, we report the running time (the best values are in bold and the worst values are in italic) needed to find the given best k-coloring for each parameter value and each instance. "$*$" indicates that PLSCOL cannot find a k-coloring with this parameter setting. At the last row of the table, we also give the maximum running time difference of PLSCOL under

two different β values. Table 3.9 indicates that PLSCOL can find a legal coloring with the given best k in the majority of cases except for DSJC500.9 with $\beta = 0.05$, DSJC1000.9 with $\beta = 0.45$ and flat1000_76_0 with $\beta = 0.45$ and $\beta = 0.20$ respectively. This parameter mainly affects the running time of PLSCOL to find the given k-coloring. Taking the instance DSJC500.9 as an example, with $\beta = 0.10$, PLSCOL finds the best 126-coloring within 316.26 seconds while it needs much more time (10149.23 seconds) to find the 126-coloring when $\beta = 0.20$. The maximum difference of running time with different β values is 9832.97 seconds. We also observe that the optimal setting of β depends to a large extent on the structures of the instances which greatly vary for different graphs. Although the time of PLSCOL is sensitive to the β values, PLSCOL can successfully find the k-coloring with many β values. This explains why we used a fixed β value ($\beta = 0.10$) for most of the 20 tested difficult instances and varied this parameter for the remaining instances.

3.6 Chapter conclusion

In this chapter, we presented a Probability learning based Local Search (PLS) approach for solving the class of grouping problems. The proposed PLS approach combines probability learning technique with optimization procedure. Probability learning is used to maintain and update a set of probability vectors, each probability vector specifying the probability that an item belongs to a particular group. At each iteration, PLS builds a starting grouping solution according to the probability vectors and with the help of a group selection strategy. PLS then applies a optimization procedure to improve the given grouping solution until a local optimum is reached. At this point, a solution comparison is made between the starting solution and the ending local optimum solution. The probability vector of each item is updated according to the situation of the item. Specifically, PLS rewards the selected group of the item if the item stays in the original group, otherwise PLS penalizes the selected group and compensates the new group. A case study of PLS has been made on the well-known Graph Coloring Problem (GCP), and the corresponding PLS algorithm is denoted as PLSCOL. PLSCOL further enhances the generic PLS approach designed for solving grouping problems from two aspects. The first enhancement improves the probability learning scheme of the generic PLS approach by using a group matching procedure to find the group-to-group relation between a starting solution and its improved solution. This matching procedure copes with the difficulty raised by symmetric solutions of GCP and allows to as a specific grouping problem. The second enhancement concerns the coloring algorithm based on tabu search, which is both more powerful than the descent-based coloring algorithm used in PLS, and still remains simple compared to more complex hybrid algorithms.

Experimental evaluations on popular DIMACS challenge benchmark graphs showed that PLSCOL competes favorably with all existing local search based coloring algorithms. Compared with the most effective hybrid evolutionary algorithms which are much more sophisticated in their design, PLSCOL remains competitive. Furthermore, we provided an analysis to show the impact of some key ingredients of PLSCOL over their performance.

4

Opposition-based Memetic Search for Maximum Diversity Problem

As a usual model for a variety of practical applications, the maximum diversity problem is computational challenging. In this chapter, we present an *Opposition-Based Memetic Algorithm (OBMA)* for solving *Maximum Diversity Problem (MDP)*, which integrates the concept of *Opposition-Based Learning (OBL)* into the well-known memetic search framework. OBMA explores both candidate solutions and their opposite solutions during its initialization and evolution processes. Combined with a powerful local optimization procedure and a rank-based quality-and-distance pool updating strategy, OBMA establishes a suitable balance between exploration and exploitation of its search process. Computational results on 80 popular MDP benchmark instances show that the proposed algorithm matches the best-known solutions for most of instances, and finds improved best solutions (new lower bounds) for 22 instances. We provide experimental evidences to highlight the beneficial effect of opposition-based learning for solving MDP. The main content reported in this chapter is based on an article published in *IEEE Transactions on Evolutionary Computation* [Zhou *et al.*, 2017c].

Contents

4.1 Introduction

MDP belongs to a large family of diversity or dispersion problems whose purpose is to identify a subset S from a set N of elements while optimizing an objective function defined over the distances between the elements in S [Prokopyev *et al.*, 2009]. Over the past three decades, MDP has been widely studied under different names, such as max-avg dispersion [Ravi *et al.*, 1994], edge-weighted clique [Macambira and De Souza, 2000], dense k-subgraph [Feige *et al.*, 2001], maximum edge-weighted subgraph [Macambira, 2002] and equitable dispersion [Prokopyev *et al.*, 2009]. In addition, MDP also proves to be a useful model to formulate a variety of practical applications including facility location, molecular structure design, agricultural breeding stocks, composing jury panels and product design [Glover *et al.*, 1998; Martí *et al.*, 2013]. In terms of computational complexity, MDP is known to be $\mathcal{NP}$-hard [Ghosh, 1996].

Given the interest of MDP, a large number of solution methods for MDP have been investigated. These methods can be divided into two main categories: exact algorithms and heuristic algorithms. In particular, exact algorithms like [Aringhieri *et al.*, 2009; Martí *et al.*, 2010] are usually effective on small instances with $n < 150$. To handle larger instances, heuristic algorithms are often preferred to find sub-optimal solutions in an acceptable time frame. Existing heuristic algorithms for MDP include construction methods [Ghosh, 1996; Glover *et al.*, 1998], greedy randomized adaptive search procedure (GRASP)[Aringhieri *et al.*, 2008; Duarte and Martí, 2007; Santos *et al.*, 2005], iterative tabu search (ITS) [Palubeckis, 2007], variable neighborhood search (VNS) [Aringhieri and Cordone, 2011; Brimberg *et al.*, 2009], fine-tuning iterated greedy algorithm (TIG) [Lozano *et al.*, 2011], memetic and hybrid evolutionary algorithms (MSES [de Freitas *et al.*, 2014], G_SS [Gallego *et al.*, 2009], MAMDP [Wu and Hao, 2013] and TS/MA [Wang *et al.*, 2014]). Comprehensive surveys and comparisons of some important heuristic algorithms prior to 2012 for MDP can be found in [Aringhieri and Cordone, 2011; Martí *et al.*, 2013].

Among the existing heuristics for MDP, two methods involve hybridization of heuristics and machine learning techniques. In [Santos *et al.*, 2005], the proposed GRASP_DM algorithm combines GRASP with a data mining technique (i.e., frequent itemset mining). After each GRASP phase, the data mining process extracts useful patterns from recorded elite solutions to guide the following GRASP iterations. These patterns correspond to items that are shared by a significant number of elite solutions. Another learning-based heuristic is LTS_EDA [Wang *et al.*, 2012], which uses data mining techniques (k-means clustering and estimation of distribution algorithms) to extract useful information from the search history of tabu search in order to guide the search procedure to promising search regions. These learning-based methods have reported competitive results when they were published.

In this chapter, we propose a new learning-based optimization method for solving MDP. The proposed *Opposition-Based Memetic Algorithm (OBMA)* integrates the concept of *Opposition-Based Learning (OBL)* into the popular *Memetic Algorithm (MA)* framework. OBMA brings several improvements into the original MA framework. First, we employ OBL to reinforce population initialization as well as the evolutionary search process, by simultaneously considering a candidate solution and its corresponding opposite solution. Second, we apply a tabu search procedure for local optimization which relies on an improved parametric constrained neighborhood. Third, we propose a rank-based quality-and-distance pool updating strategy to maintain a healthy population diversity. We identify the main contributions of this work as follows.

— From an **algorithmic perspective**, we explore for the first time the usefulness of opposition-based learning to enhance a popular method (i.e., MA) for combinatorial optimization. We investigate how OBL can be beneficially integrated into the MA framework and show the effectiveness of the approach within the context of solving the maximum diversity problem.

— From a **computational perspective**, we compare the proposed OBMA algorithm with state-of-the-art results on several sets of 80 large size MDP benchmark instances with $2,000$ to $5,000$ elements. Our results indicate that OBMA matches most of the best-known results and in particular finds improved best solutions (new lower bounds) for 22 instances. These new bounds are valuable for the assessment of new MDP algorithms. These computational results demonstrate the competitiveness of OBMA and the benefit of using OBL to enhance a memetic algorithm.

The remainder of the chapter is organized as follows. After a brief introduction of opposition-based learning and memetic search in Section 4.2, we present in Section 4.3 the proposed opposition-based memetic algorithm. Sections 4.4 and 4.5 show computational results and comparisons as well as an experimental study of key issues of the proposed algorithm. Conclusions and perspective are provided in Section 4.6.

4.2 Background

This section introduces the concept of opposition-based learning and the general memetic search framework, which are then combined in the proposed approach.

4.2.1 Opposition-based Learning

Opposition-Based Learning (OBL) was originally proposed as a machine intelligence scheme for reinforcement learning [Tizhoosh, 2005]. The main idea behind OBL is the simultaneous consideration of a candidate solution and its corresponding opposite solution. To explain the concept of opposition, we consider a real number $x \in [a, b]$, then the opposite number $\overline{x}$ is defined as $\overline{x} = a + b - x$. For the case of MDP, we define the concept of opposite solution in Section 4.3. OBL is a fast growing research field in which a variety of new theoretical models and technical methods have been studied to deal with complex and significant problems [Al-Qunaieer *et al.*, 2010; Rahnamayan *et al.*, 2008a; Ventresca and Tizhoosh, 2008; Xu *et al.*, 2014]. Recently, the idea of OBL has also been used to reinforce several *global* optimization methods such as differential evolution, particle swarm optimization, biogeography-based optimization, artificial neural network, bee and ant colony optimization [Xu *et al.*, 2014; Aziz and Tayarani-N., 2016].

To apply OBL to solve an optimization problem, one needs to answer a fundamental question: given a solution from the search space, why is it more advantageous to consider an opposite solution of the current solution than a second pure random solution? For one dimensional search space, a proof and an empirical evidence confirmed how much an opposite solution is better than a uniformly generated random solution [Rahnamayan *et al.*, 2008b]. This result was further generalized to the N-dimensional search spaces for black-box (continuous) problems in [Rahnamayan *et al.*, 2012].

We observe that existing studies on OBL-based optimization concern only global optimization with two exceptions. In 2008, Ventresca and Tizhoosh [Ventresca and Tizhoosh, 2008] proposed a diversity maintaining population-based incremental learning algorithm for solving the *Traveling Salesman Problem (TSP)*, where the concept of opposition was used to control the amount of diversity within a given sample population. In 2011, Ergezer and Simon [Ergezer and Simon, 2011] hybridized open-path opposition and circular opposition with biogeography-based optimization for solving the graph coloring problem and TSP. The main difficulty of these applications is how to define and evaluate opposite solutions in a discrete space. OBL being a generally applicable technique, its efficiency depends on the matching degree between the definition of OBL and the solution space of the considered problem, as well as the rationality justifying a combination of OBL with a search algorithm [Xu *et al.*, 2014].

4.2.2 Memetic algorithm

The *Memetic Algorithm (MA)* framework [Moscato, 1999; Krasnogor and Smith, 2005] is a well-known hybrid search approach combining population-based search and local optimization. MA has been successfully applied to tackle numerous classical $\mathcal{NP}$-hard problems [Chen *et al.*, 2011; Hao, 2012], such as graph coloring [Lü and Hao, 2010], graph partitioning [Benlic and Hao, 2011; Galinier *et al.*, 2011] and generalized quadratic multiple knapsack [Chen and Hao, 2016] as well as the maximum diversity problem [de Freitas *et al.*, 2014; Wu and Hao, 2013].

Algorithm 5: The general memetic algorithm framework

Input: Problem instance I and population size p
Output: The best solution S^* found

```
// we suppose I is a maximization problem
begin
    // build an initial population
    P = {S^1, S^2, ..., S^p} ← PoolInitialization();
    // record the best solution found so far
    S* = arg max{f(S^i)|i = 1, 2, ..., p};
    while a stopping condition is not reached do
        (S^i, ..., S^j) ← ParentsSelection(P);
        // generate an offspring solution
        S^o ← CrossoverOperator(S^i, ..., S^j);
        // improve the offspring solution
        S^o ← LocalImprovement(S^o);
        // accept or discard the improved solution
        P ← UpdatePopulation(P, S^o);
        // update the best solution found
        if f(S^o) > f(S*) then
            S* ← S^o;
```

A typical MA algorithm (Algorithm 5) begins with a set of random or constructed solutions (initial population). At each generation, MA selects two or more parent solutions from the population, and performs a recombination or crossover operation to generate one or more offspring solutions. Then a local optimization procedure is invoked to improve the offspring solution(s). Finally, a population management strategy is applied to decide if each improved offspring solution is accepted to join the population. The process repeats until a stopping condition is satisfied. We show below how OBL and MA can be combined to obtain a powerful search algorithm for the highly combinatorial maximum diversity problem.

4.3 Opposition-based memetic search for MDP

We describe in this section the proposed opposition-based memetic algorithm for MDP. We start with the issues of solution representation and search space, followed by a detailed presentation of the ingredients of the proposed approach.

4.3.1 Solution representation and search space

Given a MDP instance with set $N = \{e_1, e_2, \ldots, e_n\}$ and integer m, any subset $S \subset N$ of size m is a feasible solution. A candidate solution S can then be represented by $S = \{e_{S(1)}, e_{S(2)}, \ldots, e_{S(m)}\}$ such that $S(i)$ is the index of element i in N or equivalently by a binary vector of size n such that exactly m variables receive the value of 1 and the other $n - m$ variables receive the value of 0. The quality of a candidate solution S is assessed by the objective function f of Equation (1.1).

Given a MDP instance, the search space Ω is composed of all the m-element subsets of N, i.e., $\Omega = \{S \subset N : |S| = m\}$. The size of Ω is given by $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ and increases extremely fast with n and m.

4.3.2 Main scheme

The proposed OBMA algorithm for MDP is based on opposition learning, which relies on the key concept of opposite solution in the context of MDP, which is defined as follows.

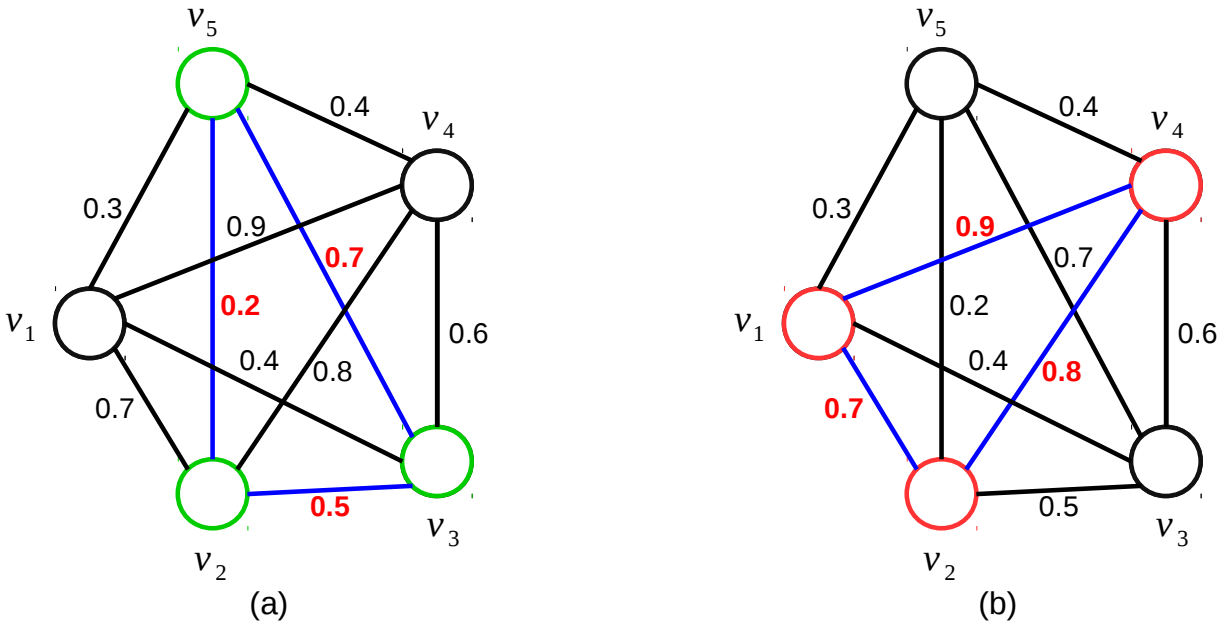

Figure 4.1: (a) A candidate solution $S = \{v_2, v_3, v_5\}$ and (b) its an opposite solution $S' = \{v_1, v_2, v_4\}$.

Definition 2 (Opposite solution)**.** *Given a MDP instance with set N and integer m, let S be a feasible solution of Ω represented by its binary n-vector x, an opposite solution $\overline{S}$ corresponds to a feasible binary vector $\overline{x}$ whose components match the complement* [1] *of x as closely as possible.*

According to this definition, if $m < \frac{n}{2}$, $\overline{S}$ corresponds to any subset of elements of size of m from $N \setminus S$. If $m = \frac{n}{2}$, the unique opposite solution is given by $\overline{S} = N \setminus S$. If $m > \frac{n}{2}$, $\overline{S}$ is any subset of $n - m$ elements from $N \setminus S$, completed by other $2m - n$ elements from S. An illustrative example is shown in Figure 4.1.

We observe that a diversification framework introduced in [Glover, 1997] also yields the type of opposite solution provided by our definition and applies to constraints more general than the constraint defined by Equation (1.2). We make two related comments. First, as we observe in Section 5.5, the MDP benchmark instances in the literature correspond to the case $m \leqslant \frac{n}{2}$. Second, in practice, when an opposite solution is required while there are multiple opposite solutions, we can just select one solution at random among the candidate opposite solutions.

The proposed OBMA algorithm consists of four key components: an opposition-based population initialization procedure, a backbone-based crossover operator, an opposition-based double trajectory search procedure and a rank-based quality-and-distance pool updating strategy. OBMA starts from a collection of diverse elite solutions which are obtained by the opposition-based initialization procedure (Section 4.3.3). At each generation, two parent solutions are selected at random from the population, and then the backbone-based crossover operator (Section 4.3.5) is applied to the selected parents to generate an offspring solution and a corresponding opposite solution. Subsequently, the opposition-based double trajectory search procedure (Section 4.3.4) is invoked to search simultaneously from the offspring solution and its opposite solution. Finally, the rank-based pool updating strategy (Section 4.3.6) decides whether these two improved offspring solutions should be inserted into the population. This process repeats until a stopping condition (e.g., a time limit) is satisfied. The general framework and pseudo-code of the OBMA algorithm are shown in Figure 4.2 and Algorithm 6 respectively, while its components are described in the following sections.

4.3.3 Opposition-based population initialization

The initial population P is composed of p diverse and high quality solutions. Unlike traditional population initialization, our population initialization procedure integrates the concept of OBL. As shown in

1. *Let $x \in \{0,1\}^n$, its complement $\overline{x}$ is an n-vector such that $\overline{x}[i] = 1$ if $x[i] = 0$; $\overline{x}[i] = 0$ if $x[i] = 1$.*

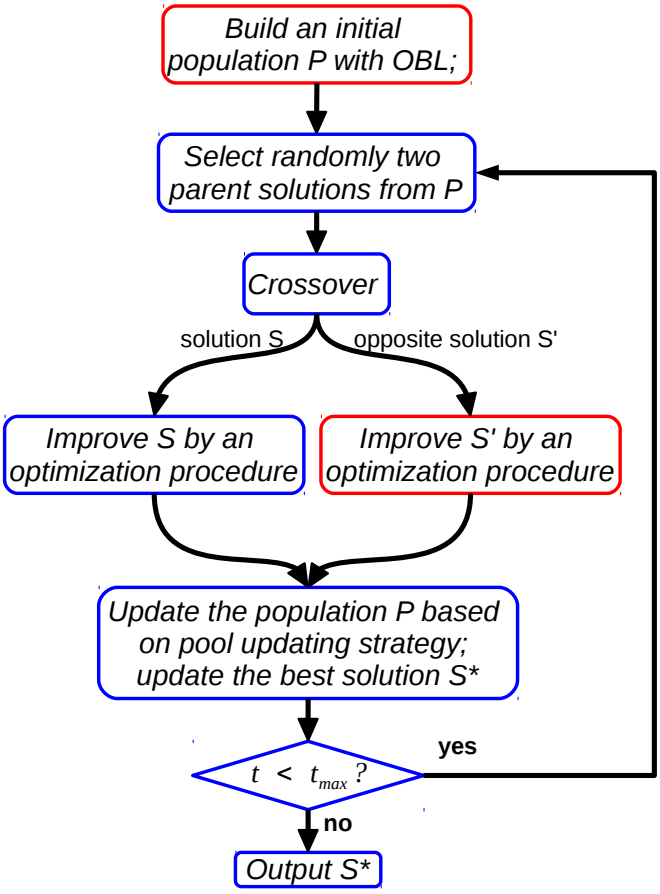

Figure 4.2: A general framework of OBMA algorithm.

Algorithm 6: Opposition-based memetic algorithm for MDP

Input: An instance of $n \times n$ distance matrix (d_{ij}), and an integer $m < n$.
Output: The best solution S^* found

```
1  begin
2      // build an initial population, Section 4.3.3
3      P = {S^1, S^2, ..., S^p} ← OppositionBasedPoolInitialize()
4      S* ← arg max{f(S^i) : i = 1, 2, ..., p}
5      while stopping condition not reached do
6          randomly select two parent solutions S^i and S^j from P
7          // generate an offspring solution and its opposite solution by crossover, Section 4.3.5
8          S^o, S̄^o ← BackboneBasedCrossover(S^i, S^j)
9          // perform a double trajectory search, Section 4.3.4
10         S^o ← TabuSearch(S^o)  /*trajectory 1: search around S^o */
11         // update the best solution found
12         if f(S^o) > f(S*) then
13             S* ← S^o
14         // insert or discard the improved solution, Section 4.3.6
15         P ← RankBasedPoolUpdating(P, S^o)
16         S̄^o ← TabuSearch(S̄^o)  /*trajectory 2: search around S̄^o */
17         // update the best solution found
18         if f(S̄^o) > f(S*) then
19             S* = S̄^o
20         // insert or discard the improved solution, Section 4.3.6
21         P ← RankBasedPoolUpdating(P, S̄^o)
```

Algorithm 7, the OBL-based initialization procedure considers not only a random candidate solution but also a corresponding opposite solution. Specifically, we first generate a pair of solutions, i.e., a random solution $S_r \in \Omega$ and a corresponding opposite solution $\overline{S}_r$ according to Definition (2) of Section 4.3.2 (if multiple opposite solutions exist, one of them is taken at random). These two solutions are then improved by the tabu search procedure described in Section 4.3.4. Finally, the better one of the two improved solutions S' is inserted into the population if S' is not the same as any existing individual of the population. Otherwise, we modify S' with the swap operation (see Section 4.3.4) until S' becomes different from all individuals in P before inserting it into the population. This procedure is repeated until the population is filled up with p solutions. With the help of this initialization procedure, the initial solutions of P are not only of good quality, but also of high diversity.

4.3.4 Opposition-based double trajectory search procedure

In the proposed OBMA algorithm, we use an *Opposition-based Double Trajectory Search (ODTS)* procedure for local optimization. ODTS simultaneously searches around an offspring solution S^o and one opposite solution $\overline{S^o}$. The local optimization procedure used here is an improved constrained neighborhood tabu search. Tabu search is a well-known metaheuristic that guides a local search heuristic to explore the solution space beyond local optimality [Glover and Laguna, 1997]. The original constrained neighborhood tabu search algorithm was proposed in [Wu and Hao, 2013], which is specifically designed for MDP by using a constrained neighborhood and a dynamic tabu tenure management mechanism. Compared with this tabu search algorithm, our improved tabu search procedure (see Algorithm 8) distinguishes itself by its parametric constrained neighborhood which allows the search process to explore more promising candidate solutions. In the following, we present the key ingredients of this local optimization procedure including the parametric constrained neighborhood, the fast neighborhood evaluation technique and the dynamic tabu

Algorithm 7: Opposition-based population initialization

Input: Population size p.
Output: An initial population $P = \{S^1, S^2, \ldots, S^p\}$

begin
  $count \leftarrow 0$
  while $count < p$ **do**
    generate a random solution S^r and its opposite solution $\overline{S^r}$
    $S^r \leftarrow TabuSearch(S^r)$
    $\overline{S^r} \leftarrow TabuSearch(\overline{S^r})$
    // identify the better solution between S^r and $\overline{S^r}$
    $S' \leftarrow \arg\max\{f(S^r), f(\overline{S^r})\}$
    // insert S' into the population P or modify it
    if S' *is different from any solutions in the* P **then**
      add S' into the population P directly
    else
      modify S' and add it into the population P
    $count \leftarrow count + 1$

tenure management scheme.

Algorithm 8: Parametric constrained neighborhood tabu search

Input: A starting solution S, the maximum allowed iterations $MaxIter$.
Output: The best solution S^* found

begin
  $S^* \leftarrow S$
  $iter \leftarrow 0$
  initialize the tabu list
  calculate the $gain(e_i)$ for each element $e_i \in N$ according to Equation (4.1)
  while $iter < MaxIter$ **do**
    $minGain \leftarrow \min\{gain(e_i) : e_i \in S\}$
    determine subset U_S^c according to Equation (4.7)
    $maxGain \leftarrow \max\{gain(e_i) : e_i \in N \setminus S\}$
    determine subset $U_{N\setminus S}^c$ according to Equation (4.8)
    choose a best eligible $swap(e_u, e_v)$
    $S \leftarrow S \setminus \{e_u\} \cup \{e_v\}$
    update the tabu list and $gain(e_i)$ for each element $e_i \in N$ according to Equation (4.9)
    if $f(S) > f(S^*)$ **then**
      $S^* \leftarrow S$
    $iter \leftarrow iter + 1$

Parametric constrained neighborhood

In general, local search for MDP starts from an initial solution S and subsequently swaps an element of S and an element of $N \setminus S$ according to some specific transition rule (e.g., accepting the first or the best improving transition). Clearly, the size of this neighborhood is bound by $O(m(n-m))$ and an exhaustive exploration of all the possible swap moves is too time-consuming for the large values of n. To reduce the size of the swap neighborhood, we employ an extension of a candidate list strategy sometimes called a

neighborhood decomposition strategy [Glover *et al.*, 1993] or a successive filtration strategy [Rangaswamy *et al.*, 1998], and which we refer to as a constrained swap strategy [Wu and Hao, 2013]. As it is shown in the experimental analysis of Section 4.5.1, although this constrained swap strategy accelerates the search process, it imposes a too strong restriction and may exclude some promising swap moves for the tabu search procedure. In this work, we introduce the *parametric constrained neighborhood* which adopts the idea of the constrained neighborhood, but weakens the imposed constraint by introducing a parameter ρ ($\rho \geqslant 1$) to control the size of the explored neighborhood. Both constrained neighborhoods rely on the notion of move gain of each element e_i with respect to the objective value of the current solution S defined as follows.

$$gain(e_i) = \sum_{e_j \in S} d_{ij}, i = 1, 2, \ldots, m \tag{4.1}$$

Let $swap(e_u, e_v)$ denote the swap operation which exchanges an element $e_u \in S$ against an element $e_v \in N \setminus S$. Once a swap $S \xrightarrow{swap(e_u,e_v)} S'$ is made, it provides a new solution $S' = S \setminus \{e_u\} \cup \{e_v\}$ and the move gain Δ_{uv} of this swap can be calculated according to the following formula.

$$\Delta_{uv} = f(S') - f(S) = gain(e_v) - gain(e_u) - d_{uv} \tag{4.2}$$

Equation (4.2) suggests that in order to maximize the move gain, it is a good strategy to consider swap moves that replaces in the current solution S an element e_u with a small gain by an element e_v out of S with a large gain. In other words, the search process can only consider swap moves that involve an element e_{u^*} from S with the minimal gain value and an element e_{v^*} in $N \setminus S$ with a maximal gain value. However the move gain also depends on the distance $d_{u^*v^*}$ between e_{u^*} and e_{v^*}. Specifically, we suppose swap(e_{u^*}, e_{v^*}) is the best swap, and for a current solution S, let $minGain = \min\{gain(e_i) : e_i \in S\}$ and $maxGain = \max\{gain(e_i) : e_i \in N \setminus S\}$ so we have

$$(e_{u^*}, e_{v^*}) \leftarrow \arg\max_{e_u \in S, e_v \in N \setminus S} \Delta_{u,v} \tag{4.3}$$

we suppose

$$e_{v'} \leftarrow \arg maxGain_{e_i \in N \setminus S} \textit{ and } e_{u'} \leftarrow \arg minGain_{e_i \in S} \tag{4.4}$$

so, we have

$$\Delta_{u^*,v^*} \geqslant \Delta_{u',v'} = maxGain - minGain - d_{u',v'} \geqslant maxGain - minGain - d_{max} \tag{4.5}$$

where $d_{max} = \max\{d_{ij}, 1 \leqslant i < j \leqslant n\}$, and we set $(\rho \geqslant 1)$, then we have

$$\Delta_{u^*,v^*} \geqslant maxGain - minGain - \rho * d_{max} = (maxGain - \frac{\rho}{2} * d_{max}) - (minGain + \frac{\rho}{2} * d_{max}) \tag{4.6}$$

Therefore, we have the parametric constrained neighborhood which relies on the two following sets.

$$U_S^c = \{e_i \in S \ : gain(e_i) \leqslant minGain + \frac{\rho}{2} d_{max}\} \tag{4.7}$$

and

$$U_{N \setminus S}^c = \{e_i \in N \setminus S \ : gain(e_i) \geqslant maxGain - \frac{\rho}{2} d_{max}\} \tag{4.8}$$

Therefore, a constrained neighbor solution S' can be obtained from S by swapping one element $e_u \in U_S^c$ and another element $e_v \in U_{N \setminus S}^c$. Clearly, the evaluation of all constrained neighboring solutions can be achieved in $O(|U_S^c| \times |U_{N \setminus S}^c|)$. Figure 4.3 demonstrates a simple diagram between the original neighborhood ($|S| * |N \setminus S|$) and its constrained neighborhood ($|U_S^c| * |U_{N \setminus S}^c|$). Conveniently, we can adjust the value of parameter ρ ($\rho \geqslant 1$) to control the size of the constrained neighborhood.

One notices that the neighborhood of [Wu and Hao, 2013] is a special case of the above neighborhood when $\rho = 2$. In general, a larger ρ value leads to a less constrained neighborhood compared to the neighborhood of [Wu and Hao, 2013], allowing thus additional promising candidate solutions to be considered by the tabu search procedure. The experimental analysis of Section 4.5.1 confirms the effectiveness of this parametric constrained neighborhood.

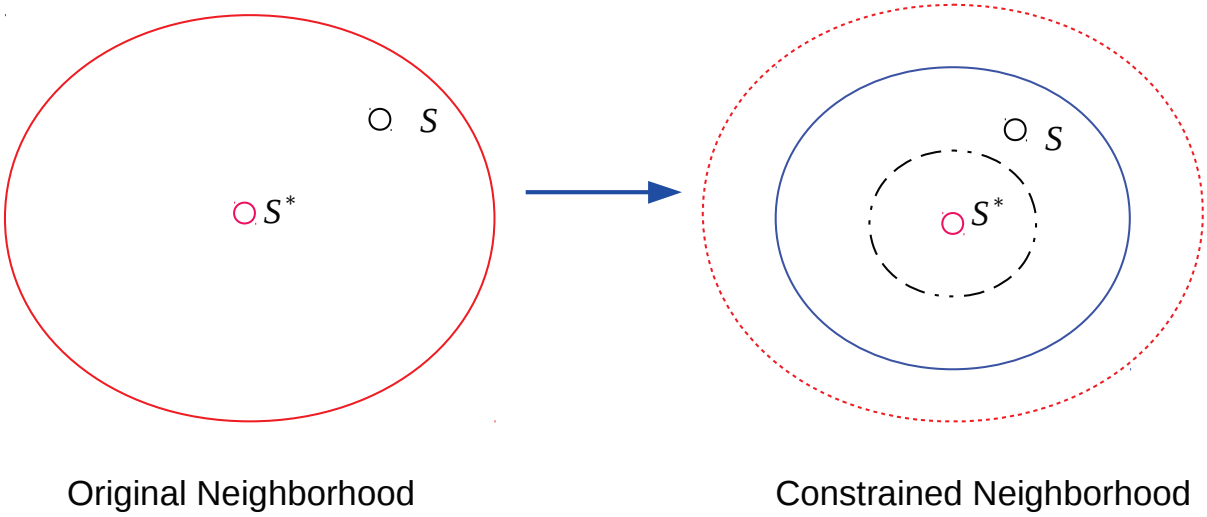

Figure 4.3: A simple diagram of the original neighborhood ($|S| * |N \setminus S|$) and its constrained neighborhood ($|U_S^c| * |U_{N\setminus S}^c|$).

Fast neighborhood evaluation technique

Once a $swap(e_u, e_v)$ move is performed, we need to update the gains $gain(e_i)$ affected by the move. To rapidly determine the gain of each element e_i, we resort to the fast neighborhood evaluation technique used in [Aringhieri *et al.*, 2008; Aringhieri and Cordone, 2011; Wu and Hao, 2013].

$$gain(e_i) = \begin{cases} gain(e_i) + d_{iv} & \text{if } e_i = e_u \\ gain(e_i) - d_{iu} & \text{if } e_i = e_v \\ gain(e_i) + d_{iv} - d_{iu} & \text{if } e_i \neq e_u \text{ and } e_i \neq e_v. \end{cases} \tag{4.9}$$

Updating the gains of n elements requires $O(n)$ time. Therefore, the time to update the parametric constrained neighborhood at each iteration is bounded by $O(n) + O(|U_S^c| \times |U_{N\setminus S}^c|)$.

Dynamic tabu tenure management scheme

Starting with a solution S, tabu search iteratively visits a series of neighboring solutions generated by the swap operator. At each iteration, a *best swap* (i.e., with the maximum move gain Δ_{uv}) is chosen among the eligible swap moves to transform the current solution even if the resulting solution is worse than the current solution. To prevent the search from cycling among visited solutions, tabu search typically incorporates a short-term history memory H, known as the tabu list [Glover and Laguna, 1997].

Initially, all elements are eligible for a swap operation. Once a $swap(e_u, e_v)$ is performed, we record it in the tabu list H to mark element e_u as tabu, meaning that element e_u is forbidden to join again solution S during the next T_u iterations (T_u is called the tabu tenure). Similarly, element e_v is also marked as tabu for the next T_v iterations and thus cannot be removed from S during this period. The tabu status of an element is disabled if the swap operation with this element leads to a solution better than any already visited solution (this rule is called the *aspiration criterion* in tabu search). An eligible swap move involves only elements that are not forbidden by the tabu list or satisfy the aspiration criterion.

It is important to determine a suitable tabu tenure for the elements of a swap. Yet, there does not exist a universally applicable tabu tenure management scheme. In our algorithm, we adopt a dynamic tabu list management technique which was proposed in [Galinier *et al.*, 2011] and proved to work well for MDP [Wu and Hao, 2013]. The tabu tenure T_x of an element e_x taking part in a swap operation is determined according to a periodic step function $T(iter)$, where $iter$ is the number of iterations. Specifically, $T(iter)$ takes the value of α (a parameter set to 15 in this work), $2 \times \alpha$, $4 \times \alpha$ and $8 \times \alpha$ according to the value of

$iter$, and each $T(iter)$ value is kept for 100 consecutive iterations (see [Galinier *et al.*, 2011; Wu and Hao, 2013] for more details). Following [Wu and Hao, 2013], we set $T_u = T(iter)$ for the element e_u dropped from the solution and $T_v = 0.7 * T(iter)$ for the element e_v added to the solution.

To implement the tabu list, we use an integer vector H of size n whose components are initially set to 0 (i.e., $H[i] = 0, \forall i \in [1, \ldots, n]$). After each $swap(e_u, e_v)$ operation, we set $H[u]$ (resp. $H[v]$) to $iter + T_u$ (resp. $iter + T_v$), where $iter$ is the current number of iterations and T_u (resp. T_v) is the tabu tenure explained above. With this implementation, it is very easy to know whether an element e_i is forbidden by the tabu list as follows. If $iter \leq H[i]$, then e_i is forbidden by the tabu list; otherwise, e_i is not forbidden by the tabu list.

4.3.5 Backbone-based crossover operator

The crossover operator plays a critical role in memetic search and defines the way information is transmitted from parents to offspring [Hao, 2012]. A meaningful crossover operator should preserve good properties of parent solutions through the recombination process. In our case, we adopt a backbone-based crossover operator which generates an offspring solution in the same way as in [Wu and Hao, 2013] while introducing additionally an opposite solution. For MDP, the backbone is a good property that is to be transmitted from parents to offspring, as shown in Definition 3. Specially, the backbone-based crossover operator not only produces an offspring solution, but also creates a corresponding opposite solution.

Definition 3 (**backbone** [Wu and Hao, 2013]). *Let S^u and S^v be two solutions of MDP, the backbone of S^u and S^v is defined as the set of common elements shared by these two solutions, i.e., $S^u \cap S^v$.*

Given a population $P = \{S^1, S^2, \ldots, S^p\}$ of p individuals, an offspring solution is constructed in two phases. The first phase randomly selects two parents S^u and S^v in P and identifies the backbone which is used to form the partial offspring S^o, i.e., $S^o = S^u \cap S^v$. If $|S^o| < m$, then the second phase successively extends S^o with $m - |S^o|$ other elements in a greedy way. Specifically, we alternatively consider each parent and select an unassigned element with maximum gain with respect to S^o until S^o reaches the size of m. Once the offspring solution S^o is obtained, we generate its corresponding opposite solution $\overline{S^o}$ according to Definition 2. Consequently, we obtain two different and distant offspring solutions S^o and $\overline{S^o}$ which are further improved by the tabu search procedure of Section 4.3.4.

4.3.6 Rank-based pool updating strategy

To maintain a healthy diversity of the population, we use a rank-based pool updating strategy to decide whether the improved solutions (S^o and $\overline{S^o}$) should be inserted into the population or discarded. This pool updating strategy simultaneously considers the solution quality and the distance between individuals in the population to guarantee the population diversity. Similar quality-and-distance pool updating strategies have been used in memetic algorithms in [Chen and Hao, 2016; Lü and Hao, 2010; Sörensen and Sevaux, 2006; Wu and Hao, 2013].

For two solutions S^u and S^v, we use the well-known set-theoretic partition distance [Gusfield, 2002] to measure their distance.

$$D(S^u, S^v) = m - Sim(S^u, S^v) \tag{4.10}$$

where $Sim(S^u, S^v) = |S^u \cap S^v|$ denotes the number of common elements shared by S^u and S^v.

Given a population $P = \{S^1, S^2, \ldots, S^p\}$ and one solution S^i in P, the average distance between S^i and the remaining individuals in P is computed by [Chen and Hao, 2016].

$$AD(S^i, P) = \frac{1}{p} \sum_{S^j \in P, j \neq i} D(S^i, S^j) \tag{4.11}$$

To update the population with an improved offspring solution (S^o or $\overline{S^o}$), let us consider the case of S^o (the same procedure is applied to $\overline{S^o}$). We first tentatively insert S^o into the population P, i.e., $P' \leftarrow P \cup \{S^o\}$. Then all individuals in P' are assessed based on the following function.

$$Score(S^i, P') = \beta * RF(f(S^i)) + (1 - \beta) * RF(AD(S^i, P')) \tag{4.12}$$

where $RF(f(S^i))$ and $RF(AD(S^i, P'))$ represent respectively the rank of solution S^i with respect to its objective value and the average distance to the population. Specifically, $RF(\cdot)$ ranks the solutions of P in decreasing order according to their objective values or their average distances to the population. In case of ties, the solution with the smallest index is preferred. β is the weighting coefficient between the objective value of the solution and its average distance to the population, which is empirically set to $\beta = 0.6$.

Based on this scoring function, we identify the worst solution S^w with the largest score value from the population P'. If the worst solution S^w is not the improved offspring S^o, then the population is updated by replacing S^w by S^o; otherwise, S^o is simply discarded.

4.3.7 Computational complexity of OBMA

To analyze the computational complexity of the proposed OBMA algorithm, we consider the main steps in one generation in the main loop of Algorithm 6.

As shown in Algorithm 6, each generation of the OBMA algorithm is composed of four components: parents selection, backbone-based crossover, tabu search and rank-based pool updating strategy. The step of parents selection is bounded by $O(1)$. The backbone-based crossover operation can be achieved in $O(nm^2)$. The computational complexity of the parametric constrained neighborhood search procedure is $O((n + |U_S^c| \times |U_{N\setminus S}^c|)MaxIter)$, where $|U_S^c|$ is the number of elements that can be swapped out from S, $|U_{N\setminus S}^c|$ is the number of elements in $N \setminus S$ that can be swapped into S, and $MaxIter$ is the allowable maximum number of iterations in tabu search. The computational complexity for the pool updating strategy is $O(p(m^2 + p))$, where p is the population size. To summarize, the total computational complexity of the proposed OBMA within one generation is $O(nm^2 + (n + |U_S^c| \times |U_{N\setminus S}^c|)MaxIter)$.

4.4 Computational results

This section presents computational experiments to test the efficiency of our OBMA algorithm. We aim to 1) demonstrate the added value of OBMA (with OBL) compared to the memetic search framework (without OBL), and 2) evaluate the performance of OBMA with respect to the best-known results ever reported by state-of-the-art algorithms in the literature.

4.4.1 Benchmark instances

Our computational assessment were based on 80 large instances with 2000 to 5000 elements which are classified into the following sets.

Set I contains three data sets: MDG-a (also known as Type1_22), MDG-b and MDG-c. They are available at http://www.optsicom.es/mdp/.

— **MDG-a**: This data set consists of 20 instances with $n = 2000$, $m = 200$. The distance d_{ij} between any two elements i and j is an integer number which is randomly selected between 0 and 10 from a uniform distribution.

— **MDG-b**: This data set includes 20 instances with $n = 2000$, $m = 200$. The distance d_{ij} between any two elements i and j is a real number which is randomly selected between 0 and 1000 from a uniform distribution.

— **MDG-c**: This data set is composed of 20 instances with $n = 2000$, $m = 300$, 400, 500, 600. The distance d_{ij} between any two elements i and j is an integer number which is randomly selected between 0 and 1000 from a uniform distribution.

Set II (**b2500**) contains 10 instances with $n = 2500$, $m = 1000$, where the distance d_{ij} between any two elements e_i and e_j is an integer randomly generated from $[-100, 100]$. This data set was originally derived from the unconstrained binary quadratic programming problem by ignoring the diagonal elements and is part of ORLIB: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/.

Set III (**p3000** and **p5000**) contains 5 very large instances with $n = 3000$ and $m = 1500$, and 5 instances with $n = 5000$ and $m = 2500$, where d_{ij} are integers generated from a $[0, 100]$ uniform distribution. The sources of the generator and input files to replicate this data set can be found at: http://www.proin.ktu.lt/~gintaras/mdgp.html.

4.4.2 Experimental settings

Table 4.1: 80 large MDP benchmark instances used in the experiments.

Data set	n	m	#instance	time limit (s)	#run
MDG-a	2000	200	20	20	30
MDG-b	2000	200	20	600	15
MDG-c	2000	300-600	20	600	15
b2500	2500	1000	10	300	30
p3000	3000	1500	5	600	15
p5000	5000	2500	5	1800	15

Our algorithm [2] was implemented in C++, and complied using GNU gcc 4.1.2 with '-O3' option on an Intel E5-2670 with 2.5GHz and 2GB RAM under Linux. Without using any compiler flag, running the DIMACS machine benchmark program dfmax [3] on our machine requires 0.19, 1.17 and 4.54 seconds to solve graphs r300.5, r400.5 and r500.5 respectively. To obtain our experimental results, each instance was solved according to the settings (including time limit and number of runs) provided in Tables 4.1 and 4.2. Notice that, like most reference algorithms of Section 4.4.4, we used a cutoff time limit (instead of fitness evaluations) as the stopping condition. This choice is suitable in the context of MDP given that its fitness evaluation is computationally cheap enough, contrary to expensive-to-evaluate problems like many engineering optimization problems where using fitness evaluations is a standard practice [Jin, 2011].

Table 4.2: Parameter settings of OBMA algorithm.

Parameter	description	value	section
p	population size	10	4.3.3
$MaxIter$	allowable number of iterations of TS	50,000	4.3.4
α	tabu tenure management factor	15	4.3.4
ρ	scale coefficient	4	4.3.4
β	weighting coefficient	0.6	4.3.6

2. The best solution certificates and our program will be made available at http://www.info.univ-angers.fr/pub/hao/OBMA.html.

3. dfmax: ftp://dimacs.rutgers.edu/pub/dsj/clique

4.4.3 Benefit of OBL for memetic search

To verify the benefit of OBL for memetic search, we compare OBMA with its alternative algorithm OBMA_0 without OBL. To obtain OBMA_0, two modifications have been made on OBMA: 1) for the population initialization phase, we randomly generate two initial solutions at a time (instead of a random solution and an opposite solution); 2) for the crossover phase, we perform twice the crossover operation to generate two offspring solutions (instead of one offspring solution and an opposite solution). To make a fair comparison between OBMA and OBMA_0, we ran both algorithms under the same conditions, as shown in Tables 4.1 and 4.2. The comparative results for the five data sets are summarized in Tables 4.3-4.7.

In these tables, columns 1 and 2 respectively show for each instance its name (Instance) and the current best objective value (f_{prev}) jointly reported in recent studies [Martí *et al.*, 2013; Gallego *et al.*, 2009; Wu and Hao, 2013; Wang *et al.*, 2012]. Columns 3-7 report the results of the OBMA_0 algorithm: the difference between f_{prev} and the best objective value f_{best} (i.e., $\Delta f_{best} = f_{prev} - f_{best}$), the difference between f_{prev} and average objective value f_{avg} (i.e., $\Delta f_{avg} = f_{prev} - f_{avg}$), the standard deviation of objective values (σ), the average CPU time to attain the best objective values (t_{best}) and the success rate $(\#succ)$ over 30 or 15 independent runs. Columns 8-12 present the same information of the OBMA algorithm. The best values among the results of the two compared algorithms are indicated in bold. At the last row, we also provide the average number of instances for which one algorithm outperforms the other algorithm. 0.5 is assigned to each compared algorithm in case of ties.

To analyze these results, we resort to a widely-used statistical methodology known as *two-tailed sign test* [Demšar, 2006]. This test is a popular way to compare the overall performance of algorithms by counting the number of winning instances of each compared algorithm and thus to identify the overall winner algorithm. The test makes the null hypothesis that the compared algorithms are equivalent. The null hypothesis is accepted if each algorithm wins on approximately $X/2$ out of X instances. Otherwise, the test rejects the null hypothesis, suggesting a difference between the compared algorithms. The *Critical Values* (CV) for the two-tailed sign test at a significance level of 0.05 are respectively $CV_{0.05}^{20} = 15$ for $X = 20$ instances and $CV_{0.05}^{10} = 9$ for $X = 10$ instances. In other words, algorithm A is significantly better than algorithm B if A performs better than B for at least $CV_{0.05}^{X}$ instances for a data set of X instances.

From Table 4.3 which shows the results of OBMA_0 and OBMA for the 20 MDG-a instances, we first observe that both algorithms attain the best-known results reported in the literature. However, OBMA performs better than OBMA_0 in terms of the average objective value and success rate, and wins 14.5 instances and 13.5 instances respectively. We also observe that the standard deviation of the best objective values is significantly smaller for OBMA, and OBMA wins 14.5 instances, which is very close to the critical value $(CV_{0.05}^{20} = 15)$. Finally, compared to OBMA_0, OBMA needs less average CPU time to find the best-known solutions for all instances except MDG-a_26 and wins 13.5 instances in terms of the success rate.

Table 4.4 shows the results of OBMA_0 and OBMA for the 20 MDG-b instances. The best-known objective values (f_{prev}) of this data set were obtained by a scatter search algorithm (G_SS) [Gallego *et al.*, 2009] with a time limit of 2h on an Intel Core 2 Quad CPU 8300 with 6GB of RAM running Ubuntu 9.04 [Martí *et al.*, 2013]. This table indicates that both OBMA and OBMA_0 find improved best-known solutions for 14 out of 20 instances and attain the best objective values for the remaining 6 instances. On the other hand, compared to the OBMA_0 algorithm, OBMA obtains a better average objective value and higher success rate for 13.5 and 13 instances. It is worth noting that OBMA has a steady performance, and achieves these results with a 100% success rate on almost all instances except for MDG-b_24, MDG-b_32 and MDG-b_33. To summarize, OBMA performs better than OBMA_0 for this data set, but the differences are not very significant at a significance level of 0.05.

Table 4.5 presents the results of OBMA_0 and OBMA for the 20 instances of the MDG-c instances. All best-known results (f_{prev}) were achieved by iterative tabu search (ITS) [Palubeckis, 2007] or variable neighborhood search (VNS) [Brimberg *et al.*, 2009] under a time limit of 2 hours on an Intel Core 2 Quad CPU 8300 with 6GB of RAM running Ubuntu 9.04 [Martí *et al.*, 2013]. We observe that both OBMA and OBMA_0 obtain improved best-known solutions for 8 instances and match the best-known solutions for 4 instances. In fact, OBMA improves all best-known solutions obtained by VNS, but it fails to attain 8

Table 4.3: Comparison of the results obtained by $OBMA_0$ and OBMA on the data set MDG-a.

		$OBMA_0$					OBMA				
Instance	f_{prev}	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ
MDG-a_21	114271	**0**	**4.5**	**10.6**	9.3	**22/30**	**0**	5.2	10.8	**7.4**	20/30
MDG-a_22	114327	**0**	4.2	22.6	9.2	**29/30**	**0**	**0.1**	**0.5**	**7.2**	**29/30**
MDG-a_23	114195	**0**	**10.9**	15.2	11.5	**15/30**	**0**	15.2	**15.1**	**7.0**	11/30
MDG-a_24	114093	**0**	25.3	21.0	11.3	4/30	**0**	**11.2**	**12.1**	**8.2**	**7/30**
MDG-a_25	114196	**0**	55.9	32.7	13.3	1/30	**0**	**41.5**	**30.7**	**8.8**	**5/30**
MDG-a_26	114265	**0**	**7.3**	**10.2**	**9.9**	**17/30**	**0**	**10.2**	11.5	11.6	14/30
MDG-a_27	114361	**0**	**0.0**	**0.0**	4.8	**30/30**	**0**	0.2	0.9	**4.1**	29/30
MDG-a_28	114327	**0**	57.1	53.8	15.2	12/30	**0**	**18.9**	**39.2**	**7.9**	**23/30**
MDG-a_29	114199	**0**	11.2	16.0	14.6	8/30	**0**	**4.4**	**8.4**	**9.0**	**14/30**
MDG-a_30	114229	**0**	12.8	16.3	10.5	**14/30**	**0**	**8.1**	**10.5**	**7.2**	**14/30**
MDG-a_31	114214	**0**	30.4	22.7	16.2	5/30	**0**	**16.7**	**13.6**	**11.9**	**9/30**
MDG-a_32	114214	**0**	28.5	19.9	10.4	**4/30**	**0**	**23.7**	**17.1**	**6.0**	3/30
MDG-a_33	114233	**0**	6.1	10.5	12.8	15/30	**0**	**2.0**	**5.6**	**8.8**	**23/30**
MDG-a_34	114216	**0**	25.4	43.8	11.6	15/30	**0**	**2.4**	**7.0**	**6.6**	**26/30**
MDG-a_35	114240	**0**	**1.6**	**2.2**	12.2	9/30	**0**	**1.6**	2.4	**10.7**	**11/30**
MDG-a_36	114335	**0**	7.5	11.7	12.4	19/30	**0**	**5.7**	**9.5**	**10.5**	**21/30**
MDG-a_37	114255	**0**	**4.2**	**8.2**	12.2	**18/30**	**0**	5.2	8.6	**7.8**	**18/30**
MDG-a_38	114408	**0**	1.2	3.1	10.6	19/30	**0**	**0.5**	**1.1**	**7.6**	**25/30**
MDG-a_39	114201	**0**	2.2	**6.0**	9.7	26/30	**0**	**2.0**	**6.0**	**4.9**	**27/30**
MDG-a_40	114349	**0**	28.1	37.7	9.9	18/30	**0**	**23.0**	**31.5**	**9.1**	**19/30**
	wins	10	5.5	5.5	1	6.5	10	14.5	14.5	19	13.5

The f_{prev} values were obtained by several algorithms including LTS-EDA [Wang *et al.*, 2012] and MAMDP [Wu and Hao, 2013].

best-known solutions found by ITS. Compared to $OBMA_0$, OBMA obtains 2 improved best solutions for MDG-c_17 and MDG-c_19. Moreover, OBMA performs significantly better than $OBMA_0$ in terms of the average best solution $(19 > CV^{20}_{0.05} = 15)$, success rate $(15 >= CV^{20}_{0.05} = 15)$ and standard deviation $(19 > CV^{20}_{0.05} = 15)$ at a significance level of 0.05.

Table 4.6 reports the results of $OBMA_0$ and OBMA for the 10 instances of the b2500 data set. From this table, we observe that both algorithms reach the best-known values for all the instances. Meanwhile, the average value of best objective values of OBMA is better than that of $OBMA_0$, and the difference of this measure between these two algorithms is weakly significant $(8.5 < CV^{10}_{0.05} = 9)$. Even though there is no significant difference on the success rate, OBMA obtains a higher success rate for 8.5 instances, while the reverse is true only for 1.5 instances. In addition, OBMA achieves these results more steadily than $OBMA_0$, wining 8.5 out of 10 instances in terms of the standard deviation.

Table 4.7 displays the results of $OBMA_0$ and OBMA for the 10 largest instances (p3000 and p5000 instances). For these very large instances, OBMA matches all the best-known objective values without exception while $OBMA_0$ fails to do so for 4 instances. In addition, OBMA performs significantly better than $OBMA_0$, and wins 10, 9.5 instances in terms of the average best objective value and success rate, respectively. The performance of OBMA is also more stable than $OBMA_0$ by wining 8 out of 10 instances in term of the standard deviation.

Finally, Table 4.8 provides a summary of the comparative results for the five data sets between OBMA (OBL enhanced memetic algorithm) and $OBMA_0$ (memetic algorithm without OBL). As we observe from the table, OBMA achieves a better performance than $OBMA_0$, i.e., achieving improved solutions for 6 instances and matching the best solutions on the remaining 75 instances. In addition, OBMA also achieves a better performance in terms of the average best value, the success rate and the standard deviation, wining $OBMA_0$ on most benchmark instances. Therefore, we conclude that opposition-based learning can beneficially enhance the popular memetic search framework to achieve an improved performance.

Table 4.4: Comparison of the results obtained by $OBMA_0$ and OBMA on the data set MDG-b.

		$OBMA_0$					OBMA				
Instance	f_{prev}	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ
MDG-b_21	11299895	**0**	0.2	**0.0**	378.6	**15/15**	**0**	0.2	**0.0**	**325.1**	**15/15**
MDG-b_22	11286776	**-5622**	**-5622.2**	**0.0**	**336.5**	**15/15**	**-5622**	**-5622.2**	**0.0**	380.8	**15/15**
MDG-b_23	11299941	**0**	**0.5**	**0.0**	300.5	**15/15**	**0**	**0.5**	**0.0**	**283.4**	**15/15**
MDG-b_24	11290874	**-245**	**-229.1**	**61.2**	345.5	**14/15**	**-245**	-220.5	67.2	**323.4**	13/15
MDG-b_25	11296067	**-1960**	**-1959.9**	**0.0**	**271.2**	**15/15**	**-1960**	**-1959.9**	**0.0**	313.9	**15/15**
MDG-b_26	11292296	**-6134**	-5216.0	1836.9	**276.8**	12/15	**-6134**	**-6134.4**	**0.0**	336.0	**15/15**
MDG-b_27	11305677	**0**	**0.2**	**0.0**	330.0	**15/15**	**0**	**0.2**	**0.0**	**256.0**	**15/15**
MDG-b_28	11279916	**-2995**	-2994.6	0.5	**329.5**	10/15	**-2995**	**-2994.7**	**0.4**	351.1	**12/15**
MDG-b_29	11297188	**-151**	**-151.5**	**0.0**	323.0	**15/15**	**-151**	**-151.5**	**0.0**	**288.3**	**15/15**
MDG-b_30	11296415	**-1650**	**-1649.6**	**0.0**	311.2	**15/15**	**-1650**	**-1649.6**	**0.0**	**274.9**	**15/15**
MDG-b_31	11288901	**0**	**-0.2**	**0.0**	313.9	**15/15**	**0**	**-0.2**	**0.0**	**308.0**	**15/15**
MDG-b_32	11279820	**-3719**	-3669.3	**25.0**	**177.5**	3/15	**-3719**	**-3694.3**	30.6	283.0	**9/15**
MDG-b_33	11296298	**-1740**	-1381.7	216.1	**112.2**	4/15	**-1740**	**-1675.0**	**166.1**	277.5	**13/15**
MDG-b_34	11281245	**-9238**	-8881.8	435.8	**325.7**	9/15	**-9238**	**-9237.6**	**0.0**	355.4	**15/15**
MDG-b_35	11307424	**0**	**-0.1**	**0.0**	343.6	**15/15**	**0**	**-0.1**	**0.0**	**331.0**	**15/15**
MDG-b_36	11289469	**-13423**	-13174.5	929.8	**251.7**	14/15	**-13423**	**-13423.0**	**0.0**	329.4	**15/15**
MDG-b_37	11290545	**-5229**	-5099.5	329.7	**217.5**	13/15	**-5229**	**-5228.8**	**0.0**	291.2	**15/15**
MDG-b_38	11288571	**-7965**	**-7964.5**	**0.0**	**242.5**	**15/15**	**-7965**	**-7964.5**	**0.0**	297.6	**15/15**
MDG-b_39	11295054	**0**	**-0.2**	**0.0**	374.5	**15/15**	**0**	**-0.2**	**0.0**	**289.7**	**15/15**
MDG-b_40	11307105	**-2058**	**-2057.6**	**0.0**	301.1	**15/15**	**-2058**	**-2057.6**	**0.0**	**266.5**	**15/15**
	wins	10	6.5	8	10	7	10	13.5	12	10	13

The best-known values f_{prev} were obtained by a scatter search algorithm (G_SS) [Gallego *et al.*, 2009] with a time limit of 2 hours, which are available at `http://www.optsicom.es/mdp/`.

4.4.4 Comparison with state-of-the-art algorithms

We turn now our attention to a comparison of our OBMA algorithm with state-of-the-art algorithms, including iterated tabu search (ITS) [Palubeckis, 2007], scatter search (G_SS) [Gallego *et al.*, 2009], variable neighborhood search (VNS) [Brimberg *et al.*, 2009], fine-tuning iterated greedy algorithm (TIG) [Lozano *et al.*, 2011], tabu search with estimation of distribution algorithm (LTS-EDA) [Wang *et al.*, 2012] and memetic algorithm (MAMDP) [Wu and Hao, 2013]. We omit the tabu search/memetic algorithm (TS/MA) [Wang *et al.*, 2014] and the memetic self-adaptive evolution strategies (MSES) [de Freitas *et al.*, 2014] since TS/MA performs quite similar to MAMDP of [Wu and Hao, 2013] while MSES does not report detailed results. Among these reference algorithm, only the program of the memetic algorithm (MAMDP) [Wu and Hao, 2013] is available. For our comparative study, we report the results of the MAMDP algorithm by running its code on our platform with its default parameter values reported in [Wu and Hao, 2013]. For the other reference algorithms, we use their results presented in the corresponding references. The detailed comparative results in terms of Δf_{best} and Δf_{avg} are reported in Tables 4.9 and 4.10.

Table 4.9 presents the comparative results on the 40 instances of the data sets MDG-a, b2500, p3000-p5000 for which the detailed results of reference algorithms are available. At the last row of the table, we also indicate the number of wining instances relative to our OBMA algorithm both in terms of the best objective value and average objective value (recall that a tied result counts 0.5 for each algorithm). From this table, we observe that OBMA dominates all the reference algorithms. Importantly, OBMA is the only algorithm which obtains the best-known values and the largest average objective values for all 40 instances.

Table 4.10 displays the comparative results on the data sets MDG-b and MDG-c. The best-known objective values f_{prev} for the MDG-b instances are obtained by G_SS [Gallego *et al.*, 2009] while the f_{prev} values of the MDG-c instances are obtained by ITS and VNS [Martí *et al.*, 2013], both with a time limit of 2 hours. No result is available for the TIG and LTS-EDA algorithms for these data sets. The results of our OBMA algorithm (and MAMDP) are obtained with a time limit of 10 minutes. Table 4.10 indicates

Table 4.5: Comparison of the results obtained by $OBMA_0$ and OBMA on the data set MDG-c.

		$OBMA_0$					OBMA				
Instance	f_{prev}	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ
MDG-c_1	24924685*	**-1659**	-493.5	852.7	**165.5**	5/15	**-1659**	**-1262.4**	**749.0**	251.3	**11/15**
MDG-c_2	24909199*	**-3347**	140.3	2514.6	**94.6**	4/15	**-3347**	**-3346.3**	**2.5**	286.6	**14/15**
MDG-c_3	24900820*	**-4398**	-299.2	4040.7	**22.5**	7/15	**-4398**	**-2805.2**	**2717.9**	239.4	**11/15**
MDG-c_4	24904964*	**-4746**	-1890.5	1999.4	**106.5**	4/15	**-4746**	**-3917.2**	1657.6	276.2	**12/15**
MDG-c_5	24899703*	**3999**	4767.8	1025.0	**8.7**	9/15	**3999**	**4047.3**	**180.6**	212.5	**14/15**
MDG-c_6	43465087*	**20139**	22534.4	2512.3	**77.4**	**6/15**	**20139**	**21054.5**	**1190.4**	290.8	**6/15**
MDG-c_7	43477267*	**0**	277.5	1038.2	**6.1**	**14/15**	**0**	**126.9**	**314.7**	111.7	12/15
MDG-c_8	43458007*	**-7565**	-4644.3	1833.2	**58.9**	3/15	**-7565**	**-7546.7**	**68.6**	163.6	**14/15**
MDG-c_9	43448137*	**0**	142.2	116.1	82.1	6/15	**0**	**0.0**	**0.0**	**72.5**	**15/15**
MDG-c_10	43476251*	**10690**	**10690.0**	**0.0**	**27.1**	**15/15**	**10690**	**10690.0**	**0.0**	115.1	**15/15**
MDG-c_11	67009114*	**-12018**	-11345.3	2110.0	**90.8**	**13/15**	**-12018**	**-11776.3**	**522.3**	335.0	10/15
MDG-c_12	67021888*	**7718**	12209.1	5502.4	**9.3**	7/15	**7718**	**10179.7**	**3250.5**	302.8	**9/15**
MDG-c_13	67024373*	**0**	2082.0	2944.8	**106.4**	10/15	**0**	**839.4**	**2140.1**	380.1	**13/15**
MDG-c_14	67024804*	**-5386**	-4667.9	1830.9	**11.3**	13/15	**-5386**	**-5118.7**	**1000.3**	276.3	**14/15**
MDG-c_15	67056334*	**0**	1846.5	1353.5	**31.0**	**5/15**	**0**	**1021.2**	**1122.0**	269.0	**5/15**
MDG-c_16	95637733*	**-1196**	5861.5	8193.7	318.8	2/15	**-1196**	**-1116.3**	**298.3**	**270.8**	14/15
MDG-c_17	95645826*	75241	86848.9	8727.7	**291.8**	2/15	**74713**	**74981.7**	**373.3**	312.8	**8/15**
MDG-c_18	95629207*	**97066**	100609.9	3526.8	**90.5**	7/15	**97066**	**99767.0**	**2972.8**	292.1	**8/15**
MDG-c_19	95633549*	35131	39027.5	5420.3	**236.5**	**7/15**	**34385**	**35121.3**	**816.2**	343.7	4/15
MDG-c_20	95643586*	**59104**	**59133.2**	**109.3**	**111.0**	**14/15**	**59104**	**59133.2**	**109.3**	299.9	**14/15**
	wins	9	1	1	18	5	11	19	19	2	15

* Results are obtained by ITS with 2 hours CPU time [Martí *et al.*, 2013].
* Results are obtained by VNS with 2 hours CPU time [Martí *et al.*, 2013].

Table 4.6: Comparison of the results obtained by $OBMA_0$ and OBMA on the data set b2500.

		$OBMA_0$					OBMA				
Instance	f_{prev}	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ
b2500-1	1153068	**0**	193.1	429.2	154.1	23/30	**0**	**0.0**	**0.0**	**100.3**	**30/30**
b2500-2	1129310	**0**	106.0	163.1	149.2	21/30	**0**	**37.9**	**73.2**	**147.4**	**22/30**
b2500-3	1115538	**0**	303.2	347.2	105.5	17/30	**0**	**0.4**	**2.2**	**95.3**	**29/30**
b2500-4	1147840	**0**	549.7	461.9	191.2	8/30	**0**	**65.8**	**118.5**	**98.9**	**20/30**
b2500-5	1144756	**0**	50.9	129.3	117.6	24/30	**0**	**5.3**	**28.4**	**86.3**	**29/30**
b2500-6	1133572	**0**	88.9	210.9	89.4	22/30	**0**	**0.0**	**0.0**	**66.8**	**30/30**
b2500-7	1149064	**0**	106.2	111.9	**114.8**	13/30	**0**	**14.1**	**31.0**	128.3	**23/30**
b2500-8	1142762	**0**	113.7	349.0	**98.4**	22/30	**0**	**1.5**	**5.5**	105.4	**28/30**
b2500-9	1138866	**0**	**0.2**	**1.1**	**135.7**	**29/30**	**0**	1.3	2.9	139.8	25/30
b2500-10	1153936	**0**	**0.0**	**0.0**	**81.4**	**30/30**	**0**	**0.0**	**0.0**	107.5	**30/30**
	wins	5	1.5	1.5	4	1.5	5	8.5	8.5	6	8.5

The f_{prev} values were compiled from the results reported by ITS [Palubeckis, 2007], LTS-EDA [Wang *et al.*, 2012] and MAMDP [Wu and Hao, 2013].

that both OBMA and MAMDP improve the best-known results for the majority of the 40 instances. Moreover, compared to MAMDP, our OBMA algorithm obtains an improved best objective value for 1 MDG-b instance and 3 MDG-c instances, while matching the best objective values for the remaining instances. Finally, OBMA dominates MAMDP in terms of the average objective value, wining 18 out of the 20 MDG-b instances and all 20 MDG-c instances.

To summarize, compared to the state-of-the-art results, our OBMA algorithm finds improved best-known solutions (new lower bounds) for 22 out of the 80 benchmark instances, matches the best-known solutions for 50 instances, but fails to attain the best-known results for 8 instances. Such a performance in-

Table 4.7: Comparison of the results obtained by $OBMA_0$ and OBMA on the data sets p3000 and p5000.

		$OBMA_0$					OBMA				
Instance	f_{prev}	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ
p3000_1	6502330	**0**	84.1	**28.0**	**172.6**	1/15	**0**	**24.4**	35.8	275.6	**9/15**
p3000_2	18272568	**0**	152.8	151.1	151.5	7/15	**0**	**0.0**	**0.0**	**89.3**	**15/15**
p3000_3	29867138	**0**	544.5	344.2	244.0	4/15	**0**	**0.0**	**0.0**	**26.2**	**15/15**
p3000_4	46915044	**0**	715.0	531.0	**250.1**	2/15	**0**	**1.2**	**19.3**	336.9	**14/15**
p3000_5	58095467	**0**	209.9	198.2	180.0	6/15	**0**	**0.0**	**0.0**	**65.4**	**15/15**
p5000_1	17509369	**0**	168.9	**176.5**	**518.7**	7/15	**0**	**128.2**	181.8	1053.9	**13/15**
p5000_2	50103092	70	819.1	494.3	**242.5**	**1/15**	**0**	**22.8**	**8.0**	370.8	**1/15**
p5000_3	82040316	176	3450.3	1671.3	333.1	1/15	**0**	**209.3**	**141.3**	**217.1**	**2/15**
p5000_4	129413710	598	1460.1	661.6	1019.1	1/15	**0**	**97.8**	**122.1**	**625.7**	**7/15**
p5000_5	160598156	344	669.6	323.6	1348.7	1/15	**0**	**102.9**	**52.3**	**843.2**	**5/15**
	wins	3	0	2	4	0.5	7	10	8	6	9.5

The best-known values f_{prev} were extracted from [Wu and Hao, 2013].

Table 4.8: A summary of win statistical results ($OBMA_0$ | OBMA) on all data sets.

Data set	Δf_{best}	Δf_{avg}	σ	t_{best}	#succ
MDG-a	10 | 10	5.5 | 14.5	5.5 | 14.5	1 | 19	6.5 | 13.5
MDG-b	10 | 10	6.5 | 13.5	8 | 12	10 | 10	7 | 13
MDG-c	9 | 11	1 | 19	1 | 19	18 | 2	5 | 15
b2500	5 | 5	1.5 | 8.5	1.5 | 8.5	4 | 6	1.5 | 8.5
p3000-5000	3 | 7	0 | 10	2 | 8	4 | 6	0.5 | 9.5

dicates that the proposed algorithm competes favorably with state-of-the-art MDP algorithms and enriches the existing solution arsenal for solving MDP.

4.5 Experimental analysis

In this section, we perform additional experiments to gain some understanding of the proposed algorithm including the parametric constrained neighborhood, the rank-based quality-and-distance pool management and the benefit of OBL for population diversity.

4.5.1 Study of the parametric constrained neighborhood

Our tabu search procedure relies on the parametric constrained neighborhood whose size is controlled by the parameter ρ. To highlight the effect of this parameter and determine a proper value, we ran the tabu search procedure to solve the first 10 instances of MDG-a (i.e., MDG-a_21- MDG-a_30) with $\rho \in [1, 10]$. Each instance was independently solved until the number of iterations reached $MaxIter$. Figure 4.4 shows the average objective values achieved (left) and the average CPU times consumed (right) by tabu search on these 10 instances.

As we see from Figure 4.4 (left), the average objective value has a drastic rise when we increase ρ from 1 to 3. Then, it slowly increases if we continue to increase ρ to 10. On Figure 4.4 (right), the average CPU time of tabu search needed to finish $MaxIter$ iterations continuously increases when ρ increases from 1 to 10. As ρ increases, the size of the constrained neighborhood also increases, thus the algorithm needs more time to examine the candidate solutions. To make a compromise between neighborhood size and solution quality, we set the scale coefficient ρ to 4 in our experiments.

Table 4.9: Comparison of OBMA with other algorithms on the data sets MDG-a, b2500, p3000 and p5000.

		ITS		VNS		TIG		LTS-EDA		MAMDP		OBMA	
Instance	f_{prev}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}
MDG-a_21	114271	65	209.9	48	150.6	48	101.6	5	60.7	**0**	8.1	**0**	**5.2**
MDG-a_22	114327	29	262.3	**0**	168.9	**0**	69.9	**0**	89.9	**0**	8.8	**0**	**0.1**
MDG-a_23	114195	69	201.4	19	110.8	5	117.8	**0**	99.0	**0**	**15.2**	**0**	**15.2**
MDG-a_24	114093	22	200.5	70	188.1	58	141.9	**0**	79.9	**0**	15.7	**0**	**11.2**
MDG-a_25	114196	95	273.3	87	184.1	99	194.7	51	134.5	**0**	42.1	**0**	**41.5**
MDG-a_26	114265	41	168.2	30	99.3	9	96.2	**0**	40.2	**0**	10.8	**0**	**10.2**
MDG-a_27	114361	12	167.5	**0**	56.3	**0**	71.3	**0**	18.2	**0**	**0.0**	**0**	0.2
MDG-a_28	114327	25	256.4	**0**	163.3	**0**	193.6	**0**	159.1	**0**	20.9	**0**	**18.9**
MDG-a_29	114199	9	139.8	16	78.5	16	80.4	**0**	71.0	**0**	7.6	**0**	**4.4**
MDG-a_30	114229	24	204.9	7	139.3	35	121.4	**0**	56.2	**0**	9.3	**0**	**8.1**
MDG-a_31	114214	74	237.8	42	145.1	59	139.6	3	69.9	**0**	17.8	**0**	**16.7**
MDG-a_32	114214	55	249.5	95	143.3	88	156.0	15	84.9	**0**	26.8	**0**	**23.7**
MDG-a_33	114233	93	279.9	22	168.1	42	167.4	6	85.3	**0**	3.6	**0**	**2.0**
MDG-a_34	114216	92	248.5	117	194.3	64	202.8	**0**	81.0	**0**	3.4	**0**	**2.4**
MDG-a_35	114240	11	117.5	1	62.9	6	80.5	**0**	22.0	**0**	**1.2**	**0**	1.6
MDG-a_36	114335	11	225.4	42	215.4	35	167.9	**0**	36.5	**0**	8.6	**0**	**5.7**
MDG-a_37	114255	56	217.5	**0**	170.0	18	144.5	6	57.1	**0**	6.5	**0**	**5.2**
MDG-a_38	114408	46	170.0	**0**	57.1	2	117.4	2	22.8	**0**	0.7	**0**	**0.5**
MDG-a_39	114201	34	243.2	**0**	124.6	**0**	144.4	**0**	35.9	**0**	3.4	**0**	**2.0**
MDG-a_40	114349	151	270.7	65	159.4	45	187.2	**0**	95.4	**0**	24.1	**0**	**23.0**
b2500-1	1153068	624	3677.3	96	1911.9	42	1960.3	**0**	369.2	**0**	72.1	**0**	**0.0**
b2500-2	1129310	128	1855.3	88	1034.3	1096	1958.5	154	454.5	**0**	143.7	**0**	**37.9**
b2500-3	1115538	316	3281.9	332	1503.7	34	2647.9	**0**	290.4	**0**	184.5	**0**	**0.4**
b2500-4	1147840	870	2547.9	436	1521.1	910	1937.1	**0**	461.7	**0**	152.3	**0**	**65.8**
b2500-5	1144756	356	1800.3	**0**	749.4	674	1655.9	**0**	286.1	**0**	10.5	**0**	**5.3**
b2500-6	1133572	250	2173.5	**0**	1283.5	964	1807.6	80	218.0	**0**	80.5	**0**	**0.0**
b2500-7	1149064	306	1512.6	116	775.5	76	1338.7	44	264.6	**0**	45.0	**0**	**14.1**
b2500-8	1142762	**0**	2467.7	96	862.5	588	1421.5	22	146.5	**0**	1.7	**0**	**1.5**
b2500-9	1138866	642	2944.7	54	837.1	658	1020.6	6	206.3	**0**	3.7	**0**	**1.3**
b2500-10	1153936	598	2024.6	278	1069.4	448	1808.7	94	305.3	**0**	**0.0**	**0**	**0.0**
p3000-1	6502330	466	1487.5	273	909.8	136	714.7	96	294.1	**0**	76.7	**0**	**24.4**
p3000-2	18272568	**0**	1321.6	**0**	924.2	**0**	991.1	140	387.0	**0**	146.1	**0**	**0.0**
p3000-3	29867138	1442	2214.7	328	963.5	820	1166.1	**0**	304.3	**0**	527.9	**0**	**0.0**
p3000-4	46915044	1311	2243.9	254	1068.5	426	2482.2	130	317.1	**0**	399.5	**0**	**1.2**
p3000-5	58095467	423	1521.6	**0**	663.0	278	1353.3	**0**	370.4	**0**	210.7	**0**	**0.0**
p5000-1	17509369	2200	3564.9	1002	1971.3	1154	2545.8	191	571.0	**0**	165.1	**0**	**128.2**
p5000-2	50103092	2931	4807.8	1499	2640.0	549	2532.7	547	913.8	21	475.5	**0**	**22.8**
p5000-3	82040316	5452	8242.3	1914	3694.4	2156	6007.1	704	1458.5	176	1419.0	**0**	**209.3**
p5000-4	129413710	1630	5076.9	1513	2965.9	1696	3874.8	858	1275.2	279	800.9	**0**	**97.8**
p5000-5	160598156	2057	4433.9	1191	2278.3	1289	2128.9	579	1017.9	136	411.9	**0**	**102.9**
wins		1	0	5	0	2.5	0	9.5	0	18	2		

The f_{prev} values were compiled from the results reported by the reference methods [Palubeckis, 2007; Brimberg *et al.*, 2009; Lozano *et al.*, 2011; Wang *et al.*, 2012; Wu and Hao, 2013]. The results of MAMDP are those we obtained by running its program on our computer, which are slightly different from the results reported in [Wu and Hao, 2013] due to the stochastic nature of the algorithm.

4.5.2 Effectiveness of the pool updating strategy

To validate the effectiveness of the *Rank-Based Quality-and-Distance (RBQD)* pool updating strategy, we compare it with the *General Quality-and-Distance (GQD)* pool updating strategy used in [Wu and Hao, 2013]. GQD evaluates each individual by a weighted sum of the quality and the distance to the population. In this experiment, we compared the performance of the OBMA algorithm under these two pool updating strategies (the two OBMA variants are called OBMA$_{RBQD}$ and OBMA$_{GQD}$). The experiment was performed on the largest data set, i.e., p3000 and p5000. We performed 20 runs of each algorithm to solve each instance, and recorded the best objective value (f_{best}), the difference between the average objective value and the best objective value (Δf_{avg}), the standard deviation of objective value over each run (σ), the average time of one run (t_{avg}), the average time over the runs which attained f_{best} (t_{best}), and the success rate ($\#succ$).

Table 4.11 shows the comparison of the results obtained by OBMA under the rank-based quality-and-distance strategy (OBMA$_{RBQD}$) and the general quality-and-distance strategy (OBMA$_{GQD}$). From the table, we observe that OBMA$_{RBQD}$ achieves the same best objective values for all tested instances compared with OBMA$_{GQD}$. However, for the five metrics, OBMA$_{RBQD}$ performs better than OBMA$_{GQD}$ for much

Table 4.10: Comparison of OBMA with MAMDP on the data sets MDG-b and MDG-c, the best-known results are obtained by G_SS, ITS and VNS.

		MAMDP		OBMA				MAMDP		OBMA	
Instance	f_{prev}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}	Instance	f_{prev}	Δf_{best}	Δf_{avg}	Δf_{best}	Δf_{avg}
MDG-b_21	11299895	**0**	225.8	**0**	**0.2**	MDG-c_1	24924685	**-1659**	3481.7	**-1659**	**-1262.4**
MDG-b_22	11286776	**-5622**	-3472.1	**-5622**	**-5622.2**	MDG-c_2	24909199	0	4938.7	**-3347**	**-3346.3**
MDG-b_23	11299941	**0**	**0.5**	**0**	**0.5**	MDG-c_3	24900820	**-4398**	5206.1	**-4398**	**-2805.2**
MDG-b_24	11290874	**-245**	226.0	**-245**	**-220.5**	MDG-c_4	24904964	**-4746**	-411.2	**-4746**	**-3917.2**
MDG-b_25	11296067	**-1960**	-1888.9	**-1960**	**-1959.9**	MDG-c_5	24899703	**3999**	7500.3	**3999**	**4047.3**
MDG-b_26	11292296	**-6134**	-2530.6	**-6134**	**-6134.4**	MDG-c_6	43465087	**20139**	25023.7	**20139**	**21054.5**
MDG-b_27	11305677	**0**	**0.2**	**0**	**0.2**	MDG-c_7	43477267	**0**	1020.8	**0**	**126.9**
MDG-b_28	11279916	-2994	-2634.6	**-2995**	**-2994.7**	MDG-c_8	43458007	-4568	-1329.9	**-7565**	**-7546.7**
MDG-b_29	11297188	**-151**	451.8	**-151**	**-151.5**	MDG-c_9	43448137	237	1207.3	**0**	**0.0**
MDG-b_30	11296415	**-1650**	**-1649.6**	**-1650**	**-1649.6**	MDG-c_10	43476251	**10690**	11060.9	**10690**	**10690.0**
MDG-b_31	11288901	**0**	375.7	**0**	**-0.2**	MDG-c_11	67009114	**-12018**	-6942.7	**-12018**	**-11776.3**
MDG-b_32	11279820	**-3719**	-3632.3	**-3719**	**-3694.3**	MDG-c_12	67021888	**7718**	17470.0	**7718**	**10179.7**
MDG-b_33	11296298	**-1740**	-878.7	**-1740**	**-1675.0**	MDG-c_13	67024373	**0**	6673.1	**0**	**839.4**
MDG-b_34	11281245	**-9238**	-8191.3	**-9238**	**-9237.6**	MDG-c_14	67024804	**-5386**	-1050.9	**-5386**	**-5118.7**
MDG-b_35	11307424	**0**	**-0.1**	**0**	**-0.1**	MDG-c_15	67056334	**0**	3716.2	**0**	**1021.2**
MDG-b_36	11289469	**-13423**	-10792.5	**-13423**	**-13423.0**	MDG-c_16	95637733	**-1196**	1495.2	**-1196**	**-1116.3**
MDG-b_37	11290545	**-5229**	-4372.1	**-5229**	**-5228.8**	MDG-c_17	95645826	**74713**	79061.1	**74713**	**74981.7**
MDG-b_38	11288571	**-7965**	-5896.0	**-7965**	**-7964.5**	MDG-c_18	95629207	**97066**	106806.6	**97066**	**99767.0**
MDG-b_39	11295054	**0**	472.4	**0**	**-0.2**	MDG-c_19	95633549	**34385**	36189.1	**34385**	**35121.3**
MDG-b_40	11307105	**-2058**	-517.5	**-2058**	**-2057.6**	MDG-c_20	95643586	**59104**	61961.2	**59104**	**59133.2**
wins		9.5	2	10.5	18	wins		8.5	0	11.5	20

The f_{prev} values for the MDG-b instances are reported by G_SS [Gallego *et al.*, 2009], while the f_{prev} values for the MDG-c instances are from [Martí *et al.*, 2013] with a time limit of 2 hours, all available at `http://www.optsicom.es/mdp/`. The results of MAMDP were obtained by running the program on our computer (results of MAMDP for these instances are not reported in [Wu and Hao, 2013]).

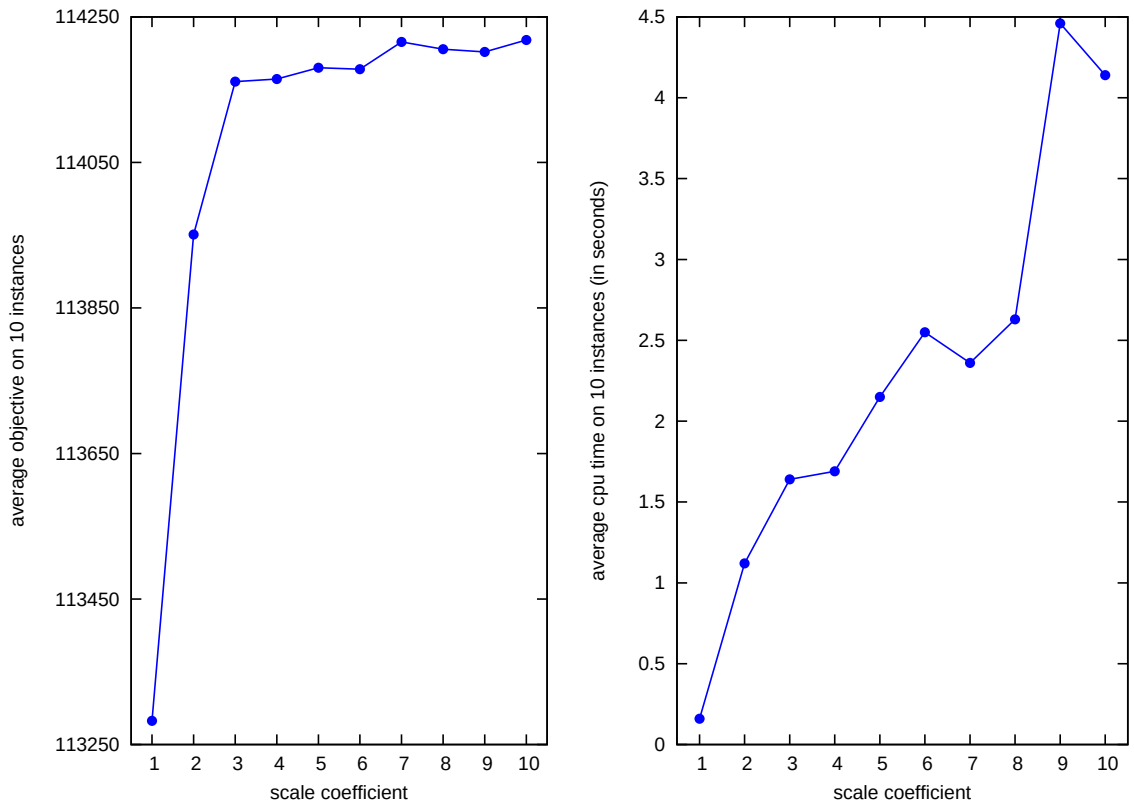

Figure 4.4: Average objective values and average CPU times spent on 10 MDG-a instances obtained by executing TS with different values of the scale coefficient ρ.

more instances, and respectively winning 8, 8, 6, 6 and 8 out of 10 tested instances. These results confirm the effectiveness of our proposed rank-based quality-and-distance pool updating strategy.

Table 4.11: Comparison of the results obtained by OBMA under the rank-based quality-and-distance pool updating strategy (OBMA$_{RBQD}$) and the general quality-and-distance (GQD) pool updating strategy (OBMA$_{GQD}$).

	OBMA$_{RBQD}$						OBMA$_{GQD}$					
Instance	f_{best}	Δf_{avg}	σ	t_{avg}	t_{best}	#succ	f_{best}	Δf_{avg}	σ	t_{avg}	t_{best}	#succ
p3000_1	6502330	-23.0	35.3	176.9	118.8	14/20	6502330	-27.3	37.4	158.3	81.4	13/20
p3000_2	18272568	0.0	0.0	75.8	75.8	20/20	18272568	-10.5	45.8	54.7	57.1	19/20
p3000_3	29867138	0.0	0.0	37.7	37.7	20/20	29867138	0.0	0.0	55.4	55.4	20/20
p3000_4	46915044	0.0	0.0	113.7	113.7	20/20	46915044	-0.9	3.9	147.5	146.4	19/20
p3000_5	58095467	0.0	0.0	22.9	22.9	20/20	58095467	0.0	0.0	92.8	92.8	20/20
p5000_1	17509369	-13.8	60.4	621.9	624.3	19/20	17509369	-27.8	83.1	674.3	646.1	17/20
p5000_2	50103071	-23.4	4.8	561.1	594.3	16/20	50103071	-26.4	6.0	584.3	464.0	11/20
p5000_3	82040316	-305.8	304.2	791.1	527.5	01/20	82040316	-241.0	176.8	642.2	718.4	01/20
p5000_4	129413710	-116.4	143.9	756.5	802.8	12/20	129413710	-174.3	176.2	662.5	705.2	09/20
p5000_5	160598156	-161.8	99.4	511.7	471.6	02/20	160598156	-182.6	112.7	745.5	1081.3	02/20
wins	5	8	8	6	6	8	5	2	2	4	4	2

4.5.3 Opposition-based learning over population diversity

In this section, we further verify the benefit brought by OBL in maintaining the population diversity of the OBMA algorithm. To assess the diversity of a population, a suitable metric is necessary. In this experiment, we resort to *minimum distance* and *average distance* of individuals in the population to measure the population diversity. The minimum distance is defined as the minimum distance between any two individuals in the population, i.e., $MD = \min_{i \neq j \in \{1,2,\ldots,p\}} D(S^i, S^j)$. Correspondingly, the AD is the average distance between all individuals in the population, as defined by Equation (4.11).

Using the data sets MDG-a and b2500, we compared the diversity of the population with or without OBL. The population initialization (PI_0) procedure without OBL first generates two random solutions, which are then respectively improved by the tabu search procedure. The best of two improved solutions is inserted into the population if it does not duplicate any existing individual in the population. We repeat this process until p different solutions are generated. In contrast, the population initialization with OBL (PI_{OBL}) is the procedure described in Section 4.3.3, which considers both a random solution and its corresponding opposite solution. We solved each instance 20 times and recorded the minimum distance and average distance of each population initialization procedure on each instance. The comparative results of the population constructed with or without OBL are shown in Figure 4.5, where the X-axis shows the instances in each benchmark and Y-axis indicates the average distance and minimum distance.

From Figure 4.5, we observe that the population built by PI_{OBL} has a relatively larger average distance and minimum distance. This is particularly true for all instances of the MDG-a data set except for MDG-a_31. Also, the population produced by PI_{OBL} has a larger minimum distance than that of PI_0 for 18 out of 20 instances of the MDG-a data set. Equal or better results are found for the b2500 data set, since the population generated by PI_{OBL} dominates the population produced by PI_0 in terms of the average and minimum distances. This experiment shows that OBL helps the OBMA algorithm to start its search with a population of high diversity, which is maintained by the rank-based quality-and-distance strategy during the search.

4.6 Chapter conclusion

In this chapter, we proposed an Opposition-Based Memetic Algorithm (OBMA) which uses opposition-based learning to improve a memetic algorithm for solving Maximum Diversity Problem (MDP). The OBMA algorithm employs Opposition-Based Learning (OBL) to reinforce population diversity and im-

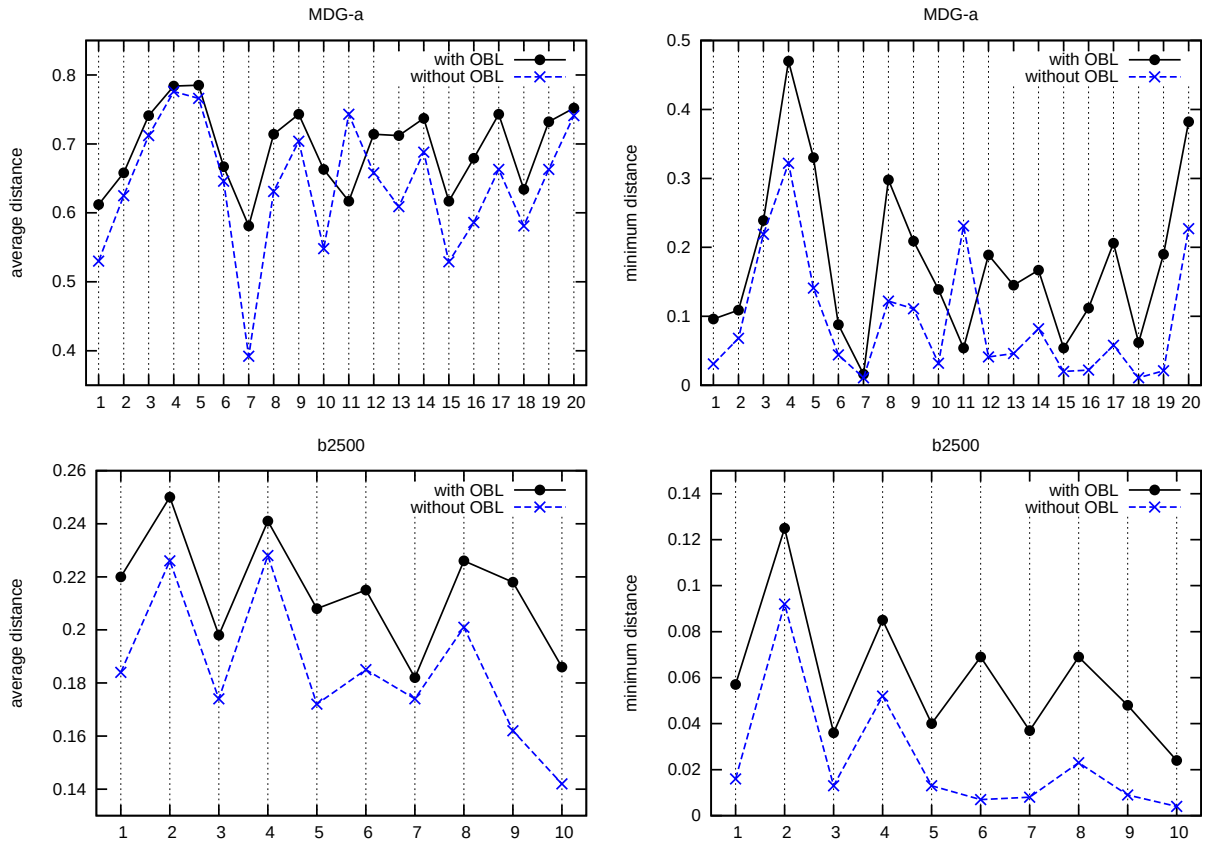

Figure 4.5: Comparative results of the populations built by population initialization with OBL (PI_{OBL}) or without OBL (PI_0).

prove evolutionary search. OBMA distinguishes itself from existing memetic algorithms by three aspects: a double trajectory search procedure which simultaneously both a candidate solution and a corresponding opposite solution, a parametric constrained neighborhood for effective local optimization, and a rank-based quality-and-distance pool updating strategy.

Extensive comparative experiments on 80 large benchmark instances (with 2000 to 5000 items) from the literature have demonstrated the competitiveness of the OBMA algorithm. OBMA matches the best-known results for most of instances and in particular finds improved best results (new lower bounds) for 22 instances which are useful for the assessment of other MDP algorithms. Our experimental analysis has also confirmed that integrating OBL into the memetic search framework does improve the search efficiency of the classical memetic search.

5

Frequent Pattern-based Search for Quadratic Assignment Problem

In this chapter, we present a hybrid approach called frequent pattern based search that combines data mining and optimization. The proposed method uses a data mining procedure to mine frequent patterns from a set of high-quality solutions collected from previous search, and the mined frequent patterns are then employed to build starting solutions that are improved by an optimization procedure. After presenting the general approach and its composing ingredients, we illustrate its application to solve the well-known and challenging quadratic assignment problem. Computational results on the 21 hardest benchmark instances show that the proposed approach competes favorably with state-of-the-art algorithms both in terms of solution quality and computing time. The context of this chapter is based on a article submitted for publication [Zhou *et al.*, 2017d].

Contents

5.1 Introduction

Much effort has been made to use machine learning techniques to enhance the performance of heuristic optimization for solving hard combinatorial optimization problems [Samorani and Laguna, 2012; Wauters *et al.*, 2013; Wauters *et al.*, 2015; Zhou *et al.*, 2016; Zhou and Hao, 2017b; Benlic *et al.*, 2017]. Heuristic search algorithms can greatly benefit from machine learning techniques. Patterns extracted by machine learning techniques can be employed to refine the operations of the heuristic algorithm and improve its performance.

In this chapter, we investigate a hybrid approach called *Frequent Pattern Based Search (FPBS)* that combines data mining techniques and optimization methods. Basically, FPBS employs a data mining procedure to mine frequent patterns that frequently occur in high-quality (or elite) solutions collected from previous search and then uses the mined frequent patterns to construct new starting solutions that are further improved by an optimization method. FPBS also integrates a procedure to manage the discovered elite solutions. The key intuition behind the proposed approach is that a good pattern extracted from high-quality solutions identifies a particularly promising region in the search space that is worthy of an intensive examination, the latter being ensured by a dedicated optimization procedure. By using multiple mined patterns combined with an effective optimization procedure, FPBS can achieve a suitable balance between search exploration and exploitation that is critical for a high performance of the whole search process.

To verify the viability of the proposed FPBS approach, we consider the well-known and highly challenging *Quadratic Assignment Problem (QAP)* as a case study. Besides its popularity as one of the most studied $\mathcal{NP}$-hard combinatorial optimization problem, the QAP is also a relevant representative of many permutation problems. To apply the general FPBS approach to solve the QAP, we specify the meanings of patterns and frequent patterns, the frequent pattern mining algorithm and the dedicated optimization procedure. We then assess the resulting algorithm on the set of the most difficult QAP benchmark instances from QAPLIB. Our experimental results show that the proposed algorithm is highly competitive compared to the state-of-the-art algorithms both in terms of solution quality and computing time.

The rest of the chapter is organized as follows. In the next section, we introduce the frequent pattern mining technique. In Section 5.3, we present the proposed frequent pattern based search approach. Section 5.4 describe the QAP and an application of FPBS to QAP. Sections 5.5 and 5.6 are dedicated to show computational results and some interesting experimental analysis, respectively. Finally, conclusions and further work are provided in Section 5.7.

5.2 Frequent pattern mining

5.2.1 Basic concept

Frequent pattern mining was originally introduced for market basket analysis in the form of association rules mining [Agrawal *et al.*, 1993] . Frequent pattern mining is used to analyze customer buying habits by identifying associations between different items that customers place in their "shopping baskets". Also, the concept of frequent itemset was first introduced for mining transaction database. Let $\mathcal{D} = \{T_1, T_2, \ldots, T_N\}$ be a transaction database defined over a set of items $I = \{x_1, x_2, x_3, \ldots, x_M\}$. A frequent itemset typically refers to a set of items that often appear together in a transactional dataset [Han *et al.*, 2011], e.g., milk and bread, which are frequently bought together in grocery stores by many customers. We call $X \subseteq I$ an itemset. An itemset with l items is called a l-itemset. X is frequent if X occurs in the transaction database $\mathcal{D}$ no lower than θ times, where θ is a user-defined minimum support threshold. X is called a frequent itemset. Let FI denote the set of all frequent itemsets. X is called a maximal frequent itemset if it has no proper superset that is frequent. The set of all maximal frequent itemsets is denoted by MFI. Apparently, all subsets of a maximal frequent itemset are also frequent itemsets. Thus we have the following relationship: $MFI \subseteq FI$.

In the original model of frequent pattern mining [Agrawal *et al.*, 1993], the problem of finding association rules has also been proposed. Association rules are closely related to frequent patterns in the sense that association rules can be considered as a "second-stage" output, which are derived from the mined frequent patterns. Given two itemsets $U \subseteq I$ and $V \subseteq I$, the rule $U \Rightarrow V$ is considered to be an association rule at minimum support θ and minimum confidence η, when the following two conditions are satisfied simultaneously: the set $U \cup V$ is a frequent pattern for the minimum support θ and the ratio of the support of $U \cup V$ over U is at least η.

5.2.2 Representation of the frequent patterns

The mined knowledge or useful information can be represented in the form of frequent patterns or association rules. There are various types of patterns, such as itemsets, subsequences, and substructures.

To apply frequent pattern mining for solving combinatorial optimization problems, one of the main challenges is to define a suitable pattern for the problem under consideration. In the following, we introduce the definitions of frequent patterns (in terms of frequent itemsets) for two categories of representative optimization problems.

— Frequent pattern for **subset selection problems**. Subset selection is basically to determine a subset of specific elements among a set of given elements while optimizing an objective function defined over the selected elements or unselected elements. Subset selection is encountered in many combinatorial optimization problems, such as knapsack problems, clique problems, diversity problems (e.g., maximum diversity problem studied in chapter 4), maximum stable problems and critical node problems. A candidate solution to a subset selection problem is usually represented by a set of selected elements. Since it is naturally to treat each element as an item, frequent pattern mining techniques can be directly applied to subset selection problems. In this setting, a transaction corresponds to a solution of the subset selection problem and a pattern can be conveniently defined as a set of elements that frequently appear in some specific (e.g., high-quality) solutions. Thus, given a database $\mathcal{D}$ (i.e., a set of visited solutions), a frequent pattern p mined from $\mathcal{D}$ with support θ corresponds to a set of elements that occur at least θ times in the database.

— Frequent pattern for **permutation problems**. Permutation problems cover another large range of important combinatorial problems. Many classic $\mathcal{NP}$-hard problems, such as the traveling salesman problem, the quadratic assignment problem, graph labeling problems and flow shop scheduling problems, are typical permutation problems. Solutions of this category of problems are usually represented as permutations, while the practical meaning of a permutation depends on the problem under consideration. To apply frequent pattern mining techniques to a permutation problem, a transformation between a permutation and an itemset is necessary. For instance, in the context of a vehicle routing problem, a transformation was proposed to map a permutation (a sequence of visited cities) into a set of pairs between any two *consecutive* elements (cities) in the permutation [Guerine *et al.*, 2016]. This transformation emphasizes the order between two consecutive elements. In Section 5.4.2, we introduce a new transformation for QAP whose main idea is to decompose a permutation into a set of element-position pairs, which focuses not only on the order between elements but also on the relation between an element and its location. In these settings, a frequent pattern can be considered as a set of pairs that frequently appear in some specific solutions.

5.2.3 Mining and heuristics

In this section, we present a brief literature review on hybridizing association rule mining (or frequent pattern mining) techniques with metaheuristics. The reviewed studies are summarized in Table 5.1.

Greedy Randomized Adaptive Search Procedure (GRASP) was the first metaheuristic to be hybridized with data mining techniques (i.e., association rule mining) [Ribeiro *et al.*, 2004; Ribeiro *et al.*, 2006].

Table 5.1: Main work on hybridizing association rules mining techniques with metaheuristics for solving combinatorial optimization problems.

Algorithm	metaheuristic*	combinatorial optimization problem	year
DM-GRASP [Ribeiro *et al.*, 2004; Ribeiro *et al.*, 2006]	GRASP	set packing problem	2004,2006
DM-GRASP [Santos *et al.*, 2005]	GRASP	maximum diversity problem	2005
DM-GRASP [Santos *et al.*, 2006b]	GRASP	server replication for reliable multicast problem	2006
GADMLS [Santos *et al.*, 2006a]	GA+LS	single-vehicle routing problem	2006
DM-GRASP [Plastino *et al.*, 2011]	GRASP	p-median problem	2011
HDMNS [Reddy *et al.*, 2012]	NS	p-median problem	2012
DM-GRASP-PR [Barbalho *et al.*, 2013]	GRASP+PR	2-path network design problem	2013
MDM-GRASP [Plastino *et al.*, 2014]	GRASP	server replication for reliable multicast problem	2014
DM-HH [Martins *et al.*, 2014]	PR+LS	p-median problem	2014
VALS [Umetani, 2015]	LS	set partitioning problem	2015
GAAR [Raschip *et al.*, 2015b]	GA	constrain satisfaction problem	2015
GAAR [Raschip *et al.*, 2015a]	GA	weighted constrain satisfaction problem	2015
MDM-GRASP/VND [Guerine *et al.*, 2016]	GRASP+VND	one-commodity pickup-and-delivery traveling salesman problem	2016
DM-ILS [Martins *et al.*, 2016]	ILS	set covering with pairs problem	2016

* Greedy Randomized Adaptive Search Procedure (GRASP), Genetic Algorithm (GA), Local Search (LS), Neighborhood Search (NS), Path-Relinking (PR), Variable Neighborhood Descent (VND), and Iterated Local Search (ILS).

Denoted as DM-GRASP, this approach was originally designed to solve the set packing problem. DM-GRASP organizes its search process into two sequential phases, and incorporates an association rule mining procedure at the second phase. Specifically, high-quality solutions found in the first phase (GRASP) were stored in an elite set, and then a data mining procedure is invoked to extract some patterns from the elite set. At the second phase, a new solution is constructed based on a mined pattern instead of using the greedy randomized construction procedure of GRASP. This approach has been applied to solve several problems including the maximum diversity problem [Santos *et al.*, 2005], the server replication for reliable multicast problem [Santos *et al.*, 2006b], and the p-median problem [Plastino *et al.*, 2011]. A survey on some significant applications of DM-GRASP can be found in [Santos *et al.*, 2008]. An interesting extension of DM-GRASP is to execute the data mining procedure multiple times instead of only once [Plastino *et al.*, 2014; Guerine *et al.*, 2016; Martins *et al.*, 2016]. Compared to DM-GRASP where the data mining call occurs once at the midway of the whole search process, the multi-mining version performs the mining task when the elite set stagnates. The same idea has been explored recently by hybridizing data mining and GRASP enhanced with path-relinking [Barbalho *et al.*, 2013] or variable neighborhood descent [Guerine *et al.*, 2016].

In addition to GRASP, data mining has also been hybridized with other metaheuristics like evolutionary algorithms. To improve the performance of an evolutionary algorithm applied to an oil collecting vehicle routing problem, a hybrid algorithm (GADMLS) combining genetic algorithm, local search and data mining was proposed in [Santos *et al.*, 2006a]. Another hybrid approach (GAAR) that uses a data mining module to guide an evolutionary algorithm was presented in [Raschip *et al.*, 2015b; Raschip *et al.*, 2015a] to solve the constraint satisfaction problem. Besides the standard components of a genetic algorithm, a data mining module is added to find association rules (between variables and values) from an archive of best individuals found in the previous generations.

Apart from GRASP and evolutionary algorithms, it has been shown that other heuristics can also benefit from the incorporation of a data mining procedure. For example, a data mining approach was applied to extract variable associations from previously solved instances for identifying promising pairs of flipping variables in the large neighborhood search method, thus reducing the search space of local search algorithms for the set partitioning problem [Umetani, 2015]. Another example is the hybridization of neighborhood search with data mining techniques for solving the p-median problem [Reddy *et al.*, 2012]. Also for the p-median problem, a data mining procedure was integrated into a multistart hybrid heuristic [Martins *et al.*, 2014], which combines elements of different traditional metaheuristics (e.g., local search) and uses path-relinking as a memory-based intensification mechanism. Finally, the widely-used iterated local search method was also hybridized with data mining for solving the set covering with pairs problem [Martins *et al.*, 2016].

5.3 Frequent pattern-based search

In this section, we present *Frequent Pattern Based Search (FPBS)*, a general-purpose search approach that integrates frequent pattern mining into a metaheuristic. Below, we first show the general scheme of the proposed FPBS approach, and then present its composing ingredients.

5.3.1 General scheme

The FPBS approach uses relevant frequent patterns extracted from high-quality solutions to build promising starting solutions which are further improved by a metaheuristic optimization procedure. The improved solutions are in turn used to help mine additional patterns. By iterating the mining procedure and the optimization procedure, FPBS is expected to examine the search space effectively and efficiently. As a case study, we illustrate in Section 5.4 its application to solve the well-known QAP.

From a perspective of system architecture, apart from an archive of high-quality solutions (called elite set) that is maintained for the purpose of pattern mining, FPBS is composed of five critical components: an initialization procedure (Section 5.3.2), a local optimization search procedure (Section 5.3.4), a data mining procedure (Section 5.3.3), a frequent pattern based solution construct procedure (Section 5.3.5) and an elite solution management procedure (Section 5.3.6).

Algorithm 9: A general framework of the proposed FPBS approach

Input: Problem instance I, elite set size k and number of patterns to be mined m
Output: The best solution S^*

```
1  begin
2      // I is supposed to be a minimization problem
3      ES ← EliteSetInitialize();
4      // record the best solution found so far
5      S* ← arg min{f(S_i) : i = 1, 2, ..., k};
6      P ← FrequentPatternMine(ES,m);
7      while a stopping condition is not reached do
8          p ← PatternSelect(P);
9          // construct a new solution based mined frequent pattern
10         S ← PatternBasedConstruct(p);
11         // improve the constructed solution
12         S' ← Optimize(S);
13         // update the best solution found so far
14         if f(S') < f(S*) then
15             S* ← S'
16         // update the elite set
17         ES ← EliteSetUpdate(ES, S');
18         // restart the mining procedure when the elite set stagnates
19         if elite set stagnates then
20             P ← FrequentPatternMine(ES, m);
```

The general framework of the proposed FPBS approach is presented in Algorithm 9. FPBS starts from a set of high-quality solutions that are obtained by the initialization procedure. From these high-quality solutions, a data mining procedure is invoked to mine a number of frequent patterns. A new solution is then constructed based on a mined pattern and further improved by the optimization procedure. The improved solution is finally inserted to the elite set according to the elite solution management policy. The process is repeated until a stopping condition (e.g., a time limit) is satisfied. In addition to its invocation just after

the initialization procedure, the data mining procedure is also called each time the elite set is judged to be stagnating, i.e., it has not been updated during a predefined number of iterations.

5.3.2 Elite set initialization

FPBS starts its search with an *Elite Set (ES)* composed of k distinct high-quality solutions. To build such an elite set, we first generate, by any means (e.g., with a random or greedy construction method), an initial solution that is improved by an optimization procedure (see Section 5.3.4). The improved solution is then inserted into the elite set according to the elite set management strategy (see Section 5.3.6). We repeat this process until an elite set of k different solutions is built. Note that similar ideas have been successfully applied to build a high-quality initial population for memetic algorithms [Lü and Hao, 2010; Benlic and Hao, 2015; Zhou *et al.*, 2017c; Zhou *et al.*, 2017e].

5.3.3 Frequent pattern mining procedure

In our approach, the frequent pattern mining procedure is used to discover some patterns that frequently occurs in high-quality solutions stored in elite set. Besides the itemsets (see Section 5.2.2), mined patterns can also be represented as subsequences or substructures [Aggarwal *et al.*, 2014]. A subsequence is an order of items, such as buying a mobile phone first, then a power bank, and finally a memory card. If a subsequence occurs frequently in a transaction database, then it is a frequent sequential pattern. A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemset or subsequences. If a substructure occurs frequently in a graph database, it is called a frequent structural pattern. A schematic diagram of the frequent pattern mining is given in Figure 5.1.

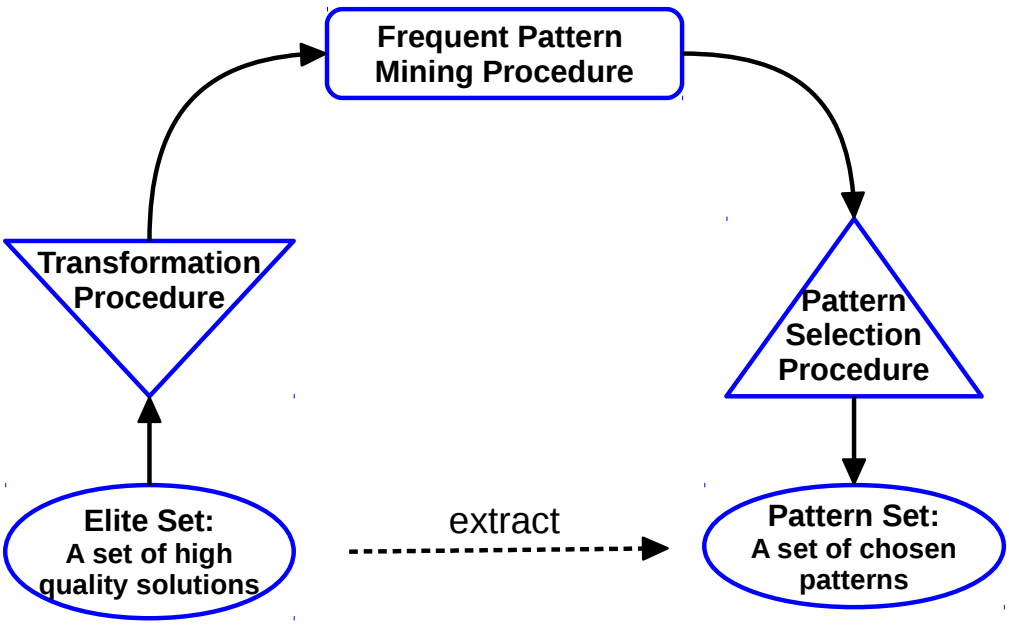

Figure 5.1: A schematic diagram of the frequent pattern mining.

Table 5.2: A simple summary of frequent pattern mining algorithms [Aggarwal *et al.*, 2014].

		algorithm	
Task	pattern	apriori-based	pattern-growth
frequent itemset mining	itemsets	e.g., Apriori	e.g., FP-growth, FPmax
sequential pattern mining	subsequences	e.g., GSP, SPADE	e.g., PrefixSpan
structural pattern mining	substructures	e.g., AGM, FSG	e.g., gSpan, FFSM

To handle a wide diversity of data types, numerous mining tasks and algorithms have been proposed in the literature [Han *et al.*, 2007; Aggarwal *et al.*, 2014]. A simple summary of frequent pattern mining algorithms is provided in Table 5.2. For a specific application, the patterns of high-quality solutions collected in the elite set can be expressed as itemsets, subsequences, or substructures. Once the form of patterns is determined, a suitable mining algorithm can be selected accordingly. Consequently, we adopt the FPmax* algorithm (see Section 5.4.2 for more details) to mine only the maximal frequent itemsets. Detailed reviews of frequent pattern mining algorithms can be found in [Han *et al.*, 2007; Aggarwal *et al.*, 2014].

5.3.4 Optimization procedure

For the purpose of solution improvement (to built the initial elite set and to improve each new solution built from a mine pattern), any optimization procedure dedicated to the given problem can be applied in principle. On the other hand, since the optimization component ensures the key role of search intensification, it is desirable to call for a powerful search algorithm. Basically, the optimization procedure can be considered to be a black box optimizer that is called to improve the input solution. In practice, the optimization procedure can be based on local search, population-based search or even hybrid memetic search. In any case, the search procedure must be carefully designed with respect to the problem under consideration and should ideally be effective both in terms of search capacity and time efficiency. As we show in Section 5.4, for the QAP considered in this work, we will adopt the powerful breakout local search procedure [Benlic and Hao, 2013] as our optimization procedure.

5.3.5 Construction based on mined pattern

Once a set of frequent patterns $\mathcal{P}$ is extracted from the elite set, new solutions are constructed based on these mined patterns. For this purpose, we first select a mined frequent pattern by the tournament selection strategy as follows. Let λ be the size of the tournament pool. We randomly choose λ ($1 \leqslant \lambda \leqslant |\mathcal{P}|$) individuals with replacement from the mined pattern set $\mathcal{P}$, and then pick the best one (i.e., with the largest size), where λ is a parameter. The computational complexity of this selection strategy is $O(|\mathcal{P}|)$. The advantage of the tournament selection strategy is that the selection pressure can be easily adjusted by changing the size of the tournament pool λ. The larger the tournament pool is, the smaller the chance for shorter patterns to be selected.

Since frequent patterns usually corresponds to a set of common elements shared by the high-quality solutions examined by the mining procedure, each mined pattern directly defines a partial solution. To obtain a complete solution, we can apply a greedy or random procedure to the partial solution. The way to build such a solution shares similarity with the general backbone-based crossover procedure [Benlic and Hao, 2011; Zhou *et al.*, 2017c; Zhou *et al.*, 2017e]. Compared to the notion of backbones that are typically shared by two parent solutions, our frequent patterns are naturally shared by two or more high-quality solutions. In this sense, backbones can be considered as a special case of more general frequent patterns.

Finally, it is possible to construct a new solution for each mined pattern instead of using a long pattern selected by the tournament selection strategy, as explained above. Also, an elite solution can be selected to guide the construction of a new solution. In particular, the way to use frequent patterns should be determined according to the studied problem.

5.3.6 Elite set management

As explained above, each new solution constructed using a mined frequent pattern is improved by the optimization procedure. Then, we decide whether the improved solution should be inserted into the *Elite Set* (ES).

There are a number of updating strategies in the literature [Sörensen and Sevaux, 2006]. For example, the classic quality-based replacement strategy simply inserts the solution into ES to replace the worst solution if it is better than the worst solution in ES. In addition, more elaborated updating strategies consider other criteria than the quality of solutions. For example, the quality-and-distance updating strategy not only considers the quality of the solution, but also its distance to other solutions in the population [Lü and Hao, 2010]. An improved quality-and-distance updating strategy, called the rank-based updating strategy, was recently proposed in [Zhou and Hao, 2017b]. A suitable elite set management strategy can be determined according to the practical problem.

5.4 FPBS applied to the quadratic assignment problem

In this section, we present a case study of applying the general FPBS approach to the well-known *Quadratic Assignment Problem (QAP)* and show its competitiveness compared to state-of-the-art QAP algorithms.

5.4.1 Related work

In this section, we provide a brief literature review of the most popular heuristic algorithms for QAP in the literature.

Due to its practical and theoretical significance, QAP has attracted much research effort since its first formulation [James *et al.*, 2009; Benlic and Hao, 2013; Benlic and Hao, 2015; Tosun, 2015; Acan and Ünveren, 2015]. In fact, QAP is among the most studied and the most competitive combinatorial optimization problems. Since exact algorithms are unpractical for instances of size larger than 36 [Anstreicher *et al.*, 2002], a large number of heuristic methods have been proposed for QAP, which could provide a near-optimal solution in reasonable computation times. Roughly, these heuristic algorithms can be divided into two categories:

— **Local search** metaheuristics, such as simulated annealing [Wilhelm and Ward, 1987; Connolly, 1990; Peng *et al.*, 1996; Misevičius, 2003], tabu search [Skorin-Kapov, 1990; Taillard, 1991; Skorin-Kapov, 1994; Misevicius, 2005; James *et al.*, 2009; Acan and Ünveren, 2015], iterated local search [Stützle, 2006; Benlic and Hao, 2013; Aksan *et al.*, 2017].

— **Population-based** metaheuristics, such as ant colony optimization [Stützle and Dorigo, 1999; Gambardella *et al.*, 1999; Demirel and Toksarı, 2006], genetic algorithm [Ahmed, 2015; Tosun, 2015; Benlic and Hao, 2015].

Besides the translational heuristic algorithms, many high performance computation techniques are introduced to accelerate the exiting heuristic algorithms in recent years.

— CPU computation with Compute Unified Device Architecture (CUDA) combined with tabu search and ant colony optimization is proposed for the QAP in [Tsutsui and Fujimoto, 2009].

— A high-performance multistart hyper-heuristic algorithm on the grid (using parallel processing) for the solution of the QAP is proposed in [Dokeroglu and Cosar, 2016].

— A parallel multistart tabu search is proposed in [Czapiński, 2013], and it is implemented on a highly powerful GPU hardware intended for high-performance computing with the CUDA platform.

— A parallel hybrid algorithm with three phases was proposed in [Tosun, 2015].

— An enhanced Breakout local search algorithm incorporating the multi-threaded computation using OpenMP is proposed in [Aksan *et al.*, 2017].

Detailed reviews of heuristic and metaheuristic algorithms developed till 2007 for QAP are available in [Drezner *et al.*, 2005; Loiola *et al.*, 2007].

5.4.2 FPBS for QAP

Algorithm 10 shows the FPBS algorithm for QAP (denoted as FPBS-QAP), which is an instantiation of the general scheme of Algorithm 9. Since FPBS-QAP inherits the main components of FPBS, hereafter we only present the specific features related to QAP: solution representation and evaluation, optimization procedure, frequent pattern mining for QAP, solution construction using QAP patterns, and elite set update strategy.

Algorithm 10: The FPBS algorithm for QAP

Input: Instance G, elite set size k, the number of patterns m, time limit t_{max} and the maximum number of iterations without updating max_no_update

Output: The best solution π^* found so far

```
1  begin
2      ES ← EliteSetInitialize();
3      π* ← arg min{f(π_i) : i = 1, 2, ..., k};
4      P ← FrequentPatternMine(ES, m);
5      no_update ← 0;
6      t ← 0;
7      while t < t_max do
8          p_i ← PatternSelection(P);
9          // construct a new solution based selected pattern
10         π ← PatternBasedConstruct(p_i);
11         // improve the constructed solution
12         π' ← BreakoutLocalSearch(π);
13         // update the best solution found so far
14         if f(π') < f(π*) then
15             π* ← π';
16         // update the elite set
17         if EliteSetUpdate(ES, π') = True then
18             no_update ← 0;
19         else
20             no_update ← no_update + 1;
21         // restart the mining procedure when the elite set is steady
22         if no_update > max_no_update then
23             P ← FrequentPatternMine(ES, m);
24             no_update ← 0;
```

Solution representation, neighborhood and evaluation

Given a QAP instance with n facilities (or locations), a candidate solution is naturally represented by a permutation π of $\{1, 2, \ldots, n\}$, such that $\pi(i)$ is the location assigned to facility i. The search space Ω is thus composed of all possible $n!$ permutations.

To examine the search space, we adopt an iterated local search algorithm called BLS (see Section 5.4.2). For this purpose, we introduce the neighborhood used by BLS. Give a solution, i.e., a permutation π, its neighborhood $N(\pi)$ is defined as the set of all possible permutations that can be obtained by exchanging the values of any two different positions $\pi(u)$ and $\pi(v)$ in π, i.e., $N(\pi) = \{\pi' \mid \pi'(u) = \pi(v), \pi'(v) = \pi(u), u \neq v \; and \; \pi'(i) = \pi(i), \forall i \neq u, v\}$, which has a size of $n(n-1)/2$.

As indicated in [Taillard, 1991], given a permutation π and its objective value $f(\pi)$, the objective value of any neighnoring permutation π' can be effectively calculated according to an incremental evaluation technique.

Breakout local search

To ensure an effective optimization of a given solution, we adopt the *Breakout Local Search (BLS)* algorithm [Benlic and Hao, 2013], which is one of the most powerful algorithms for QAP currently available in the literature.

BLS is a variant of *Iterated Local Search (ILS)* [Lourenço *et al.*, 2003], which alternates iteratively between a descent search phase (to find local optima) and a dedicated perturbation phase (to discover new promising regions). BLS starts from an initial random permutation, and then improves the initial solution to a local optimum by a best improvement descent search with the above exchange-based neighborhood. Afterwards, BLS triggers a perturbation based diversification mechanism. The perturbation mechanism adaptively selects a tabu-based perturbation (called directed perturbation) or a random perturbation (called undirected perturbation). BLS also determines the number of perturbation steps (called perturbation strength) in an adaptive way [Benlic and Hao, 2013].

The tabu-based perturbation and random perturbation provide two complementary means for search diversification. The former applies a selection rule that favors neighboring solutions that minimize the objective degradation, under the constraint that the neighboring solutions have not been visited during the last γ iterations (where γ is the pre-defined tabu tenure), while the latter performs moves selected uniformly at random. To keep a good balance between an intensified and a diversified search, BLS alternates probabilistically between these two perturbations. The probability to select a particular perturbation is determined dynamically according to the current number of times to visit local optima without any improvement on the best solution found. The probability of applying the tabu-based perturbation over the random perturbation is empirically limited to be at least as Q. The perturbation strength L is determined based on a simple reactive strategy. One increases L by one if the local search procedure returned to the immediate previous local optimum, and otherwise resets L to a given initial value L_0. Once the type and the strength L of the perturbation are determined, the selected perturbation with strength L is applied to the current solution. The resulting solution is then used as the starting solution of the next round of the descent search procedure (see [Benlic and Hao, 2013] for more details).

FPmax* for QAP

The quadratic assignment problem is a typical permutation problem whose solutions are naturally represented by permutations. For QAP, we define a frequent pattern to be a set of identical location-facility assignments shared by high-quality solutions, and represent a frequent pattern by an itemset. To apply a frequent itemset mining algorithm, we need to transform a permutation into a set of items. A transformation is recently proposed in [Guerine *et al.*, 2016]. The transformation works as follows. For each pair of elements ($\pi(i)$ and $\pi(j)$) of a given permutation π, an arc $(\pi(i), \pi(j))$ is generated, which maps a permutation π to a set S' of $|\pi| - 1$ arcs. For example, considering a permutation $\pi = (5, 4, 7, 2, 1, 6, 3)$ represented by a sequence of these elements, π can be mapped as $S' = \{(5, 4), (4, 7), (7, 2), (2, 1), (1, 6), (6, 3)\}$. By this transformation, a permutation is divided into a set of arcs, which conserves the order of elements. However, this transformation loses the information between the elements and their locations. In practice, we can not identify the true location of an element when only a part of pairs are available.

To overcome this difficulty, we propose a new transformation. Our transformation decompose a permutation into a set of ordered pairs, each pair being formed by an element (facility) i and its position (location) $\pi(i)$. Specifically, let π be a candidate solution of QAP, for each element (facility) i, a corresponding element-position pair $(i, \pi(i))$ is generated, which transforms permutation π into a set of n element-position pairs. For example, given $\pi = (5, 4, 7, 2, 1, 6, 3)$, $\{(1, 5), (2, 4), (3, 7), (4, 2), (5, 1), (6, 6), (7, 3)\}$ is the corresponding set of element-position pairs.

Once a permutation is transformed into a set of pairs, we will treat each pair $(i, \pi(i))$ as an item $(i - 1) * |\pi| + \pi(i)$, where $|\pi|$ is the length of the permutation. An illustrative example is given in Figure 5.2. Now, the task of mining frequent patterns from multiple permutations can be conveniently transformed into the task of mining frequent itemsets. The main drawback of mining all frequent itemsets is that if there is

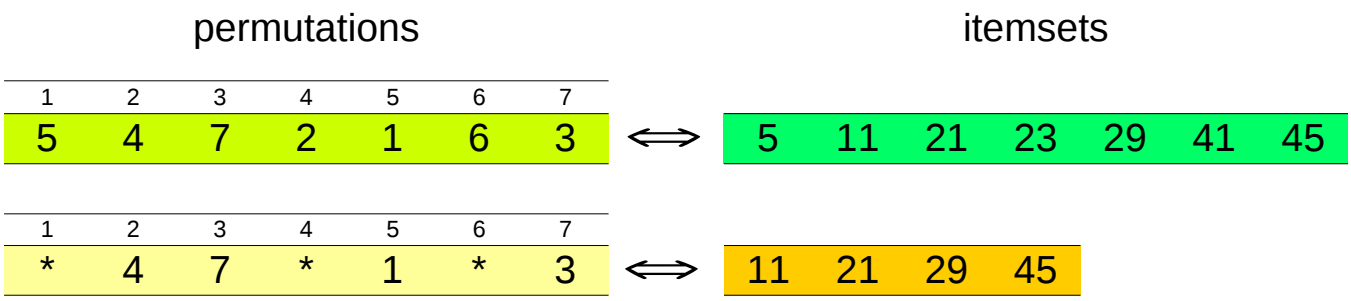

Figure 5.2: An illustrative example of transformation procedure.

a large frequent itemset, then almost all subsets of the itemset might be examined. However, it is usually sufficient to find only the maximal frequent itemsets (a maximal frequent itemset is such that if it has no superset that is frequent). Thus mining frequent itemsets can be reduced to mine only the maximal frequent itemsets. For this purpose, we adopt the popular **FPmax*** [1] algorithm [Grahne and Zhu, 2003a].

FPmax is an algorithm to mine maximal frequent itemsets using the FP-tree (frequent pattern tree) structure, while the FPmax* is an improved version of FPmax. Compared to FPmax, in addition to the FP-array technique, the improved method FPmax* also has a more efficient maximality checking approach, as well as several other optimizations. Experimental results show that FPmax* outperforms FPmax [Grahne and Zhu, 2003b].

FP-tree and **FP-growth method**: FP-tree is a compact data structure used by FP-growth method to store the information about frequent itemsets in a database. The frequent items of each transactions are inserted into the tree in their frequency descending order. Compression is achieved by building the tree in such a way that overlapping itemsets share prefixes of the corresponding branches. An FP-tree T has a header table, $T.header$, associated with it. Single items and their counts are stored in the header table in decreasing order of their frequency. The entry for an item also contains the head of a list that links all the corresponding nodes in the tree.

Compared with Apriori-like algorithms which may need as many database scans as the length of the longest pattern, the FP-growth method needs only two database scans. The first scan collects all frequent itemsets, while the second scan constructs the initial FP-tree. Figure 5.3 (a) shows an example of a database and Figure 5.3 (b) is the FP-tree for this database. The constructed FP-tree records all frequency information of the database.

In FPmax, a global data structure, the maximal frequent itemsets tree (**MFI-tree**) is introduced to keep the track of MFIs. Each MFI-tree is associated with a particular FP-tree. An MFI-tree resembles an FP-tree. There are two main differences between MFI-trees and FP-trees. In an FP-tree, each node in the subtree has three fields: item-name, count, and node-link. In an MFI-tree, the count is replaced by the level of the node. Another difference is that the header table in an FP-tree is constructed from traversing the previous FP-tree or using the associated FP-array, while the header table of an MFI-tree is constructed based on the item order in the table of the FP-tree it is associated with. For more information about the FP-tree, FP-growth algorithm, FP-array and MFI-tree, readers are referred to [Grahne and Zhu, 2003a; Grahne and Zhu, 2005].

Algorithm 11 shows the pseudo code of the **FPmax*** algorithm. Here we only provide a brief presentation of its general procedure. In the initial call, a FP-tree is constructed from the first scan of the database, together with with an initial empty MFI-tree. During the recursion, if there is only one single path in FP-tree T, this single path together with $T.base$ is a MFI of the dataset. The MFI is then inserted into M (line 3). Otherwise, for each item i in the header table, $Y = T.base \cup \{i\}$, and we start preparing for the recursive call FPmax*(T_Y, M_Y). The items in the header table are processed in increasing order of frequency, so that maximal frequent itemsets will be found before any of their frequent subsets (line 11). Lines 7-10 use the array technique, and line 12 invokes the *subset_checking*() function to check if Y together with all fre-

1. The source code of the FPmax* algorithm is publicly available at http://fimi.ua.ac.be/src/

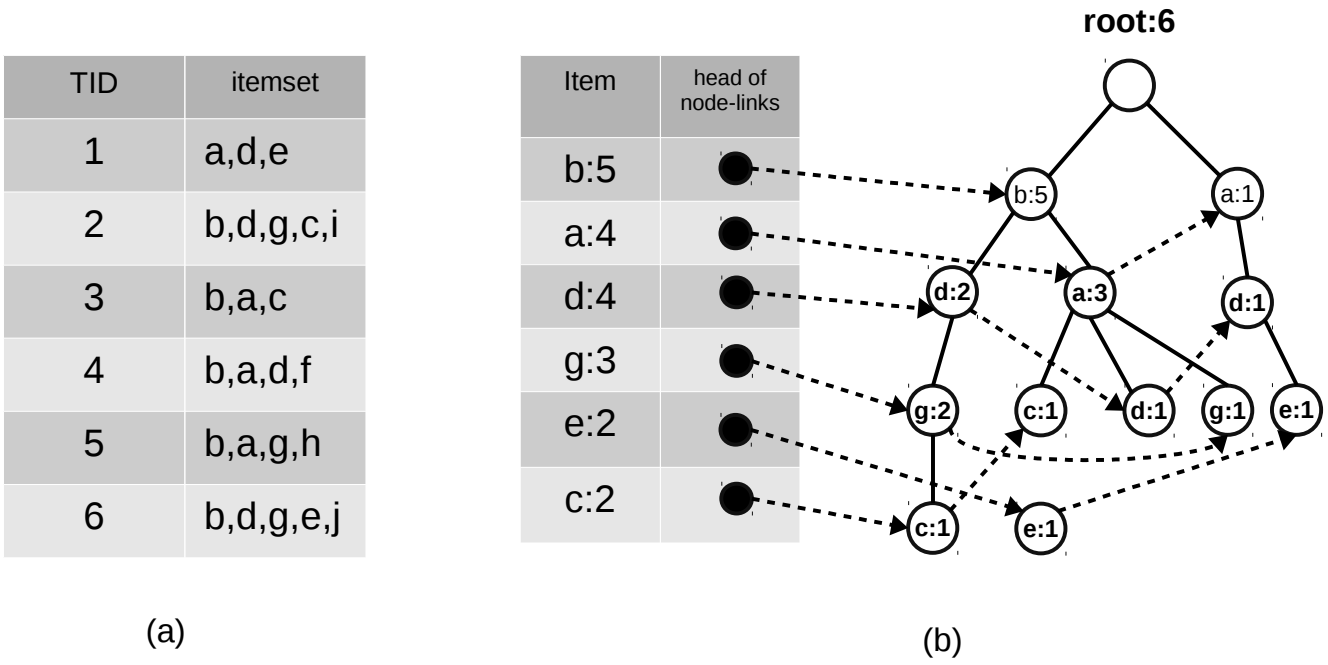

Figure 5.3: An FP-tree example. (a) A database. (b) The FP-tree for the database (minimum support = 2).

Algorithm 11: Pseudo code of the **FPmax*** Algorithm [Grahne and Zhu, 2003]

Input: T: an FP-tree
M: the MFI-tree for $T.base$.
Output: Updated M

```
1  begin
2      if T only contains a single path P then
3          insert P into M
4      else
5          for each i in T.header do
6              set Y = T.base ∪ {i};
7              if T.array is not Null then
8                  Tail={frequent items for i in T.array}
9              else
10                 Tail={frequent items in i's conditional pattern base}
11             sort Tail in decreasing order of the items' counts;
12             if subset_checking(Y ∪ Tail, M) = False then
13                 construct Y's conditional FP-tree T_Y and its array A_Y;
14                 initialize Y's conditional MFI-tree M_Y;
15                 call FPmax*(T_Y, M_Y);
16                 merge M_Y with M;
```

quent items in Y's conditional pattern base is a subset of any existing MFI in M, thus we perform superset pruning. If function *subset_checking*() returns $False$, FPmax* will be called recursively, with (T_Y, M_Y) (lines 13-16). For a detailed description of the FPmax* algorithm, please refer to [Grahne and Zhu, 2003a; Grahne and Zhu, 2005]. An example to illustrate the whole process of the frequent pattern mining is provided in Figure 5.4.

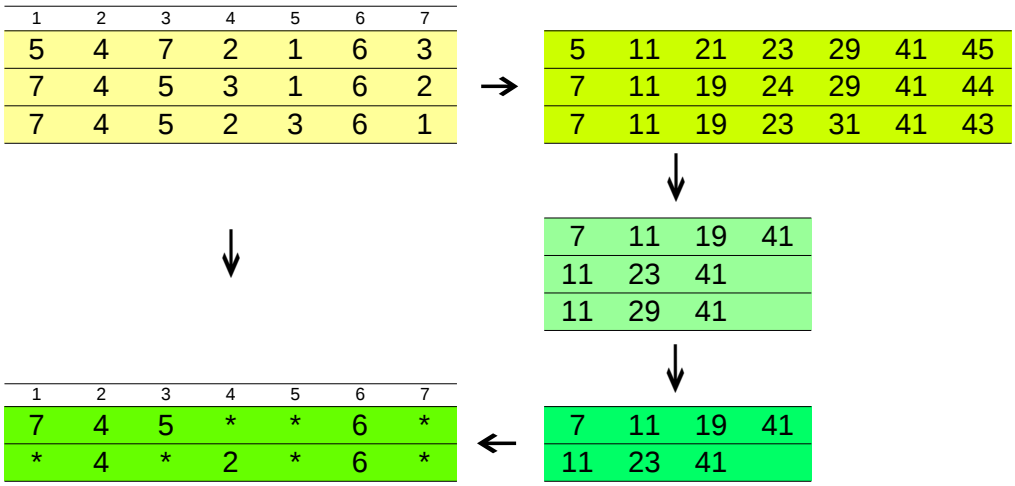

Figure 5.4: An FP-tree example.

Solution construction based on mined pattern

Algorithm 12 describes the main steps to construct a new solution based on a mined frequent pattern. Initially, a pattern is first selected from the set of mined frequent patterns $\mathcal{P}$ according to the tournament selection strategy (line 2). Then, we re-map this chosen pattern into a partial solution π (line 3). If the length of the partial solution $|\pi|$ is less than a given threshold (i.e., $\beta * n$, β is a length threshold), we introduce an elite solution to guide the construction (lines 4-6). Specifically, we first select a good elite solution π^0 (called guiding solution) from the elite set (line 5), and then we continue to complete π based on the guiding solution π^0 (i.e., directly copies the elements of all unassigned positions of π^0 to π if the elements have not been assigned in π). Finally, if π is still an illegal solution, we randomly assign the remaining elements to the unassigned positions until a legal solution is obtained (line 7).

Algorithm 12: Solution construction based on mined frequent pattern

Input: Instance G, a set of mined patterns $\mathcal{P}$ and an elite set ES of size k and length threshold β
Output: A new solution π

1 **begin**
2 $p \leftarrow$ *PatternSelection*$(\mathcal{P})$; /* select a mined pattern */
3 $\pi \leftarrow$ *re-map*(p); /* generate a partial solution based on selected pattern */
4 **if** $|\pi| < \beta * n$ **then**
5 $\pi^0 \leftarrow$ *GuidedSolutionSelection*(ES); /* select a guided solution */
6 $\pi \leftarrow$ *GuidedComplete*(π, π^0); /* complete based on guided solution */
7 $\pi \leftarrow$ *RandomComplete*(π); /* complete at random */

Our solution construction method shares similar idea with the backbone-based crossovers [Benlic and Hao, 2011; Zhou and Hao, 2017b; Zhou *et al.*, 2017c], i.e., directly inheriting some common elements from the high-quality solutions. However, our solution construction method distinguishes itself from these backbone-based crossovers by adopting the concept of frequent patterns, which are the common parts shared by two or more high-quality solutions instead of only two parent solutions.

Elite set update strategy

Once a new improved solution π' is obtained by the breakout local search, then we need to decide whether π' should be inserted into the elite set ES. In our case, we adopt the classic quality-based update strategy. Specifically, we insert π' into the elite set ES when the following two conditions are satisfied simultaneously. That is, (i) π' is different from any individual in the elite set and (ii) the quality of π' is not

worse than the worst individual in the elite set, i.e., $f(\pi') \leqslant f(\pi^w)$, where $\pi^w \leftarrow \arg\max_{\pi \in ES} f(\pi)$ is the worst individual in the ES. The solution of QAP is usually represented as a permutation. Therefore, two solutions π^u and π^v of QAP are the same if and only if their assignments at all positions are the same. That is, the hamming distance between π^u and π^v is zero.

5.5 Computational results

Our computational studies aim to evaluate the efficiency of the FPBS-QAP algorithm. For this purpose, we first perform a detailed performance comparisons between FPBS-QAP and two state-of-the-art algorithms, i.e., BLS [Benlic and Hao, 2013] and BMA [Benlic and Hao, 2015], whose source codes are available to us. Furthermore, we compare FPBS-QAP with four additional state-of-the-art algorithms in the literature.

5.5.1 Benchmark instances

The experimental evaluations of QAP algorithms are usually performed on 135 popular benchmark instances from QAPLIB [2], with n ranging from 12 to 150. These instances can be classified into four categories:

— Type I. 114 **real-life instances** are obtained from practical QAP applications;

— Type II. 5 **unstructured, randomly generated instances** whose distance and flow matrices are randomly generated based on a uniform distribution;

— Type III. 5 **real-like-life instances** are generated instances that are similar to the real-life QAP instances;

— Type IV. 11 **instances with grid-based distances** in which the distances are the Manhattan distance between points on a grid.

Like [Benlic and Hao, 2013; Benlic and Hao, 2015], we do not consider the 114 real-life instances from Type I because they can be solved optimally with a high success rate and a short computation time (often less than one second). Our experiments will focus on the remaining 21 hard instances from Type II, Type III and Type IV. Notice that for these 21 challenging instances, no single algorithm can attain the best-known results for all the 21 instances. Indeed, even the best performing algorithm misses at least two best-known results.

5.5.2 Experimental settings

The proposed FPBS-QAP algorithm [3] is implemented in the C++ programming language and complied with gcc 4.1.2 and flag '-O3'. All the experiments were carried out on a computer equipped with an Intel E5-2670 processor with 2.5 GHz and 2 GB RAM operating under the Linux system. Without using any compiler flag, running the well-known DIMACS machine benchmark program dfmax [4] on our machine requires 0.19, 1.17 and 4.54 seconds to solve graphs r300.5, r400.5 and r500.5 respectively. Our computational results were obtained by running the FPBS-QAP algorithm with the parameter settings provided in Table 5.3. To identify an appropriate value for a given parameter, we compare the performance of the algorithm with different parameter values, while fixing other parameter values. An example to select an appropriate m value is provided in Section 5.6.3.

2. https://www.opt.math.tugraz.at/qaplib/
3. The source code of our FPBS-QAP algorithm will be made available at http://www.info.univ-angers.fr/~hao/fpbs.html
4. dfmax: ftp://dimacs.rutgers.edu/pub/dsj/clique

Table 5.3: Parameter settings of FPBS-QAP algorithm.

Parameter	description	value
t_{max}	time limit (hours)	0.5 or 2.0
k	elite set size	15
max_no_update	number of times without updating	15
θ	minimum support	2
m	frequent pattern set size	11
λ	tournament pool size	3
β	length threshold	0.75
max_iter	number of iterations for BLS*	10000

* Other parameters of BLS adopt the default values provided in [Benlic and Hao, 2013].

Given its stochastic nature, the proposed FPBS-QAP algorithm was independently ran 10 times on each test instance, which is a standard practice for solving QAP [Benlic and Hao, 2013; Benlic and Hao, 2015; Tosun, 2015; Acan and Ünveren, 2015]. Performance assessment is based on the Percentage Deviation (PD) metrics that are widely used in the literature. The PD metrics measure the percentage deviation from the Best-Known Value (BKV). The PD metrics, such as the Best Percentage Deviation (BAD), Average Percentage Deviation (APD) and the Worst Percentage Deviation (WPD), which are respectively calculated according to:

$$XPD = 100 * \frac{X - BKV}{BKV}[\%] \tag{5.1}$$

where X is the best objective value, average objective value and worst objective value. The smaller the XPD value, the better the evaluated algorithm.

5.5.3 Comparison of FPBS-QAP with BLS and BMA

To demonstrate the effectiveness of the FPBS-QAP algorithm, we first show a detailed comparison between FPBS-QAP and two reference algorithms BLS [Benlic and Hao, 2013] and BMA [Benlic and Hao, 2015]. This experiment was based on two motivations. First, BLS and BMA are among the best performing QAP algorithms in the literature. Second, the source codes of BLS and BMA are available to us, making it possible to make a fair comparision (using the same computing platform and stopping conditions). Third, both FPBS-QAP and BMA use BLS as their underlying optimization procedure, this comparison allows us to assess the added value of the data mining component of FPBS-QAP. For this experiment, we run FPBS-QAP and the two reference algorithms under two stopping conditions, i.e., a limit of $t_{max} = 30$ minutes (0.5 hour) and a limit of $t_{max} = 120$ minutes (2 hours). This allows us to study the behavior of the compared algorithms under short and long conditions.

The comparative performances of FPBS-QAP with BLS and BMA under $t_{max} = 30$ minutes and $t_{max} = 120$ minutes are presented in Table 5.4 and Table 5.5, respectively. In these two tables, we report the BPD, APD, WPD values of each algorithm, and the average time (T) to achieve the results. At the end of each table, we also give the average value of each indicator. The smaller the value, the better the performance of an algorithm.

From Table 5.4, we observe that FPBS-QAP achieves the best performance compared to the algorithms BLS and BMA under $t_{max} = 30$ minutes. Specifically, FPBS-QAP is able to reach the best-known values of two largest instances, i.e., tai150b and tho150 within the given computing time. To the best of our knowledge, it is the first heuristic algorithm that achieves these two best-known values within only 30 minutes. For the BLS and BMA algorithms, they achieve the best-known values of tai150b and tho150 with a low success rate only under a very long time limit of $t_{max} = 10$ hours. The BPD value of FPBS-QAP for the 21 benchmark instances is only 0.043%, which is smaller than 0.059% of BLS, and 0.051% of BMA

Table 5.4: Comparative performance of the proposed FPBS-QAP algorithm with BLS and BMA on 21 hard instances under $t_{max} = 30$ minutes. The success rate of reaching the best-known value over 10 runs is indicated in parentheses.

		BLS				BMA				FPBS-QAP			
Instance	BKV	BPD	APD	WPD	$T(m)$	BPD	APD	WPD	$T(m)$	BPD	APD	WPD	$T(m)$
tai40a	3139370	0.000(1)	0.067	0.074	5.5	0.000(1)	0.067	0.074	7.0	0.000(1)	0.067	0.074	6.4
tai50a	4938796	0.000(1)	0.181	0.364	14.5	0.053(0)	0.216	0.317	12.8	0.000(1)	0.279	0.415	17.3
tai60a	7205962	0.273(0)	0.371	0.442	17.7	0.165(0)	0.285	0.381	17.6	0.165(0)	0.377	0.469	8.7
tai80a	13499184	0.474(0)	0.571	0.647	13.5	0.468(0)	0.553	0.621	14.5	0.430(0)	0.516	0.614	18.2
tai100a	21052466	0.467(0)	0.566	0.630	16.4	0.389(0)	0.510	0.623	19.0	0.311(0)	0.402	0.553	13.4
tai50b	458821517	0.000(10)	0.000	0.000	0.2	0.000(10)	0.000	0.000	0.2	0.000(10)	0.000	0.000	0.1
tai60b	608215054	0.000(10)	0.000	0.000	0.6	0.000(10)	0.000	0.000	0.3	0.000(10)	0.000	0.000	0.5
tai80b	818415043	0.000(10)	0.000	0.000	3.3	0.000(10)	0.000	0.000	1.8	0.000(10)	0.000	0.000	1.2
tai100b	1185996137	0.000(9)	0.005	0.045	10.3	0.000(6)	0.044	0.143	2.4	0.000(6)	0.040	0.100	4.0
tai150b	498896643	0.014(0)	0.219	0.390	14.5	0.000(1)	0.137	0.316	21.5	0.000(1)	0.191	0.321	20.5
sko72	66256	0.000(10)	0.000	0.000	4.4	0.000(9)	0.006	0.063	0.7	0.000(10)	0.000	0.000	1.4
sko81	90998	0.000(9)	0.001	0.011	8.0	0.000(10)	0.000	0.000	3.5	0.000(10)	0.000	0.000	2.7
sko90	115534	0.000(4)	0.023	0.095	9.5	0.000(9)	0.004	0.038	5.9	0.000(6)	0.015	0.038	4.2
sko100a	152002	0.000(4)	0.006	0.018	12.4	0.000(8)	0.003	0.016	5.2	0.000(10)	0.000	0.000	7.7
sko100b	153890	0.000(8)	0.001	0.004	4.6	0.000(10)	0.000	0.000	8.5	0.000(10)	0.000	0.000	7.7
sko100c	147862	0.000(7)	0.001	0.004	10.2	0.000(10)	0.000	0.000	6.9	0.000(10)	0.000	0.000	8.7
sko100d	149576	0.000(3)	0.004	0.009	13.5	0.000(9)	0.007	0.066	6.6	0.000(10)	0.000	0.000	9.0
sko100e	149150	0.000(5)	0.002	0.005	13.7	0.000(10)	0.000	0.000	6.5	0.000(7)	0.001	0.004	10.3
sko100f	149036	0.000(5)	0.006	0.032	11.5	0.000(6)	0.005	0.021	6.3	0.000(7)	0.003	0.021	4.4
wil100	273038	0.000(6)	0.001	0.003	11.4	0.000(10)	0.000	0.000	6.4	0.000(10)	0.000	0.000	9.5
tho150	8133398	0.013(0)	0.091	0.128	11.9	0.003(0)	0.021	0.067	19.9	0.000(1)	0.051	0.123	24.2
avg.		0.059	0.101	0.138	9.8	0.051	0.095	0.139	**8.2**	**0.043**	**0.092**	**0.130**	8.6

respectively. Similar observations can also be found for the APD and WPD indicators. It is worth noting that FPBS-QAP needs less time to achieve these results than BLS, but consumes nearly the same computing time as BMA.

When a long time limit $t_{max} = 120$ minutes is allowed, our FPBS-QAP algorithm is able to achieve even better results. As we can see from Table 5.5, the best-known values are obtained with a higher success rate compared to the results under $t_{max} = 30$ minutes in Table 5.4. The BPD value of FPBS-QAP is 0.028%, which is the smallest compared to 0.037% of BMA, and 0.038% of BLS. FPBS-QAP also achieves the smallest average APD value and average WPD value. As to the computing times, FPBS-QAP requires on average 22.0 minutes to reach its best solution, which is the shortest time among the compared algorithms (32.2 minutes for BLS, and 23.1 minutes for BMA).

In summary, our FPBS-QAP algorithm competes favorably with the two best-performing QAP algorithms (i.e., BLS and BMA) in terms of both solution quality and computing time. The computational results demonstrate the effectiveness of our proposed FPBS-QAP algorithm, and further shows the usefulness of using frequent patterns mined from high-quality solutions to guide the search for an effective exploration of the solution space.

5.5.4 Comparison with state-of-the-art algorithms

We now extend our experimental study by comparing FPBS-QAP with four other very recent state-of-the-art QAP algorithms in the literature.

— Parallel hybrid algorithm (PHA) [Tosun, 2015] (2015) used the MPI libraries and was implemented on a high-performance cluster computer which has 46 nodes, each with 2 CPUs giving 92 CPUs. Each CPU has 4 cores giving a total of 368 cores. Each node has 16 GB if RAM giving 736 GB of total memory, and a total disk capacity of 6.5 TB configured in a high performance RAID.

— Two-stage memory powered great deluge algorithm (TMSGD) [Acan and Ünveren, 2015] (2015) was implemented on a personal computer with 2.1 GHz and 8 GB RAM. The algorithm was stopped when the number of fitness evaluations reaches $20000 * n$ (n is the instance size).

Table 5.5: Comparative performance of FPBS-QAP with BLS and BMA on 21 hard instances under $t_{max} = 120$ minutes. The success rate of reaching the best-known value over 10 runs is indicated in parentheses.

		BLS				BMA				FPBS-QAP			
Instance	BKV	BPD	APD	WPD	$T(m)$	BPD	APD	WPD	$T(m)$	BPD	APD	WPD	$T(m)$
tai40a	3139370	0.000(7)	0.022	0.074	40.7	0.000(5)	0.037	0.074	28.9	0.000(7)	0.022	0.074	52.5
tai50a	4938796	0.000(1)	0.100	0.245	47.0	0.000(3)	0.098	0.291	38.8	0.000(2)	0.107	0.231	67.8
tai60a	7205962	0.036(0)	0.233	0.331	73.9	0.165(0)	0.221	0.352	34.7	0.000(1)	0.216	0.300	60.0
tai80a	13499184	0.416(0)	0.502	0.587	58.0	0.332(0)	0.428	0.505	69.9	0.313(0)	0.451	0.618	55.2
tai100a	21052466	0.335(0)	0.460	0.560	58.4	0.223(0)	0.370	0.511	59.9	0.280(0)	0.378	0.466	36.1
tai50b	458821517	0.000(10)	0.000	0.000	0.2	0.000(10)	0.000	0.000	0.1	0.000(10)	0.000	0.000	0.2
tai60b	608215054	0.000(10)	0.000	0.000	0.3	0.000(10)	0.000	0.000	0.4	0.000(10)	0.000	0.000	0.4
tai80b	818415043	0.000(10)	0.000	0.000	3.9	0.000(10)	0.000	0.000	2.0	0.000(10)	0.000	0.000	1.4
tai100b	1185996137	0.000(10)	0.000	0.000	6.1	0.000(6)	0.040	0.100	8.1	0.000(10)	0.000	0.000	2.9
tai150b	498896643	0.001(0)	0.040	0.183	42.7	0.055(0)	0.213	0.414	56.0	0.000(5)	0.099	0.313	46.4
sko72	66256	0.000(10)	0.000	0.000	3.0	0.000(10)	0.000	0.000	0.7	0.000(10)	0.000	0.000	4.8
sko81	90998	0.000(10)	0.000	0.000	10.3	0.000(10)	0.000	0.000	3.2	0.000(10)	0.000	0.000	3.3
sko90	115534	0.000(10)	0.000	0.000	19.6	0.000(9)	0.004	0.038	3.6	0.000(9)	0.004	0.038	2.4
sko100a	152002	0.000(10)	0.000	0.000	51.0	0.000(8)	0.003	0.016	28.6	0.000(10)	0.000	0.000	8.5
sko100b	153890	0.000(10)	0.000	0.000	21.6	0.000(10)	0.000	0.000	11.0	0.000(10)	0.000	0.000	5.8
sko100c	147862	0.000(10)	0.000	0.000	22.4	0.000(10)	0.000	0.000	7.1	0.000(10)	0.000	0.000	8.7
sko100d	149576	0.000(6)	0.001	0.005	38.5	0.000(10)	0.000	0.000	12.6	0.000(10)	0.000	0.000	16.2
sko100e	149150	0.000(10)	0.000	0.000	44.2	0.000(10)	0.000	0.000	5.3	0.000(10)	0.000	0.000	12.2
sko100f	149036	0.000(7)	0.002	0.005	40.2	0.000(8)	0.001	0.005	23.7	0.000(6)	0.002	0.005	4.0
wil100	273038	0.000(9)	0.000	0.002	28.9	0.000(10)	0.000	0.000	6.9	0.000(10)	0.000	0.000	16.4
tho150	8133398	0.009(0)	0.056	0.116	64.2	0.000(1)	0.036	0.065	83.9	0.000(3)	0.008	0.064	57.4
avg.		0.038	0.067	**0.100**	32.2	0.037	0.069	0.113	23.1	**0.028**	**0.061**	**0.100**	**22.0**

— Parallel multi-start hyper-heuristic algorithm (MSH) [Dokeroglu and Cosar, 2016] (2016) was implemented on a high performance cluster computer as well as the above PHA algorithm.

— Breakout local search using OpenMP (BLS-OpenMP) [Aksan *et al.*, 2017] (2017) was implemented based on OpenMP (i.e., an API for shared-memory parallel computations and works on multi-core computers.) and was executed on a personal computer with an Intel Core i7-6700 CPU 3.4 GHZ with 4 cores, 16 GB RAM. It is possible to execute 8 logical processors on this computer.

One notices that three of these four recent QAP algorithms are implemented and evaluated on parallel machines. Moreover, their results have been obtained under different computing platforms, with different stopping conditions. For example, TMSGD terminates when the number of fitness evaluations reaches a fixed value, while BLS-OpenMP terminates after a fixed number of iterations.

Table 5.6 presents the comparative results between the proposed FPBS-QAP algorithm and the four reference algorithms. Like [Acan and Ünveren, 2015; Dokeroglu and Cosar, 2016; Aksan *et al.*, 2017], we adopt the APD indicator (defined in Section 5.5.2) for this comparative study. In the table, we also include the running time ($T(m)$), which is presented only for indicative purposes. For completeness, we also include the results of BLS and BMA from the Table 5.6. At the end of the table, we again indicate the average value of each indicator.

From Table 5.6, we observe that our FPBS-QAP algorithm achieves a highly competitive performance compared to the state-of-the-art algorithms. The average APD value of FPBS-QAP is 0.061%, which is only slightly worse than 0.058% of PHA and better than all remaining reference algorithms. Moreover, PHA is a parallel algorithm, and its average computing time (≥ 177.4 minutes) is significantly than that of FPBS-QAP (≤ 67.8 minutes). Note that the results of three instances, including two hardest and largest instances tai150b and tho150, are not reported for BLS-OpenMP. The APD value of BLS-OpenMP is computed for the remaining 18 instances. Importantly, our algorithm requires the least time to achieve the best results, and its average time is only 22.0 minutes. These observations show that our FPBS-QAP algorithm is highly competitive compared to the state-of-the-art algorithms in terms of solution quality and computing time.

Table 5.6: Comparative performance between our FPBS-QAP and the state-of-the-art algorithms on hard instances in terms of the APD value. Computational time are given in minutes for indicative purposes.

		BLS⋆		PHA°		BMA⋆		TMSGD		MSH°		BLS-OpenMP°		FPBS-QAP	
Instance	BKV	APD	$T(m)$	APD	$T(m)$	APD	$T(m)$	APD	$T(m)$	APD	$T(m)$	APD	$T(m)$	APD	$T(m)$
tai40a	3139370	0.022	40.7	0.000	10.6	0.037	28.9	0.261	27.8	0.261	30.0	0.000	32.2	0.022	52.5
tai50a	4938796	0.100	47.0	0.000	12.7	0.098	38.8	0.276	41.1	0.165	37.5	0.000	68.2	0.107	67.8
tai60a	7205962	0.233	73.9	0.000	19.6	0.221	34.7	0.448	78.9	0.270	45.0	0.000	107.9	0.216	60.0
tai80a	13499184	0.502	58.0	0.644	40.0	0.428	69.9	0.832	111.3	0.530	60.0	0.504	236.0	0.451	55.2
tai100a	21052466	0.460	58.4	0.537	71.9	0.370	59.9	0.874	138.3	0.338	75.0	0.617	448.5	0.378	36.1
tai50b	458821517	0.000	0.2	0.000	5.8	0.000	0.1	0.005	10.2	0.000	3.0	0.000	0.7	0.000	0.2
tai60b	608215054	0.000	0.3	0.000	9.5	0.000	0.4	0.000	33.6	0.000	3.2	0.000	18.6	0.000	0.4
tai80b	818415043	0.000	3.9	0.000	27.7	0.000	2.0	0.025	0.0	0.000	4.0	0.000	218.1	0.000	1.4
tai100b	1185996137	0.000	6.1	0.000	42.5	0.040	8.1	0.028	72.6	0.000	5.0	0.000	160.8	0.000	2.9
tai150b	498896643	0.040	42.7	0.026	177.4	0.213	56.0	0.051	258.0	*	*	*	*	0.099	46.4
sko72	66256	0.000	3.0	0.000	33.6	0.000	0.7	0.007	38.0	0.000	3.6	0.000	1.8	0.000	4.8
sko81	90998	0.000	10.3	0.000	39.9	0.000	3.2	0.019	57.1	0.000	4.1	0.000	2.4	0.000	3.3
sko90	115534	0.000	19.6	0.000	40.5	0.004	3.6	0.031	93.8	0.000	4.5	0.000	3.3	0.004	2.4
sko100a	152002	0.000	51.0	0.000	41.7	0.003	28.6	0.029	153.2	0.003	75.0	0.000	29.8	0.000	8.5
sko100b	153890	0.000	21.6	0.000	42.3	0.000	11.0	0.015	164.3	0.004	75.0	0.000	8.5	0.000	5.8
sko100c	147862	0.000	22.4	0.000	42.2	0.000	7.1	0.013	154.5	0.003	75.0	0.000	4.3	0.000	8.7
sko100d	149576	0.001	38.5	0.000	41.9	0.000	12.6	0.017	148.9	0.004	75.0	0.000	12.9	0.000	16.2
sko100e	149150	0.000	44.2	0.000	42.5	0.000	5.3	0.016	146.1	0.000	75.0	0.000	4.3	0.000	12.2
sko100f	149036	0.002	40.2	0.000	42.0	0.001	23.7	0.013	153.4	0.000	75.0	0.000	17.1	0.002	4.0
wil100	273038	0.000	28.9	0.000	42.0	0.000	6.9	0.008	155.1	*	*	*	*	0.000	16.4
tho150	8133398	0.056	64.2	0.009	177.4	0.036	83.9	0.039	512.8	*	*	*	*	0.008	57.4
avg.		0.067	32.2	**0.058**	47.8	0.069	23.1	0.143	121.4	0.088	40.3	0.062	76.4	0.061	**22.0**

⋆ The results of BLS and BMA are obtained by re-running the programs on our platform with $t_{max} = 120$ minutes, which are slightly different from the results reported in [Benlic and Hao, 2013; Benlic and Hao, 2015].
° PHA, MSH and BLS-OpenMP are parallel algorithms.

5.6 Experimental analysis

This section is devoted to performing additional experiments to gain some understanding of the proposed FPBS algorithm including the rationale behind the solution construction based on frequent patterns, the effectiveness of the solution construction based on mined frequent patterns, and the impact of the number of largest patterns m on the performance of the proposed FPBS algorithm.

5.6.1 Rationale behind the solution construction based on mined patterns

To explain the rationale behind the solution construct based on mined frequent patterns, we analyze the structural similarity between high-quality solutions in the elite set, and the length distribution of the frequent patterns mined from the elite set. Given two high-quality solutions π^s and π^t, we define their similarity as follows:

$$sim(\pi^s, \pi^t) = \frac{|\pi^s \cap \pi^t|}{n} \tag{5.2}$$

where $\pi^s \cap \pi^t$ is the set of common elements shared by π^s and π^t. The larger the similarity between two solutions, the more common elements they share.

As we mentioned above, a mined frequent pattern represents a set of same elements shared by two or more solutions under a given minimum support θ. A frequent pattern can be directly converted to a partial solution, thus we define the length of a pattern p as follows.

$$len(p) = \frac{|p|}{n} \tag{5.3}$$

where the length of a pattern is the number of same elements. The larger the length of the pattern, the more common elements they share. The solution similarity defined in Equation (5.2) can be considered as a special case of the pattern length defined in Equation (5.3). Specifically, when the support value of

a mined pattern equals to 2, the pattern is simplified as the set of common elements shared by only two solutions. It can be further confirmed according to the results reported in Figure 5.5, where the curve of the maximum solution similarity (left sub-figure) is exactly the same as the curve of the maximum length (right sub-figure).

In the experiments, we solve each benchmark instance one time with a time limit of $t_{max} = 30$ minutes. Once the time limit is reached, we analyze the solution similarity of PS high-quality solutions stored in the elite set according to Equation (5.2), and calculate the length distribution of a set of frequent patterns mined from the elite set according to Equation (5.3). The computational results of the solution similarity between high-quality solutions and the length distribution of the mined frequent patterns are presented in Figure 5.5.

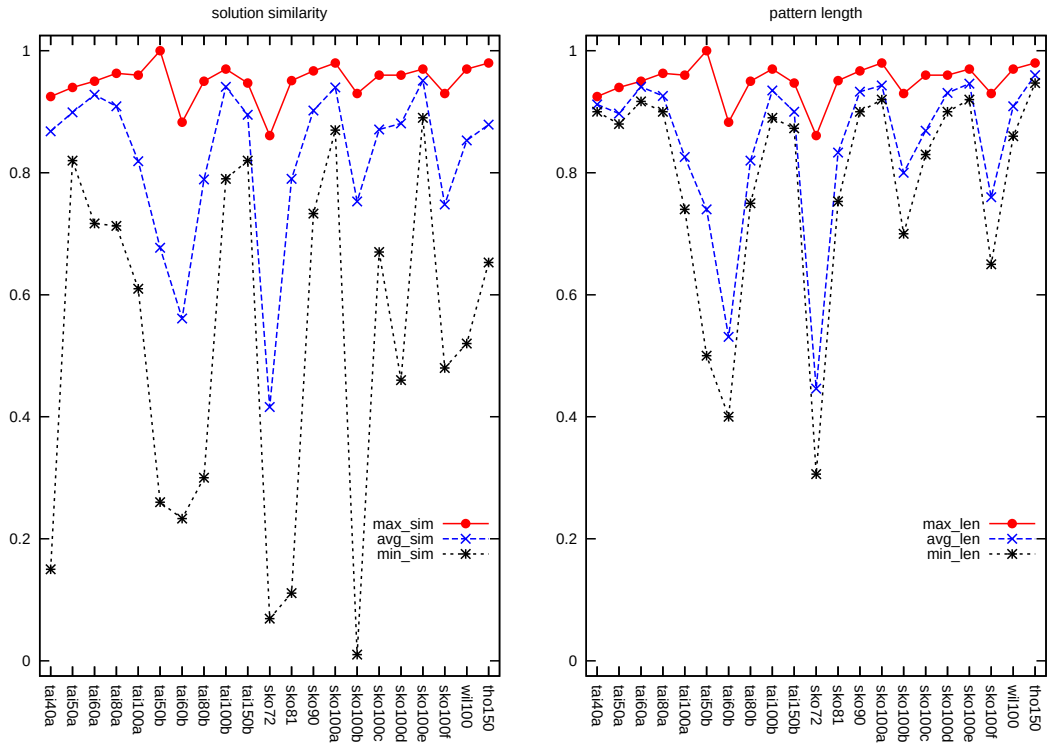

Figure 5.5: Solution similarity between high-quality solutions (left sub-figure) and the length distribution of the mined pattern (right sub-figure).

In Figure 5.5, we report the maximum value, average value, and minimum value of the solution similarity and the pattern length, respectively. We can clearly observe that there is a high similarity between these high-quality solutions. Specifically, it is true for all instances that the maximum solution similarity is larger than 0.9. Also, the average solution similarities between any two high-quality solutions are larger than 0.5 except for sko72. While for instance sko72, its average solution similarity is about 0.4. A more significant observation can be derived based on the lengths of the mined patterns showed in the right side of Figure 5.5. The high structural similarities between the high-quality solutions provide the rationale behind our solution construction based on mined pattern.

5.6.2 Effectiveness of the solution construction based on frequent pattern

The frequent pattern based solution construction method is a good alternative to the general crossover operator in evolutionary algorithms and memetic algorithms. To demonstrate the effectiveness of the solution construction based on frequent pattern, we compare it with the general crossover operator within the framework of our FPBS-QAP algorithm. In the experiments, we compare FPBS-QAP with its alternative version FPBS-QAP$_0$. FPBS-QAP$_0$ is obtained from FPBS-QAP by replacing the frequent pattern based solution construction by the standard uniform crossover operator used in [Benlic and Hao, 2015]. For each

algorithm, we run it on each benchmark instance 10 times with a time limit $t_{max} = 30$ minutes. The comparative results between FPBS-QAP and FPBS-QAP$_0$ are summarized in Table 5.7.

Table 5.7: Comparisons between FPBS-QAP$_0$ and FPBS-QAP on hard instances under $t_{max} = 30$ minutes. The success rate of reaching the best known value over 10 runs is indicated in parentheses.

		FPBS-QAP$_0$*				FPBS-QAP			
Instance	BKV	BPD	APD	WPD	$T(m)$	BPD	APD	WPD	$T(m)$
tai40a	3139370	0.000(2)	0.059	0.074	7.4	0.000(1)	0.067	0.074	6.4
tai50a	4938796	0.241(0)	0.318	0.392	11.8	0.000(1)	0.279	0.415	17.3
tai60a	7205962	0.164(0)	0.334	0.486	13.1	0.165(0)	0.377	0.469	8.7
tai80a	13499184	0.446(0)	0.533	0.622	15.7	0.430(0)	0.516	0.614	18.2
tai100a	21052466	0.316(0)	0.466	0.615	16.8	0.311(0)	0.402	0.553	13.4
tai50b	458821517	0.000(10)	0.000	0.000	0.1	0.000(10)	0.000	0.000	0.1
tai60b	608215054	0.000(10)	0.000	0.000	0.3	0.000(10)	0.000	0.000	0.5
tai80b	818415043	0.000(10)	0.000	0.000	1.5	0.000(10)	0.000	0.000	1.2
tai100b	1185996137	0.000(8)	0.018	0.100	3.1	0.000(6)	0.040	0.100	4.0
tai150b	498896643	0.000(1)	0.204	0.358	20.2	0.000(1)	0.191	0.321	20.5
sko72	66256	0.000(9)	0.006	0.063	2.2	0.000(10)	0.000	0.000	1.4
sko81	90998	0.000(10)	0.000	0.000	5.1	0.000(10)	0.000	0.000	2.7
sko90	115534	0.000(9)	0.004	0.038	4.1	0.000(6)	0.015	0.038	4.2
sko100a	152002	0.000(9)	0.002	0.016	7.0	0.000(10)	0.000	0.000	7.7
sko100b	153890	0.000(8)	0.001	0.004	6.0	0.000(10)	0.000	0.000	7.7
sko100c	147862	0.000(10)	0.000	0.000	6.0	0.000(10)	0.000	0.000	8.7
sko100d	149576	0.000(10)	0.000	0.000	9.5	0.000(10)	0.000	0.000	9.0
sko100e	149150	0.000(10)	0.000	0.000	7.6	0.000(7)	0.001	0.004	10.3
sko100f	149036	0.000(7)	0.002	0.005	3.6	0.000(7)	0.003	0.021	4.4
wil100	273038	0.000(9)	0.000	0.002	6.0	0.000(10)	0.000	0.000	9.5
tho150	8133398	0.002(0)	0.022	0.080	19.1	0.000(1)	0.051	0.123	24.2
avg.		0.056	0.094	0.136	**7.9**	**0.043**	**0.092**	**0.130**	8.6

* FPBS-QAP$_0$ can also be considered as an alternative version of the BMA [Benlic and Hao, 2015] by removing the mutation procedure.

From Table 5.7, we can clearly observe that FPBS-QAP performs better than FPBS-QAP$_0$. FPBS-QAP is able to achieve the better or the same BPD value on all instances except tai60a. While for instance tai60a, the BPD value of FPBS-QAP is 0.165%, which is only slightly worse than 0.164% achieved by FPBS-QAP$_0$. We can also observe that the average BPD value of our FPBS-QAP algorithm is smaller than that of FPBS-QAP$_0$, i.e., $0.043\% < 0.056\%$. Our FPBS-QAP algorithm also achieves better results in terms of the average APD value and the average WPD value. To achieve these results, the average running time of FPBS-QAP$_0$ is slightly shorter than that of FPBS-QAP (i.e., $7.9 < 8.6$ minutes). It seems reasonable because FPBS-QAP needs to conditionally execute a frequent pattern mining procedure during the search. These observations conforms the effectiveness of the solution construction based on mined frequent pattern.

5.6.3 Impact of the number of the largest patterns m

The number of the largest frequent patterns $m (m \geqslant 1)$ is an important parameter of the proposed FPBS-QAP algorithm. The m value decides the diversity of the new solutions constructed by our solution construction method based on frequent patterns. To investigate the impact of different m values, we varied the values of m within a reasonable range and compared their performances. The box and whisker plots showed in Figure 5.6 are obtained by considering ten different values $m \in \{1, 3, \ldots, 21\}$. The experiments were conducted on four representative instances selected from different families (tai100a, tai150b, sko100f and tho150). For each algorithm variant, we ran it on each instance 10 times with the stopping condition $t_{max} = 30$ minutes.

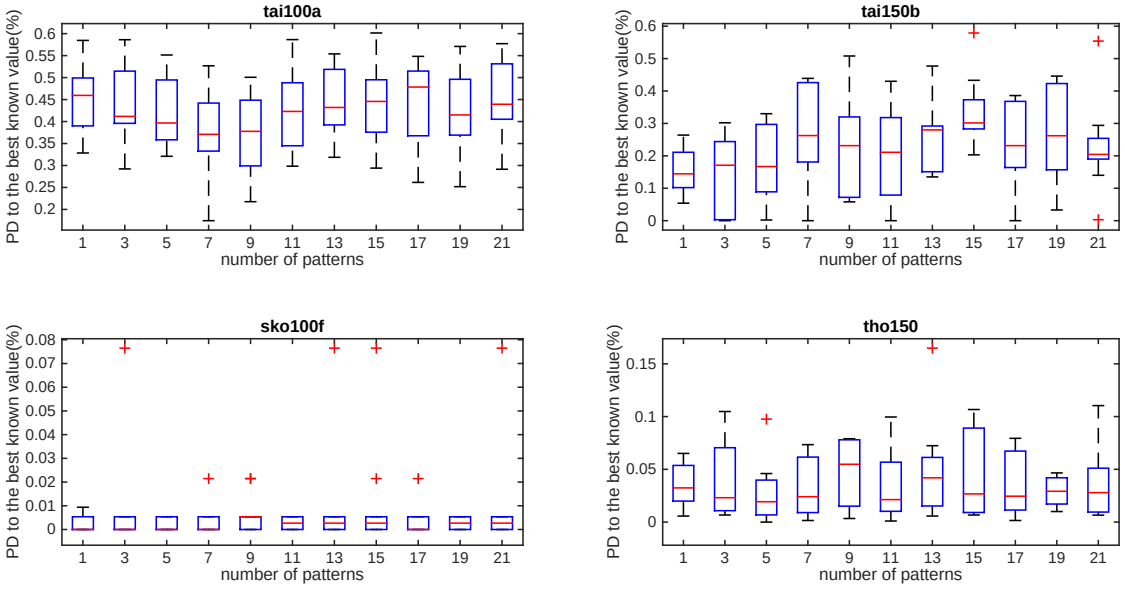

Figure 5.6: Box and whisker plots corresponding to different values of $m \in \{1, 3, \ldots, 21\}$ in terms of the percentage deviation (PD) to the best known value.

In Figure 5.6, X-axis indicates the different values for the largest pattern m and Y-axis shows the performance (i.e., the percentage deviation to the best known value). We observe that the performance of the FPBS-QAP algorithm strongly depends on the m value except for sko100f. FPBS-QAP has a good performance when the number of the largest pattern m is fixed to 11. This justifies the default value for m shown in Table 5.3.

To tune parameters β and max_no_update, we used the same approach and chosen $\beta = 0.75$, $\max\_no\_update = 15$ as their default values. It is possible that FPBS-QAP improves its performance when its parameters are tuned for each specific problem instance.

5.7 Chapter conclusion

In this chapter, we proposed a general-purpose optimization approach called Frequent Pattern Based Search (FPBS). The proposed approach relies on a data mining procedure to mine frequent patterns from high-quality solutions collected during the search. The mined patterns are then used to create new starting solutions for further improvements. By iterating the pattern mining phase and the optimization phase, FPBS is designed to ensure an effective exploration of the combinatorial search space.

The viability of the proposed approach was verified on the well-known Quadratic Assignment Problem (QAP). Extensive computational results on popular QAPLIB benchmarks showed that FPBS performs remarkably well compared to very recent and state-of-the-art algorithms both in terms of solution quality and computational efficiency. Specifically, our approach is able to find the best-known objective values for all the benchmark instances except tai80a and tai100a within a time limit of 0.5 hour or 2 hours. To the best of our knowledge, very few QAP algorithms can achieve such a performance. Furthermore, we performed additional experiments to investigate three key issues of the proposed FPBS algorithm.

General Conclusion

Conclusions

This thesis investigated combinations of machine learning techniques and meta-heuristics for solving three categories of Combinatorial Optimization Problems (COPs), including grouping problems such as Graph Coloring Problem (GCP), subset selection problems such as Maximum Diversity Problem (MDP), and permutation problems such as Quadratic Assignment Problem (QAP). Our proposed hybrid meta-heuristics belong to the family of learning-driven heuristic optimization approaches. Recently, the topic of learning-driven heuristic optimization has attracted increasing attention in combinatorial optimization. Based on three different machine learning techniques, i.e., probability learning, opposition-based learning and frequent pattern mining, we proposed three learning-driven heuristic optimization approaches for solving each category of COPs respectively. That is, a probability learning based local search for grouping problems, specially for GCP; an opposition-based memetic algorithm for MDP; and a frequent pattern based search approach for QAP.

In **Chapter 2**, we briefly introduced the learning-driven heuristic optimization which uses machine learning to help heuristic optimization. We broadly divided the learning-driven optimization algorithms into four categories according to the purpose of introducing machine learning techniques: using machine learning to improve the solution quality, speeding up the heuristic search, optimizing algorithm parameters, and conducting algorithm selection.

In **Chapter 3**, we developed a Probability learning based Local Search (PLS) approach for solving the class of grouping problems. The proposed PLS approach combines a probability learning technique with an optimization procedure. Probability learning is used to maintain and update a set of probability vectors, each probability vector specifying the probability that an item belongs to a particular group. At each iteration, PLS builds a starting grouping solution according to the probability vectors and with the help of a group selection strategy. PLS then applies a descent-based local search procedure to improve the given grouping solution until a local optimum is reached. At this point, the starting solution and the ending local optimum solution are compared to update the probability vector of each item according to the situation of the item. Experimental analyses and performance assessments of the PLS approach were carried out on GCP (denoted as PLSCOL) which is a well-known grouping problem. Compared to PLS, PLSCOL has two enhancements. The first enhancement improves the probability learning scheme of the original PLS approach by using a group matching procedure to find the group-to-group relation between a starting solution and its improved solution. This matching procedure copes with the difficulty raised by symmetric solutions of GCP. The second enhancement concerns the coloring algorithm based on tabu search, which is more powerful than the descent-based coloring algorithm used in PLS. Experimental evaluations on popular DIMACS challenge benchmark instances showed that PLSCOL competes favorably with all existing local search based coloring algorithms. Compared with the most effective hybrid evolutionary algorithms which are much more sophisticated in their design, PLSCOL remains competitive in spite of the simplicity of its underlying coloring algorithm. Our additional experimental investigations also showed that 1) probability learning scheme is highly valuable to increase the performance of the proposed PLSCOL; 2) the probability smoothing technique which forgets old decisions is very useful to avoid search traps; 3) group matching procedure is able to provide correct feedback information to the probability learning component; and 4) the

hybrid group selection strategy combining randomness and greediness is more suitable than other selection strategies.

In **Chapter 4**, we presented an Opposition-Based Memetic Algorithm (OBMA) which uses Opposition-Based Learning (OBL) to improve a memetic algorithm for solving MDP. The OBMA algorithm employs OBL to reinforce population diversity and improve evolutionary search. OBMA distinguishes itself from existing memetic algorithms by three aspects: a double trajectory search procedure which simultaneously considers both a candidate solution and a corresponding opposite solution, a parametric constrained neighborhood for effective local optimization, and a rank-based quality-and-distance pool updating strategy. Extensive comparative experiments on 80 large benchmark instances (with 2000 to 5000 items) from the literature have demonstrated the competitiveness of the proposed OBMA algorithm. OBMA matches the best-known results for most of instances and in particular finds improved best results (new lower bounds) for 22 instances which are useful for the assessment of other MDP algorithms. Our experimental analysis has also confirmed that integrating OBL into the memetic search framework does improve the search efficiency of the classical memetic search.

In **Chapter 5**, we proposed a general-purpose optimization approach called Frequent Pattern Based Search (FPBS). The proposed approach relies on a data mining procedure to mine frequent patterns from high-quality solutions collected during the search. The mined patterns are then used to create new starting solutions for further improvements. By iterating the pattern mining phase and the optimization phase, FPBS is designed to ensure an effective exploration of the combinatorial search space. The viability of the proposed approach was verified on the well-known QAP. Extensive computational results on popular QAPLIB benchmarks showed that FPBS performs remarkably well compared to very recent and state-of-the-art algorithms both in terms of solution quality and computational efficiency. Specifically, our approach is able to find the best-known objective values for all the benchmark instances except tai80a and tai100a within a time limit of 0.5 hour or 2 hours. To the best of our knowledge, very few QAP algorithms can achieve such a performance. Furthermore, we performed experiments to investigate some key issues of the proposed FPBS algorithm.

Perspectives

The main objective of this thesis is to design learning-driven heuristic optimization approaches for solving hard combinatorial search problems. We have proposed three learning-driven heuristic optimization algorithms to solve classic $\mathcal{NP}$-hard problems. Our contributions and the observations made in this thesis also pose a number of interesting open questions for future work.

— **Extend to solve other COPs**: Our proposed learning-driven heuristic optimization methods are general-purpose methods, which can be applied to solve a wide family of COPs. For example, we introduce the concept of opposition-based learning into the Maximum Diversity Problem (MDP), and propose an effective Opposition-Based Memetic Algorithm (OBMA) in chapter 4. MDP is a representative subset selection problem, and there are many similar problems in the field of combinatorial optimization. It would be very reasonable to apply OBMA to solve other subset selection problems, such as critical node problems [Zhou and Hao, 2017a; Zhou *et al.*, 2017e], and other dispersion or diversity problems [Prokopyev *et al.*, 2009].

— **Improve with parallel computing and big data**: Sequential heuristic algorithms have achieved significant progress in combinatorial optimization and they can find very high-quality solutions. However, the complexity and computational requirements of COPs are becoming increasingly high. Parallel techniques appear to be an attractive and effective means to solve hard problems. They should be able to provide better solutions and reduce the computational time required and allow new powerful algorithms to be designed and tested. Moreover, high performance computers as well as computing platforms such as CUDA and OpenMPC (Open MP extended for CUDA) become widely available, exploring the performance of parallel meta-heuristics has become an important research direction

[Alba *et al.*, 2013]. It would be very interesting to implement our learning-driven heuristic optimization algorithms in a parallel way. For our Frequent Pattern Based Search (FPBS) approach proposed in Chapter 5, the frequent patterns are mined from only a small size of high-quality solutions. With the help of big data platform, we are able to investigate the frequent patterns mined from a large size of high-quality solutions. Moreover, FPBS can be implemented in a parallel way because it has a highly parallel-able computational architecture. That it, for each mined frequent pattern, the corresponding solution construction and the optimization procedure can be executed in parallel on a high performance computing platform.

— **Deal with stochastic COPs**: In this thesis, we only considered the deterministic versions of the corresponding COPs. They frequently assume that the problem inputs, the objective function, and the set of optimization constraints are deterministic. However, uncertainty is ubiquitous in real-world applications. Importantly, machine learning techniques can be helpful to discover potential relationships between the search space and the objective function that the traditional methods ignore. For example, reinforcement learning is the study of planning and learning in a scenario, where a learner (i.e., an agent) actively interacts with an unknown environment to achieve a certain goal [Sutton and Barto, 1998]. It would be very interesting to combine machine learning techniques with heuristic algorithms to solve stochastic COPs.

List of Figures

List of Tables

List of Publications

Published/accepted journal papers

1. Yangming Zhou, Jin-Kao Hao and Béatrice Duval. Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications* 64:412-422, 2016.

2. Yangming Zhou and Jin-Kao Hao. An iterated local search for minimum differential dispersion probelm. *Knowledge-Based Systems* 125:26-38, 2017.

3. Yangming Zhou, Jin-Kao Hao and Béatrice Duval. Opposition-based memetic search for the maximum diversity problem. *IEEE Transactions on Evolutionary Computation*, 21(5):731-745, 2017.

Published/accepted conference papers

1. Yangming Zhou and Jin-Kao Hao. A fast heuristic algorithm for the critical node problem. in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, Berlin, Germany, July 15-19, 2017, 121-122.

Revised/submitted journal papers

1. Yangming Zhou, Béatrice Duval and Jin-Kao Hao. Improving probability learning based local search for graph coloring. *Applied Soft Computing*, Revised, September, 2017.

2. Yangming Zhou, Jin-Kao Hao and Fred Glover. Memetic search for identifying critical nodes in sparse graphs. *arXiv:1705.04119*, Submitted, May, 2017.

3. Yangming Zhou, Jin-Kao Hao and Béatrice Duval. When data mining meets optimization: A case study on the quadratic assignment problem. *arXiv:1708.05214*, Submitted, August, 2017.

References

[Acan and Ünveren, 2015] Adnan Acan and Ahmet Ünveren. A great deluge and tabu search hybrid with two-stage memory support for quadratic assignment problem. *Applied Soft Computing*, 36:185–203, 2015. 85, 92, 93, 94

[Adenso-Diaz and Laguna, 2006] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006. 27

[Aggarwal *et al.*, 2014] Charu C Aggarwal, Mansurul A Bhuiyan, and Mohammad Al Hasan. Frequent pattern mining algorithms: A survey. In *Frequent Pattern Mining*, pages 19–64. Springer, 2014. 21, 83, 84

[Agrawal and Srikant, 1994] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. 21

[Agrawal *et al.*, 1993] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM. 21, 79, 80

[Agustı *et al.*, 2012] LE Agustı, Sancho Salcedo-Sanz, Silvia Jiménez-Fernández, Leopoldo Carro-Calvo, Javier Del Ser, José Antonio Portilla-Figueras, et al. A new grouping genetic algorithm for clustering problems. *Expert Systems with Applications*, 39(10):9695–9703, 2012. 33

[Ahmed, 2015] Zakir Hussain Ahmed. A multi-parent genetic algorithm for the quadratic assignment problem. *Opsearch*, 52(4):714–732, 2015. 85

[Aksan *et al.*, 2017] Yagmur Aksan, Tansel Dokeroglu, and Ahmet Cosar. A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering*, 103:105–115, 2017. 85, 94

[Al-Duoli and Rabadi, 2014] Fatemah Al-Duoli and Ghaith Rabadi. Data mining based hybridization of meta-RaPS. *Procedia Computer Science*, 36:301–307, 2014. 27

[Al-Qunaieer *et al.*, 2010] Fares S. Al-Qunaieer, Hamid R. Tizhoosh, and Shahryar Rahnamayan. Opposition based computing - A survey. In *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*, pages 1–7, 2010. 56

[Alba *et al.*, 2013] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013. 101

[Anstreicher *et al.*, 2002] Kurt Anstreicher, Nathan Brixius, Jean-Pierre Goux, and Jeff Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, 91(3):563–588, 2002. 85

[Arin and Rabadi, 2016] Arif Arin and Ghaith Rabadi. Integrating estimation of distribution algorithms versus Q-learning into Meta-RaPS for solving the 0-1 multidimensional knapsack problem. *Computers & Industrial Engineering*, 2016. 25

[Aringhieri and Cordone, 2011] Roberto Aringhieri and Roberto Cordone. Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society*, 62(2):266–280, 2011. 55, 63

[Aringhieri *et al.*, 2008] Roberto Aringhieri, Roberto Cordone, and Yari Melzani. Tabu search versus GRASP for the maximum diversity problem. *4OR: A Quarterly Journal of Operations Research*, 6(1):45–60, 2008. 55, 63

[Aringhieri *et al.*, 2009] Roberto Aringhieri, Maurizio Bruglieri, and Roberto Cordone. Optimal results and tight bounds for the maximum diversity problem. *Foundation of Computing and Decision Sciences*, 34(2):73–85, 2009. 55

[Aziz and Tayarani-N., 2016] Mahdi Aziz and Mohammad-H. Tayarani-N. Opposition-based magnetic optimization algorithm with parameter adaptation strategy. *Swarm and Evolutionary Computation*, 26:97–119, 2016. 56

[Baluja *et al.*, 2000] S Baluja, A Barto, KD Boese, J Boyan, W Buntine, T Carson, R Caruana, D Cook, S Davies, T Dean, et al. Statistical machine learning for large-scale optimization. *Neural Computing Surveys (CSUR)*, 3:1–58, 2000. 22, 27

[Barbalho *et al.*, 2013] Hugo Barbalho, Isabel Rosseti, Simone L Martins, and Alexandre Plastino. A hybrid data mining GRASP with path-relinking. *Computers & Operations Research*, 40(12):3159–3173, 2013. 26, 81

[Barr *et al.*, 1995] Richard S Barr, Bruce L Golden, James P Kelly, Mauricio GC Resende, and William R Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995. 27

[Barreto *et al.*, 2007] Sérgio Barreto, Carlos Ferreira, Jose Paixao, and Beatriz Sousa Santos. Using clustering analysis in a capacitated location-routing problem. *European Journal of Operational Research*, 179(3):968–977, 2007. 25

[Battiti *et al.*, 2008] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive Search and Intelligent Optimization*, volume 45. Springer Science & Business Media, 2008. 18, 24

[Bellio *et al.*, 2012] Ruggero Bellio, Luca Di Gaspero, and Andrea Schaerf. Design and statistical analysis of a hybrid local search algorithm for course timetabling. *Journal of Scheduling*, 15(1):49–61, 2012.

[Benlic and Hao, 2011] Una Benlic and Jin-Kao Hao. A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation*, 15(5):624–642, 2011. 56, 84, 90

[Benlic and Hao, 2012] Una Benlic and Jin-Kao Hao. A study of breakout local search for the minimum sum coloring problem. In *Proceedings of the 9th International Conference on Simulated Evolution and Learning*, SEAL'12, pages 128–137, Berlin, Heidelberg, 2012. Springer-Verlag. 11

[Benlic and Hao, 2013] Una Benlic and Jin-Kao Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013. 11, 84, 85, 87, 91, 92, 95

[Benlic and Hao, 2015] Una Benlic and Jin-Kao Hao. Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 42(1):584–595, 2015. 83, 85, 91, 92, 95, 96, 97

[Benlic *et al.*, 2017] Una Benlic, Michael G. Epitropakis, and Edmund K. Burke. A hybrid breakout local search and reinforcement learning approach to the vertex separator problem. *European Journal of Operational Research*, 261(3):803–818, 2017. 11, 28, 79

[Birattari, 2009] Mauro Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer Publishing Company, Incorporated, 1st ed. 2005. 2nd printing edition, 2009. 18

[Blöchliger and Zufferey, 2008] Ivo Blöchliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operation Research*, 35(3):960–975, 2008. 45, 46

[Blum and Roli, 2003] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003. 6, 10, 11

[Blum *et al.*, 2011] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011. 10, 11, 13

[Bouhmala and Granmo, 2008] Noureddine Bouhmala and Ole-Christoffer Granmo. Solving graph coloring problems using learning automata. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2008)*, pages 277–288, 2008. 40

[Boyan and Moore, 1998] Justin A. Boyan and Andrew W. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 3–10, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. 25

[Brélaz, 1979] Daniel Brélaz. New methods to color vertices of a graph. *Commumications of the ACM*, 22(4):251–256, 1979. 39, 43

[Brimberg *et al.*, 2009] Jack Brimberg, Nenad Mladenović, Dragan Urošević, and Eric Ngai. Variable neighborhood search for the heaviest k-subgraph. *Computers & Operations Research*, 36(11):2885–2891, 2009. 55, 67, 69, 72

[Burke *et al.*, 1994] EK Burke, DG Elliman, and R Weare. A university timetabling system based on graph colouring and constraint manipulation. *Journal of Research on Computing in Education*, 27(1):1–18, 1994. 7

[Burke *et al.*, 2002] Edmund K Burke, Bart L MacCarthy, Sanja Petrovic, and Rong Qu. Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 276–287. Springer, 2002. 29

[Burke *et al.*, 2010] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer, 2010. 18

[Burke *et al.*, 2013] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, Dec 2013. 24, 29

[Cadenas *et al.*, 2009] José Manuel Cadenas, M Carmen Garrido, and E Muñoz. Using machine learning in a cooperative hybrid parallel strategy of metaheuristics. *Information Sciences*, 179(19):3255–3267, 2009. 24, 25

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013. 38

[Calégari *et al.*, 1999] Patrice Calégari, Giovanni Coray, Alain Hertz, Daniel Kobler, and Pierre Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5(2):145–158, 1999. 13

[Calvet *et al.*, 2017] Laura Calvet, Jésica de Armas, David Masip, and Angel A Juan. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280, 2017. 22, 23

[Cao *et al.*, 2007] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136. ACM, 2007. 21

[Caramia and Dell'Olmo, 2008] Massimiliano Caramia and Paolo Dell'Olmo. Coloring graphs by iterated local search traversing feasible and infeasible solutions. *Discrete Applied Mathematics*, 156(2):201–217, 2008. 45

[Chaitin, 1982] Gregory J Chaitin. Register allocation & spilling via graph coloring. *ACM Sigplan Notices*, 17(6):98–101, 1982. 7

[Chen and Hao, 2016] Yuning Chen and Jin-Kao Hao. Memetic search for the generalized quadratic multiple knapsack problem. *IEEE Transactions on Evolutionary Computation*, 20(6):908–923, 2016. 56, 64

[Chen *et al.*, 2011] Xianshun Chen, Yew-Soon Ong, Men-Hiot Lim, and Key Chen Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011. 56

[Connolly, 1990] David T Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46(1):93–100, 1990. 85

[Corne *et al.*, 2012] David Corne, Clarisse Dhaenens, and Laetitia Jourdan. Synergies between operations research and data mining: The emerging use of multi-objective approaches. *European Journal of Operational Research*, 221(3):469 – 479, 2012. 22, 23

[Cowling *et al.*, 2001] Peter I. Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 176–190, London, UK, UK, 2001. Springer-Verlag. 29

[Cruz-Reyes *et al.*, 2012] Laura Cruz-Reyes, Claudia Gómez-Santillán, Joaquín Pérez-Ortega, Vanesa Landero, Marcela Quiroz, and Alberto Ochoa. Algorithm selection: From meta-learning to hyper-heuristics. In *Intelligent Systems*. InTech, 2012. 29

[Czapiński, 2013] Michał Czapiński. An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. *Journal of Parallel and Distributed Computing*, 73(11):1461–1468, 2013. 85

[Davis, 1991] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. 13

[de Freitas *et al.*, 2014] Alan R. R. de Freitas, Frederico Gadelha Guimarães, Rodrigo César Pedrosa Silva, and Marcone Jamilson Freitas Souza. Memetic self-adaptive evolution strategies applied to the maximum diversity problem. *Optimization Letters*, 8(2):705–714, 2014. 55, 56, 69

[de Lima Júnior *et al.*, 2007] Francisco Chagas de Lima Júnior, Jorge Dantas de Melo, and Adriao Duarte Doria Neto. Using Q-learning algorithm for initialization of the GRASP metaheuristic and genetic algorithm. In *2007 International Joint Conference on Neural Networks (IJCNN 2007)*, pages 1243–1248. IEEE, 2007. 25

[de Werra, 1985] Dominique de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985. 7

[Demirel and Toksarı, 2006] Nihan Çetin Demirel and M Duran Toksarı. Optimization of the quadratic assignment problem using an ant colony algorithm. *Applied Mathematics and Computation*, 183(1):427–435, 2006. 85

[Demšar, 2006] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. 67

[Di Gaspero *et al.*, 2013] Luca Di Gaspero, Andrea Rendl, and Tommaso Urli. A hybrid ACO + CP for balancing bicycle sharing systems. In *International Workshop on Hybrid Metaheuristics*, pages 198–212. Springer, 2013. 15

[Di Gaspero, 2015] Luca Di Gaspero. *Integration of metaheuristics and constraint programming*, pages 1225–1237. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. 15

[Dokeroglu and Cosar, 2016] Tansel Dokeroglu and Ahmet Cosar. A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 52:10–25, 2016. 85, 94

[Dorigo and Blum, 2005] Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005. 13

[Dorigo *et al.*, 1996] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41, February 1996. 13

[Dorne and Hao, 1998] Raphaël Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, PPSN V, pages 745–754. Springer, 1998. 41

[dos Santos *et al.*, 2014] João Paulo Queiroz dos Santos, Jorge Dantas de Melo, Adrião Dória Duarte Neto, and Daniel Aloise. Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search. *Expert Systems with Applications*, 41(10):4939–4949, 2014. 28

[Drezner *et al.*, 2005] Zvi Drezner, Peter M Hahn, and Éeric D Taillard. Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 139(1):65–94, 2005. 9, 85

[Duarte and Martí, 2007] Abraham Duarte and Rafael Martí. Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 178(1):71–84, 2007. 55

[Dueck and Scheuer, 1990] Gunter Dueck and Tobias Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161–175, 1990. 12

[Dueck, 1993] Gunter Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993. 12

[Duman and Or, 2007] Ekrem Duman and Ilhan Or. The quadratic assignment problem in the context of the printed circuit board assembly process. *Computers & Operations Research*, 34(1):163–179, 2007. 9, 10

[Elhag and Özcan, 2015] Anas Elhag and Ender Özcan. A grouping hyper-heuristic framework: Application on graph colouring. *Expert Systems with Applications*, 42(13):5491–5507, 2015. 33, 40

[Ergezer and Simon, 2011] Mehmet Ergezer and Dan Simon. Oppositional biogeography-based optimization for combinatorial problems. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1496–1503. IEEE, 2011. 25, 26, 56

[Erkut and Neuman, 1991] Erhan Erkut and Susan Neuman. Comparison of four models for dispersing facilities. *INFOR: Information Systems and Operational Research*, 29(2):68–86, 1991. 8

[Falkenauer, 1998] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Inc., 1998. 33

[Feige *et al.*, 2001] Uriel Feige, David Peleg, and Guy Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001. 55

[Feo and Resende, 1995] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. 12

[Fink, 1998] Eugene Fink. How to solve it automatically: Selection among problem solving methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, AIPS'98, pages 128–136, 1998. 28

[Fleurent and Ferland, 1996] Charles Fleurent and Jacques A Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461, 1996. 39, 41

[Focacci *et al.*, 2004] Filippo Focacci, Francois Laburthe, and Andrea Lodi. *Local search and constraint programming*, pages 293–329. Springer US, Boston, MA, 2004. 15

[Fu and Hao, 2015] Zhang-Hua Fu and Jin-Kao Hao. Dynamic programming driven memetic search for the steiner tree problem with revenues, budget, and hop constraints. *INFORMS Journal on Computing*, 27(2):221–237, 2015. 15

[Fürnkranz and Hüllermeier, 2010] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning: An introduction. In *Preference Learning*, pages 1–17. Springer, 2010. 21

[Galinier and Hao, 1999] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999. 33, 39, 41, 45, 48

[Galinier and Hertz, 2006] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operation Research*, 33:2547–2562, 2006. 39

[Galinier *et al.*, 2008] Philippe Galinier, Alain Hertz, and Nicolas Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2):267–279, 2008. 43, 45

[Galinier *et al.*, 2011] Philippe Galinier, Zied Boujbel, and Michael Coutinho Fernandes. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1):1–22, 2011. 56, 63, 64

[Galinier *et al.*, 2013] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel. Recent advances in graph vertex coloring. *Handbook of Optimization*, pages 505–528, 2013. 7, 39, 43

[Gallego *et al.*, 2009] Micael Gallego, Abraham Duarte, Manuel Laguna, and Rafael Martí. Hybrid heuristics for the maximum diversity problem. *Computational Optimization and Applications*, 44(3):411–426, 2009. 55, 67, 69, 73

[Gambardella *et al.*, 1999] Luca Maria Gambardella, É D Taillard, and Marco Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999. 85

[Gamst, 1986] Andreas Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, 35(1):8–14, 1986. 7

[García *et al.*, 2017] José García, Broderick Crawford, Ricardo Soto, Carlos Castro, and Fernando Paredes. A k-means binarization framework applied to multidimensional knapsack problem. *Applied Intelligence*, Jul 2017. 25

[Garey and Johnson, 1979] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 7, 33

[Gersmann and Hammer, 2005] Kai Gersmann and Barbara Hammer. Improving iterative repair strategies for scheduling with the SVM. *Neurocomputing*, 63:271–292, 2005. 25

[Ghosh, 1996] Jay B Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175–181, 1996. 8, 55

[Glover and Laguna, 1993] Fred Glover and Manuel Laguna. Modern heuristic techniques for combinatorial problems. chapter Tabu Search, pages 70–150. John Wiley & Sons, Inc., New York, NY, USA, 1993. 12, 13

[Glover and Laguna, 1997] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. 12, 60, 63

[Glover *et al.*, 1993] Fred Glover, Eric Taillard, and Dominique de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41(1-4):3–28, May 1993. 62

[Glover *et al.*, 1998] Fred Glover, Ching-Chung Kuo, and Krishna S Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132, 1998. 8, 55

[Glover *et al.*, 2004] Fred Glover, Manuel Laguna, and Rafael Martí. New ideas and applications of scatter search and path relinking. In *New Optimization Techniques in Engineering*, pages 367–383. Springer, 2004. 13

[Glover, 1986] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, May 1986. 10

[Glover, 1989] Fred Glover. Tabu searchpart I. *ORSA Journal on Computing*, 1(3):190–206, 1989. 13

[Glover, 1990] Fred Glover. Tabu searchpart II. *ORSA Journal on Computing*, 2(1):4–32, 1990. 12

[Glover, 1997] Fred Glover. A template for scatter search and path relinking. In *Selected Papers from the Third European Conference on Artificial Evolution*, AE '97, pages 1–51. Springer, 1997. 58

[Glover, 2014] Fred Glover. Exterior path relinking for zero-one optimization. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 5(3):1–8, 2014. 13

[Grahne and Zhu, 2003a] Gösta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI'03 Workshop on Frequent Itemset Mining Implementations: 2003*, volume 90, 2003. 88, 89

[Grahne and Zhu, 2003b] Gösta Grahne and Jianfei Zhu. High performance mining of maximal frequent itemsets. In *6th International Workshop on High Performance Data Mining*, 2003. 88

[Grahne and Zhu, 2005] Gösta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, 2005. 88, 89

[Guerine *et al.*, 2016] Marcos Guerine, Isabel Rosseti, and Alexandre Plastino. Extending the hybridization of metaheuristics with data mining: Dealing with sequences. *Intelligent Data Analysis*, 20(5):1133–1156, 2016. 26, 80, 81, 87

[Guo *et al.*, 2017] Peng Guo, Wenming Cheng, and Yi Wang. Hybrid evolutionary algorithm with extreme machine learning fitness function evaluation for two-stage capacitated facility location problems. *Expert Systems with Applications*, 71:57–68, 2017. 26

[Gusfield, 2002] Dan Gusfield. Partition-distance: A problem and class of perfect graphs arising in clustering. *Information Processing Letters*, 82(3):159–164, 2002. 64

[Hale, 1980] William K Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980. 7

[Han *et al.*, 2007] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007. 84

[Han *et al.*, 2011] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2011. 20, 79

[Hansen and Mladenović, 2005] Pierre Hansen and Nenad Mladenović. *Variable neighborhood search*, pages 211–238. Springer US, Boston, MA, 2005. 12

[Hao and Wu, 2012] Jin-Kao Hao and Qinghua Wu. Improving the extraction and expansion method for large graph coloring. *Discrete Applied Mathematics*, 160(16-17):2397–2407, 2012. 39

[Hao *et al.*, 1998] Jin-Kao Hao, Raphaël Dorne, and Philippe Galinier. Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics*, 4(1):47–62, 1998. 7

[Hao, 2012] Jin-Kao Hao. Memetic algorithms in discrete optimization. In *Handbook of Memetic Algorithms*, pages 73–94. Springer, 2012. 14, 56, 64

[Hertz and de Werra, 1987] Alain Hertz and Dominique de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987. 39, 41

[Hertz *et al.*, 2008] Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008. 45

[Holland, 1975] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975. 13

[Hoos and Stützle, 2004] Holger H Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier, 2004. 11

[Hoos, 2012] Holger H. Hoos. *Automated algorithm configuration and parameter tuning*, pages 37–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. 28

[Hutter *et al.*, 2002] Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, CP '02, pages 233–248, London, UK, UK, 2002. Springer-Verlag. 38

[Hutter *et al.*, 2009] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009. 18, 28

[Ishtaiwi *et al.*, 2005] Abdelraouf Ishtaiwi, John Thornton, Abdul Sattar, and Duc Nghia Pham. Neighbourhood clause weight redistribution in local search for SAT. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, CP'05, pages 772–776, Berlin, Heidelberg, 2005. Springer-Verlag. 38

[James *et al.*, 2009] T. James, C. Rego, Tabitha Fred, Glover James, CÉsar Rego, and Fred Glover. Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 39(3):579–596, May 2009. 85

[Jędrzejowicz and Ratajczak-Ropel, 2014] Piotr Jędrzejowicz and Ewa Ratajczak-Ropel. Reinforcement learning strategies for a-team solving the resource-constrained project scheduling problem. *Neurocomputing*, 146:301–307, 2014. 25

[Jin, 2011] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011. 66

[Jourdan *et al.*, 2006] Laetitia Jourdan, Clarisse Dhaenens, and El-Ghazali Talbi. *Using datamining techniques to help metaheuristics: A short survey*, pages 57–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 22

[Jourdan *et al.*, 2009] Laetitia Jourdan, Matthieu Basseur, and El-Ghazali Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009. 15

[Kanda *et al.*, 2016] Jorge Kanda, Andre de Carvalho, Eduardo Hruschka, Carlos Soares, and Pavel Brazdil. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. *Neurocomputing*, 205:393–406, 2016. 24, 29

[Kashan *et al.*, 2013] Ali Husseinzadeh Kashan, Mina Husseinzadeh Kashan, and Somayyeh Karimiyan. A particle swarm optimizer for grouping problems. *Information Sciences*, 252:81–95, 2013. 33

[Kirkpatrick *et al.*, 1983] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 12

[Koopmans and Beckmann, 1957] Tjalling C Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society*, pages 53–76, 1957. 9

[Kotthoff *et al.*, 2012] Lars Kotthoff, Ian P Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, 25(3):257–270, 2012. 28

[Kotthoff, 2014] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014. 18, 24, 28

[Kotthoff, 2016] Lars Kotthoff. *Algorithm selection for combinatorial search problems: A survey*, pages 149–190. Springer International Publishing, Cham, 2016. 28

[Krasnogor and Smith, 2005] Natalio Krasnogor and James Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005. 56

[Kuhn, 1955] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955. 43

[Kuo *et al.*, 1993] Ching-Chung Kuo, Fred Glover, and Krishna S Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1993. 8

[Laguna and Marti, 1999] Manuel Laguna and Rafael Marti. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999. 14

[Lai and Hao, 2016] Xiangjing Lai and Jin-Kao Hao. Iterated maxima search for the maximally diverse grouping problem. *European Journal of Operational Research*, 254(3):780–800, 2016. 11

[Leighton, 1979] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979. 7, 39

[Lemke *et al.*, 2015] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: A survey of trends and technologies. *Artificial Intelligence Review*, 44(1):117–130, June 2015. 18, 29

[Lessmann *et al.*, 2011] Stefan Lessmann, Marco Caserta, and Idel Montalvo Arango. Tuning metaheuristics: A data mining based approach for particle swarm optimization. *Expert Systems with Applications*, 38(10):12826–12838, 2011. 28

[Lewis and Paechter, 2007] Rhydian Lewis and Ben Paechter. Finding feasible timetables using group-based operators. *IEEE Transactions on Evolutionary Computation*, 11(3):397–413, 2007. 33

[Lewis, 2009] Rhyd Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & Operations Research*, 36(7):2295–2310, 2009. 33

[Leyton-Brown *et al.*, 2014] Kevin Leyton-Brown, Holger H Hoos, Frank Hutter, and Lin Xu. Understanding the empirical hardness of NP-complete problems. *Communications of the ACM*, 57(5):98–107, 2014.

[Loiola *et al.*, 2007] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007. 9, 10, 85

[Lourenço *et al.*, 2003] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003. 11, 87

[Lozano *et al.*, 2011] Manuel Lozano, Daniel Molina, and Carlos García-Martínez. Iterated greedy for the maximum diversity problem. *European Journal of Operational Research*, 214(1):31–38, 2011. 8, 55, 69, 72

[Lü and Hao, 2010] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010. 39, 44, 45, 47, 56, 64, 83, 85

[Macambira and De Souza, 2000] Elder Magalhães Macambira and Cid Carvalho De Souza. The edge-weighted clique problem: valid inequalities, facets and polyhedral computations. *European Journal of Operational Research*, 123(2):346–371, 2000. 55

[Macambira, 2002] Elder Magalhães Macambira. An application of tabu search heuristic for the maximum edge-weighted subgraph problem. *Annals of Operations Research*, 117(1-4):175–190, 2002. 55

[Mahmoudi and Lotfi, 2015] Shadi Mahmoudi and Shahriar Lotfi. Modified cuckoo optimization algorithm (MCOA) to solve graph coloring problem. *Applied Soft Computing*, 33:48–64, 2015. 40

[Malaguti and Toth, 2010] Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010. 39

[Malaguti *et al.*, 2008] Enrico Malaguti, Michele Monaci, and Paolo Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008. 39, 44, 45

[Malaguti *et al.*, 2011] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011. 39

[Maniezzo *et al.*, 2009] Vittorio Maniezzo, Thomas Sttzle, and Stefan Vo. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. Springer Publishing Company, Incorporated, 1st edition, 2009. 15

[Martí *et al.*, 2010] Rafael Martí, Micael Gallego, and Abraham Duarte. A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research*, 200(1):36–44, 2010. 55

[Martí *et al.*, 2013] Rafael Martí, Micael Gallego, Abraham Duarte, and Eduardo G Pardo. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 19(4):591–615, 2013. 8, 55, 67, 69, 70, 73

[Martí, 2003] Rafael Martí. Multi-start methods. *International Series in Operations Research and Management Science*, pages 355–368, 2003. 11

[Martins *et al.*, 2014] Daniel Martins, Gabriel M. Vianna, Isabel Rosseti, Simone L. Martins, and Alexandre Plastino. Making a state-of-the-art heuristic faster with data mining. *Annals of Operations Research*, pages 1–22, 2014. 27, 81

[Martins *et al.*, 2016] Simone de Lima Martins, Isabel Rosseti, and Alexandre Plastino. *Data mining in stochastic local search*, pages 1–49. Springer International Publishing, Cham, 2016. 22, 26, 27, 81

[Meisel and Mattfeld, 2010] Stephan Meisel and Dirk Mattfeld. Synergies of operations research and data mining. *European Journal of Operational Research*, 206(1):1–10, 2010. 22, 23

[Misevičius, 2003] Alfonsas Misevičius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 14(4):497–514, 2003. 85

[Misevicius, 2005] Alfonsas Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30(1):95–111, 2005. 85

[Mladenović and Hansen, 1997] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, November 1997. 12

[Moalic and Gondran, 2015] Laurent Moalic and Alexandre Gondran. The new memetic algorithm HEAD for graph coloring: An easy way for managing diversity. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2015)*, pages 173–183. Springer, 2015. 39, 44, 45

[Mohri *et al.*, 2012] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. 20, 21

[Moscato and others, 1989] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826:1989, 1989. 14

[Moscato, 1999] Pablo Moscato. Memetic algorithms: A short introduction. In *New Ideas in Optimization*, pages 219–234. McGraw-Hill Ltd., UK, 1999. 56

[Olafsson *et al.*, 2008] Sigurdur Olafsson, Xiaonan Li, and Shuning Wu. Operations research and data mining. *European Journal of Operational Research*, 187(3):1429–1448, 2008. 22, 23

[Osei-Bryson and Rayward-Smith, 2009] Kweku-Muata Osei-Bryson and Vic J Rayward-Smith. Data mining and operational research: techniques and applications. *Journal of the Operational Research Society*, 60(8):1043–1044, Aug 2009. 22, 23

[Palubeckis, 2007] Gintaras Palubeckis. Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, 189(1):371–383, 2007. 8, 55, 67, 69, 70, 72

[Pardalos *et al.*, 1994] Panos M. Pardalos, Franz Rendl, and Henry Wolkowicz. The quadratic assignment problem: A survey and recent developments. In *In Proceedings of the DIMACS Workshop on Quadratic Assignment Problems, volume 16 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–42. American Mathematical Society, 1994. 10

[Peng *et al.*, 1996] Tian Peng, Wang Huanchen, and Zhang Dongme. Simulated annealing for the quadratic assignment problem: A further study. *Computers & Industrial Engineering*, 31(3-4):925–928, 1996. 85

[Pinto *et al.*, 2015] Fábio Pinto, Carlos Soares, and Pavel Brazdil. Combining regression models and metaheuristics to optimize space allocation in the retail industry. *Intelligent Data Analysis*, 19(s1):S149–S162, 2015. 25

[Plastino *et al.*, 2011] Alexandre Plastino, Richard Fuchshuber, Simone de L. Martins, Alex A. Freitas, and Said Salhi. A hybrid data mining metaheuristic for the p-median problem. *Statistical Analysis and Data Mining*, 4(3):313–335, 2011. 26, 81

[Plastino *et al.*, 2014] Alexandre Plastino, Hugo Barbalho, Luis Filipe M. Santos, Richard Fuchshuber, and Simone L. Martins. Adaptive and multi-mining versions of the DM-GRASP hybrid metaheuristic. *Journal of Heuristics*, 20(1):39–74, 2014. 26, 81

[Porumbel *et al.*, 2010a] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research*, 37(10):1822–1832, 2010. 39, 44, 45

[Porumbel *et al.*, 2010b] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. A search space cartography for guiding graph coloring heuristics. *Computers & Operations Research*, 37(4):769–778, 2010. 41, 46

[Prokopyev *et al.*, 2009] Oleg A Prokopyev, Nan Kong, and Dayna L Martinez-Torres. The equitable dispersion problem. *European Journal of Operational Research*, 197(1):59–67, 2009. 55, 100

[Puchinger and Raidl, 2005] Jakob Puchinger and Günther R Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53. Springer, 2005. 15

[Quiroz-Castellanos *et al.*, 2015] Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gómez, Héctor J Fraire Huacuja, and Adriana CF Alvim. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64, 2015. 33

[Rahnamayan *et al.*, 2008a] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy Salama. Opposition-based differential evolution. *IEEE Transactions on Evolutionary Computation*, 12(1):64–79, 2008. 56

[Rahnamayan *et al.*, 2008b] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. Opposition versus randomness in soft computing techniques. *Applied Soft Computing*, 8(2):906–918, 2008. 56

[Rahnamayan *et al.*, 2012] Shahryar Rahnamayan, G Gary Wang, and Mario Ventresca. An intuitive distance-based explanation of opposition-based sampling. *Applied Soft Computing*, 12(9):2828–2839, 2012. 56

[Rangaswamy *et al.*, 1998] Balasubramanian Rangaswamy, Anant Singh Jain, and Fred Glover. Tabu search candidate list strategies in scheduling. In *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, pages 215–233. Springer, 1998. 62

[Raschip *et al.*, 2015a] Madalina Raschip, Cornelius Croitoru, and Kilian Stoffel. Guiding evolutionary search with association rules for solving weighted CSPs. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 481–488. ACM, 2015. 27, 81

[Raschip *et al.*, 2015b] Madalina Raschip, Cornelius Croitoru, and Kilian Stoffel. Using association rules to guide evolutionary search in solving constraint satisfaction. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 744–750. IEEE, 2015. 27, 81

[Rasheed and Hirsh, 2000] Khaled Rasheed and Haym Hirsh. Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 628–635. Morgan Kaufmann Publishers Inc., 2000. 26

[Ravi *et al.*, 1994] Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri K Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994. 55

[Reddy *et al.*, 2012] D Srinivas Reddy, A Govardhan, and Ssvn Sarma. Hybridization of neighbourhood search metaheuristic with data mining technique to solve p-median problem. *International Journal of Computational Engineering Research (IJCER)*, 2(7), 2012. 27, 81

[Resendel and Ribeiro, 2005] Mauricio GC Resendel and Celso C Ribeiro. GRASP with path-relinking: Recent advances and applications. In *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer, 2005. 14

[Ribeiro *et al.*, 2004] Marcos Henrique Ribeiro, Viviane de Aragao Trindade, Alexandre Plastino, and Simone L. Martins. Hybridization of GRASP metaheuristics with data mining techniques. In *Hybrid Metaheuristics, First International Workshop, HM 2004, Valencia, Spain, August 22-23, 2004, Proceedings*, pages 69–78, 2004. 26, 80, 81

[Ribeiro *et al.*, 2006] Marcos Henrique Ribeiro, Alexandre Plastino, and Simone L. Martins. Hybridization of GRASP metaheuristic with data mining techniques. *Journal of Mathematical Modelling and Algorithms*, 5(1):23–41, 2006. 26, 80, 81

[Rice, 1976] John R Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. 28

[Rojas-Morales *et al.*, 2016] Nicolas Rojas-Morales, R María-Cristina Riff, and U Elizabeth Montero. Learning from the opposite: Strategies for ants that solve multidimensional knapsack problem. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 193–200. IEEE, 2016. 25, 26

[Samorani and Laguna, 2012] Michele Samorani and Manuel Laguna. Data-mining-driven neighborhood search. *INFORMS Journal on Computing*, 24(2):210–227, 2012. 24, 25, 79

[Santos *et al.*, 2005] Luis Filipe M. Santos, Marcos Henrique Ribeiro, Alexandre Plastino, and Simone L. Martins. A hybrid GRASP with data mining for the maximum diversity problem. In *Hybrid Metaheuristics, Second International Workshop, HM 2005, Barcelona, Spain, August 29-30, 2005, Proceedings*, pages 116–127, 2005. 26, 55, 81

[Santos *et al.*, 2006a] Haroldo G Santos, Luiz Satoru Ochi, Euler Horta Marinho, and Lúcia Maria de A Drummond. Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem. *Neurocomputing*, 70(1):70–77, 2006. 27, 81

[Santos *et al.*, 2006b] R Santos, L. F.and Milagres, C.V. Albuquerque, S. Martins, and Alexandre Plastino. NGL01-4: A hybrid GRASP with data mining for efficient server replication for reliable multicast. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–6. IEEE, 2006. 26, 81

[Santos *et al.*, 2008] Luis F. Santos, Simone L. Martins, and Alexandre Plastino. Applications of the DM-GRASP heuristic: A survey. *International Transactions in Operational Research*, 15(4):387–416, 2008. 22, 24, 26, 81

[Schuurmans *et al.*, 2001] Dale Schuurmans, Finnegan Southey, and Robert C Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 334–341, 2001. 38

[Sghir *et al.*, 2015] Ines Sghir, Jin-Kao Hao, Ines Ben Jaafar, and Khaled Ghédira. A multi-agent based optimization method applied to the quadratic assignment problem. *Expert Systems with Applications*, 42(23):9252–9262, 2015. 40

[Shaw, 1998] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998. 15

[Skorin-Kapov, 1990] Jadranka Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990. 85

[Skorin-Kapov, 1994] Jadranka Skorin-Kapov. Extensions of a tabu search adaptation to the quadratic assignment problem. *Computers & Operations Research*, 21(8):855–865, 1994. 85

[Sörensen and Glover, 2013] Kenneth Sörensen and Fred W. Glover. *Metaheuristics*, pages 960–970. Springer US, Boston, MA, 2013. 10, 11, 12

[Sörensen and Sevaux, 2006] Kenneth Sörensen and Marc Sevaux. MA|PM: memetic algorithms with population management. *Computers & Operations Research*, 33(5):1214–1225, 2006. 64, 85

[Stützle and Dorigo, 1999] Thomas Stützle and Marco Dorigo. ACO algorithms for the quadratic assignment problem. *New Ideas in Optimization*, (C50):33, 1999. 85

[Stützle, 2006] Thomas Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, 2006. 85

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An introduction*, volume 1. MIT press Cambridge, 1998. 37, 101

[Taillard, 1991] Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991. 85, 86

[Talbi, 2009] El-Ghazali Talbi. *Metaheuristics: from Design to Implementation*, volume 74. John Wiley & Sons, 2009. 27

[Talbi, 2016] El-Ghazali Talbi. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research*, 240(1):171–215, 2016. 13, 15, 22, 23

[Tan and others, 2006] Pang-Ning Tan et al. *Introduction to Data Mining*. Pearson Education India, 2006. 21

[Telelis and Stamatopoulos, 2001] Orestis Telelis and Panagiotis Stamatopoulos. Combinatorial optimization through statistical instance-based learning. In *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on*, pages 203–209. IEEE, 2001. 25

[Thierens, 2004] Dirk Thierens. Population-based iterated local search: Restricting neighborhood search by crossover. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 234–245. Springer, 2004. 14

[Thornton *et al.*, 2004] John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of the 19th National Conference on Artifical Intelligence*, volume 4 of *AAAI'04*, pages 191–196, 2004. 38

[Titiloye and Crispin, 2011] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011. 39, 44, 45

[Titiloye and Crispin, 2012] Olawale Titiloye and Alan Crispin. Parameter tuning patterns for random graph coloring with quantum annealing. *PloS one*, 7(11):e50060, 2012. 44, 45

[Tizhoosh, 2005] Hamid R. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) - Volume 01*, CIMCA '05, pages 695–701, Washington, DC, USA, 2005. IEEE Computer Society. 56

[Toffolo *et al.*, 2018] Túlio A.M. Toffolo, Jan Christiaens, Sam Van Malderen, Tony Wauters, and Greet Vanden Berghe. Stochastic local search with learning automaton for the swap-body vehicle routing problem. *Computers & Operations Research*, 89:68–81, 2018. 25

[Tosun, 2015] Umut Tosun. On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 39:267–278, 2015. 85, 92, 93

[Tsutsui and Fujimoto, 2009] Shigeyoshi Tsutsui and Noriyuki Fujimoto. Solving quadratic assignment problems by genetic algorithms with GPU computation: A case study. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2523–2530, New York, NY, USA, 2009. ACM. 85

[Umetani, 2015] Shunji Umetani. Exploiting variable associations to configure efficient local search in large-scale set partitioning problems. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 1226–1232. AAAI Press, 2015. 27, 81

[Umetani, 2017] Shunji Umetani. Exploiting variable associations to configure efficient local search algorithms in large-scale binary integer programs. *European Journal of Operational Research*, 263(1):72–81, 2017. 27

[Ventresca and Tizhoosh, 2008] Mario Ventresca and Hamid R Tizhoosh. A diversity maintaining population-based incremental learning algorithm. *Information Sciences*, 178(21):4038–4056, 2008. 25, 26, 56

[Voß *et al.*, 2012] Stefan Voß, Silvano Martello, Ibrahim H Osman, and Catherine Roucairol. *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Springer Science & Business Media, 2012. 11

[Voudouris and Tsang, 1999] Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, 1999. 12

[Wang *et al.*, 2012] Jiahai Wang, Ying Zhou, Yiqiao Cai, and Jian Yin. Learnable tabu search guided by estimation of distribution for maximum diversity problems. *Soft Computing*, 16(4):711–728, 2012. 25, 55, 67, 68, 69, 70, 72

[Wang *et al.*, 2014] Yang Wang, Jin-Kao Hao, Fred Glover, and Zhipeng Lü. A tabu search based memetic algorithm for the maximum diversity problem. *Engineering Applications of Artificial Intelligence*, 27:103–114, 2014. 55, 69

[Wauters *et al.*, 2013] Tony Wauters, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. *Boosting metaheuristic search using reinforcement learning*, pages 433–452. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. 79

[Wauters *et al.*, 2015] Tony Wauters, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. A learning-based optimization approach to multi-project scheduling. *Journal of Scheduling*, 18(1):61–74, 2015. 79

[Wilhelm and Ward, 1987] Mickey R Wilhelm and Thomas L Ward. Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, 19(1):107–119, 1987. 85

[Witten *et al.*, 2016] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016. 19, 20

[Woeginger, 2003] Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. *Lecture Notes in Computer Science*, 2570(2003):185–207, 2003. 10

[Wu and Hao, 2013] Qinghua Wu and Jin-Kao Hao. A hybrid metaheuristic method for the maximum diversity problem. *European Journal of Operational Research*, 231(2):452–464, 2013. 39, 55, 56, 60, 62, 63, 64, 67, 68, 69, 70, 71, 72, 73

[Xu and Wunsch, 2005] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005. 21

[Xu *et al.*, 2014] Qingzheng Xu, Lei Wang, Na Wang, Xinhong Hei, and Li Zhao. A review of opposition-based learning from 2005 to 2012. *Engineering Applications of Artificial Intelligence*, 29:1–12, 2014. 56

[Zhang, 2010] Qingfu Zhang. Reactive search and intelligent optimization (Battiti, R., et al.; 2008) [book review]. *IEEE Computational Intelligence Magazine*, 5(1):59–60, Feb 2010. 28

[Zhou and Hao, 2017a] Yangming Zhou and Jin-Kao Hao. A fast heuristic algorithm for the critical node problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 121–122, New York, NY, USA, 2017. ACM. 3, 100

[Zhou and Hao, 2017b] Yangming Zhou and Jin-Kao Hao. An iterated local search algorithm for the minimum differential dispersion problem. *Knowledge-Based Systems*, 125:26–38, 2017. 3, 11, 12, 79, 85, 90

[Zhou *et al.*, 2014] Zhaoyang Zhou, Chu-Min Li, Chong Huang, and Ruchu Xu. An exact algorithm with learning for the graph coloring problem. *Computers & Operations Research*, 51:282–301, 2014. 39

[Zhou *et al.*, 2016] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications*, 64:412–422, 2016. 2, 31, 79

[Zhou *et al.*, 2017a] Yangming Zhou, Béatrice Duval, and Jin-Kao Hao. Improving probability learning based local search for graph coloring. *Submitted to Applied Soft Computing, revised in September 2017*, 2017. 2, 31

[Zhou *et al.*, 2017b] Yangming Zhou, Béatrice Duval, and Jin-Kao Hao. Recherche locale avec apprentissage par reinforcement pour le problème de coloration de graphe. In *Proceedings of the 18ème congrès de la société Française de Recherche Opérationnelle et dAide à la Décision (ROADEF 2017)*, 2017. 31

[Zhou *et al.*, 2017c] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. Opposition-based memetic search for the maximum diversity problem. *IEEE Transactions on Evolutionary Computation*, 21(5):731–745, Oct 2017. 2, 13, 14, 53, 83, 84, 90

[Zhou *et al.*, 2017d] Yangming Zhou, Jin-Kao Hao, and Béatrice Duval. When data mining meets optimiation: A case study on the quadratic assignment problem. *arXiv preprint arXiv:1708.05214*, 2017. 2, 77

[Zhou *et al.*, 2017e] Yangming Zhou, Jin-Kao Hao, and Fred Glover. Memetic search for identifying critical nodes in sparse graphs. *arXiv preprint arXiv:1705.04119*, 2017. 3, 14, 83, 84, 100

[Zufferey *et al.*, 2008] Nicolas Zufferey, Patrick Amstutz, and Philippe Giaccari. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11(4):263–277, 2008. 7

UNIVERSITE BRETAGNE LOIRE

université angers

Thèse de Doctorat

Yangming ZHOU

Approches de Résolution Renforcées par des Méthodes d'Apprentissage en Optimisation Combinatoire

Learning-Driven Optimization Approaches for Combinatorial Search Problems

Résumé

Cette thèse vise à développer des approches de résolution heuristique renforcées par des méthodes dapprentissage pour résoudre des problèmes doptimisation combinatoire difficiles (COPs). Nous considèrons notamment trois types importants de COPs, les problèmes de groupement comme la coloration de graphe (GCP), les problèmes de sélection de sous-ensembles comme la diversité maximum (MDP) et les problèmes de permutation comme lassignation quadratique (QAP). Ces trois classes de problèmes ont de nombreuses applications pratiques, mais ils sont dans le cas général $\mathcal{NP}$-difficiles. Cette thèse sattache à proposer des méthodes heuristiques renforcées par des méthodes dapprentissage. Les méthodes dapprentissage permettent dexploiter les traces des explorations déjà effectuées afin de découvrir des régions prometteuses et des motifs intéressants conduisant à des meilleures solutions. Nous proposons trois approches de résolution combinées à des stratégies dapprentissage adaptées pour les trois classes de COPs considérés. Nous développons une recherche locale combinée à un apprentissage de probabilités pour les problèmes de goupement comme GCP, une recherche mémétique avec apprentissage par opposition pour MDP et une recherche exploitant des motifs fréquents pour QAP. Toutes les approches proposées ont été validées expérimentalement sur des instances benchmark, et les résultats obtenus montrent quelles sont compétitives par rapport aux méthodes de l'état de l'art.

Mots clés

Apprentissage automatique, Métaheuristique, Optimisation axée sur l'apprentissage, Problème de coloration de graphe, Problème de diversité maximum, Problème d'assignation quadratique.

Abstract

This thesis is devoted to developing learning-driven optimization approaches for solving hard Combinatorial Optimization Problems (COPs). We particularly consider three important categories of COPs, including grouping problems such as Graph Coloring Problem (GCP), subset selection problems such as Maximum Diversity Problem (MDP), and permutation problems such as Quadratic Assignment Problem (QAP). These three classes of problems are of important theoretical significance, and have a wide range of practical applications. Given that they usually belong to the $\mathcal{NP}$-hard problems, it is computationally difficult to solve them in the general case. This thesis concentrates on designing learning-driven heuristic optimization approaches for solving these problems. With the help of machine learning techniques, heuristic approaches will be able to benefit from their past search history such as discovering promising regions and useful patterns to find better solutions. In this thesis, we propose three learning-driven heuristic approaches for the three categories of considered COPs. We develop a probability learning based local search for grouping problems, especially for GCP; an opposition-based memetic search for MDP; and a frequent pattern based search for QAP. All the proposed approaches were experimentally assessed based on benchmarks, and experimental results show that they compete favorably with state-of-the-art methods. Also, the beneficial effects of the introduced learning techniques are confirmed by experimental evidences.

Key Words

Machine learning, Meta-heuristics, Learning-based optimization, Graph coloring problem, Maximum diversity problem, Quadratic assignment problem.