The maximum clique problems with applications to graph coloring

Qinghua Wu

L'UNAM l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

université angers

Thèse de Doctorat

Qinghua WU

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Informatique
Laboratoire : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)

Soutenue le fevrier 2013

École doctorale : 503 (STIM)
Thèse n° : 1307

The maximum clique problems with applications to graph coloring

Problèmes de clique maximum avec applications à la coloration de graphe

JURY

Rapporteurs :	**M. Philippe Galinier**, Professeur, l'École Polytechnique de Montréal
	M^{me} Christine Solnon, Professor, l'INSA de Lyon
Examinateurs :	**M. Claude Jard**, Professeur, l'Université de Nantes
	M. Zhipeng Lu, Professeur, l'Université de Science et de Technologie Huazhong
Directeur de thèse :	**M. Jin-Kao Hao**, Professeur, Université d'Angers

Acknowledgments

First of all, I would like to thank Professor Jin-Kao Hao, who has been a great supervisor and mentor throughout my three years at the University of Angers. His inspiring enthusiasm and energy have been contagious, and his advice and constant support have been extremely helpful. I am also thankful for the excellent example he has set for me as a successful professor with very nice personality.

I am also extremely grateful to Professor Zhipeng Lu, who has given me invaluable helps during these years. His advice and patience helped me a lot in the early stage of my Ph.D. His guidance has served me well and I owe him my heartfelt appreciation.

I am very grateful to Dr. Philippe Galinier of l'École Polytechnique de Montréal and Dr. Christine Solnon of l'INSA de Lyon, who contributed their efforts to advancing this thesis and writing thoughtful review reports. I'd like also to thank Dr. Claude Jard of Université de Nantes, the president in my jury, for handwriting the defense report accompanied with other reviewers and providing me a recommendation letter.

I would also like to acknowledge the financial support of the four-year scholarship provided by the China Scholarship Council. I also would like to show my gratitude to the LERIA, University of Angers. I also would like to acknowledge the whole technical team of LIRIA to guarantee the stable experimental environment. I am also thankful to the secretaries Christine Bardaine and Catherine Pawlonski for kindly providing me many helps.

I am thankful to my colleges at LERIA. They are: Nadarajen Veerapen, Wang Yang, Zhanghua Fu, Una Benlic, Rongqiang Zeng, Xiujun Zhang, Sami Laroum, Keqin Liu, Chhel Fabien, Yan Jin and Yuning Cheng. Their friendship and assistance has meant more to me than I could ever express. I am also thankful to my friends Jing Lu and LinLin Gao. I am very happy that, in many cases, my friendships with you have extended well beyond our shared time in Angers.

I wish to thank my parents, my brother and my sister. Their love provided my inspiration and was my driving force. I owe them everything and wish I could show them just how much I love and appreciate them. Finally, I would like to dedicate this work to my lost relatives including my Paternal Grandfather, my Paternal Grandmother, my Maternal Grandfather and my Maternal Grandmother, who left us too soon. I hope that this work makes you proud.

Dedication

Contents

Contents

General Introduction

Context

Optimization has been expanding in all directions at an astonishing rate during the last few decades and nowadays it is a basic research tool which has been successfully applied in a wide range of practical problems arising in virtually any sphere of human activities, including biomedicine, energy management, aerospace research, telecommunications, transportation and finance.

Heuristic and metaheuristic can be considered as a special case of a broad optimization area called combinatorial optimization, in which the feasible region is a finite, but usually very large, set. All of the problems studied in this work are combinatorial optimization problems, which arise in diverse areas of research and development. However, combinatorial optimization problems are known to be difficult to solve in general. Most of them, in particular those of practical interest, belong to the class of NP-hard problems, and thus cannot be efficiently solved to optimality. For these hard large-scale optimization problems, a variety of optimization approaches, called heuristics, have been proposed for finding high quality sub-optimal solutions in reasonable computing time, thus constituting a natural and useful framework to approximate hard combinatorial optimization problems.

The maximum clique problem is an important combinatorial optimization problem with applications in numerous fields, including information retrieval, signal transmission analysis, classification theory, economics, scheduling, and biomedical engineering. In addition to its practical applications in real-world areas, the maximum clique problem is tightly related to a number of important combinatorial problems such as graph coloring, sum coloring, set packing, clique partitioning and community detection in complex networks. This thesis is devoted to developing effective heuristic approaches for the maximum clique problem and its generalizations, and applying the developed heuristic approaches to solve these hard combinatorial problems which are tightly related to the maximum clique problem.

Objectives

Given the importance of the maximum clique problem and its wide range of applications in numerous fields, the first objective of this thesis is to propose highly effective heuristic approaches for the maximum clique problem as well as its two generalizations (i.e., the maximum vertex weight clique problem and the maximum edge weight clique problem). As a result, we developed an adaptive multistart tabu search algorithm for the maximum clique problem, a multi-neighborhood tabu search algorithm for the maximum vertex weight clique problem and a hybrid metaheuristic approach for the maximum edge weight clique problem. These methods are highly competitive when compared with the current state-of-art algorithms from the literature, some of them are even able to improve the best-known solutions for a number of popular benchmark instances.

In addition to its practical applications in numerous fields, the maximum clique problem is tightly related to a number of important well-known problems such as the set packing problem, the graph coloring problem and the minimum sum coloring problem. For instance, for the graph coloring problem, large independent sets are generally preferred in the final legal coloring solutions and effective approaches are needed to identify large independent sets from the graph. The second objective of this thesis is to apply the developed heuristic approaches for the MCP to solving these hard related problems. As a result, we successfully applied the multi-neighborhood tabu search for the MVWCP to the sect packing problem, and the adaptive multistart tabu search for the MCP to the graph coloring problem and the minimum sum coloring problem. Remarkable results are achieved for these 3 classical problems.

Contributions

The main contribution of this thesis are given below:

1. An adaptive multistart tabu search approach for the maximum clique problem;
2. A multi-neighborhood tabu search approach for the maximum vertex weight clique problem;
3. A hybrid metaheuristic approach for the maximum edge weight clique problem;
4. A heuristic approach based on independent set extraction for coloring large graphs;
5. An improved extraction and expansion approach for graph coloring;
6. An effective heuristic algorithm based on independent set extraction for sum coloring of graphs.

An adaptive multistart tabu search algorithm: For the maximum clique problem, we proposed an adaptive multistart tabu search algorithm (denoted by AMTS), which integrates some distinguishing features such as a constrained neighborhood, a dynamic tabu tenure mechanism and a long term memory based restart strategy. The proposed algorithm is evaluated on the whole set of 80 DIMACS challenge benchmarks and compared with five state-of-the art algorithms. Computational results show that the proposed algorithm attains the largest known clique for 79 benchmarks. A paper describing the proposed algorithm is accepted in Journal of Combinatorial Optimization [Wu and Hao, 2011a].

A multi-neighborhood tabu search algorithm: For the maximum vertex weight clique problem, we proposed a multi-neighborhood tabu search algorithm. In order to effectively explore the search space, the proposed multi-neighborhood tabu search combines three neighborhoods induced by three types of moves. The particularity of the combined neighborhood relies on the union of the underlying neighborhoods instead of the conventional sequential exploration of basic neighborhoods. At each iteration of the algorithm, the proposed tabu search approach explores the union of these three neighborhoods and selects the overall best admissible neighboring solution. The proposed algorithm is evaluated on a total of 136 benchmark instances from different sources (DIMACS, BHOSLIB and set packing). Computational results disclose that the proposed new tabu search algorithm outperforms the leading algorithm for the maximum vertex weight clique problem, and in addition rivals the performance of the best methods for the unweighted version of the problem without being specialized to exploit this problem class. A paper describing this work [Wu and Hao, 2012f] is published in Annals of Operations Research.

A hybrid metaheuristic approach: For the maximum edge weight clique problem, we proposed a hybrid metaheuristic approach. The proposed algorithm uses a dedicated crossover operator to generate new solutions and a constrained neighborhood tabu search procedure for local optimization. The proposed algorithm applies also a distance-and-quality based replacement strategy to maintain population diversity. Extensive evaluations on a large set of 120 benchmark instances show that the proposed approach competes very favorably with the current state-of-art methods for the maximum edge weight clique problem. In particular, it consistently and easily attains all the best known results. Moreover, it yields improved results for 6 large random instances. The key components of proposed hybrid metaheuristic approach are also analyzed to shed light on their functioning of the algorithm. A paper describing the proposed hybrid metaheuristic algorithm is submitted to European Journal of Operational Research [Wu and Hao, 2012c].

Application of the AMTS algorithm for MCP to the graph coloring problem: We applied the proposed AMTS for MCP to the graph coloring problem, and presented an effective heuristic approach based on independent set extraction (denoted by EXTRACOL) to coloring large graphs. The proposed approach uses a preprocessing method to extract large independent sets from the graph and a memetic algorithm to color the residual graph. At each preproceesing step, we identify with AMTS a number of pairwise disjoint independent sets of a given size in order to maximize the vertices removed from the graph. EXTRACOL is evaluated on the 11 largest graphs of the DIMACS challenge benchmarks and it is able to improve the current best known results for 4 instances. The advantage and the limitation of the proposed algorithm is also analyzed in order to shed light on ways to further improvement. The results of this study are published in Computers & Operations Research [Wu and Hao, 2012a]. To overcome the limitations of the proposed EXTRACOL algorithm, a further study of our work is presented in [Hao and Wu, 2012], in which we proposed an extraction and expansion approach (denoted by IE^{2}COL) to improve EXTRACOL. The basic idea of IE^{2}COL is to reconsider the extracted independent sets and allow some vertices of these extracted sets to change their colors. Excellent results are achieved by IE^{2}COL, it is able to improve the current best known results for 4 instances and matches consistently the current best-known results for the other graphs. This further study on the graph coloring problem is published in Discrete Applied Mathematics.

Application of the AMTS algorithm for the MCP to the minimum sum coloring problem: We applied the same idea of the proposed EXTRACOL algorithm for graph coloring to the minimum sum coloring problem, and proposed an effective heuristic algorithm based on independent set extraction (denoted by EXSCOL) for sum coloring of graphs. Basically, the proposed EXSCOL algorithm iteratively extracts from the graph as many large disjoint independent sets of equal size as possible. For each extracted independent set, we assign to it the smallest available color (colors are represented by natural numbers 1, 2...). This process is repeated until the graph becomes empty. The rationale behind this approach is that by extracting many large disjoint independent sets, we naturally favor the construction of large color classes and reduce the number of needed color classes, leading to a reduced total sum of colors. Experimental evaluations on a collection of 56 DIMACS and COLOR2 benchmark graphs show that the proposed approach achieves highly competitive results. For nearly half of the graphs used in the literature, our approach improves the current best known upper bounds. The study is reported in a full paper published in Computers & Operations Research.

Organization

The manuscript is organized in the following way:

- In the first chapter, we first formally introduce the definitions of the maximum clique and related problems. Then we provide an overview of the most representative exact and heuristic approaches proposed in the literature for the maximum clique problem. Moreover, we recall some practical applications of the MCP in different fields.

- In the second chapter, we present the first contribution of this thesis, i.e., an adaptive multistart tabu search algorithm for the maximum clique problem. We also perform extensive computational studies and comparisons with the best performing approaches from the literature to evaluate the performance of the proposed algorithm, using the whole set of DIMACS benchmark instances.

- In the third chapter, we consider the maximum vertex weight clique problem and present a multi-neighborhood tabu search algorithm. After a short introduction of the maximum vertex weight clique problem, we present in detail the components of the proposed multi-neighborhood tabu search algorithm. Finally, we evaluate the performance of the proposed algorithm on the whole sets of DIMACS-W and BOSHLIB-W benchmark instances.

- In the fourth chapter, we consider the maximum edge weight clique problem which is another generalization of the maximum clique problem and present a memetic algorithm for the maximum edge weight clique problem. The proposed memetic algorithm integrates a dedicated crossover operator which tries to preserve common vertices that are shared by parent solutions with a constrained neighborhood tabu search procedure for local optimization. In the end of this chapter, we show experimental results obtained by the proposed memetic algorithm and compare these results with those obtained by the current best-performing algorithms for the maximum edge weight clique problem from the literature.

- In the fifth chapter, we apply the AMTS algorithm proposed in Chapter 2 for the maximum clique problem to deal with the graph coloring problem, and present the heuristic approach based on independent set extraction (EXTRACOL) to coloring large graphs. After a short overview of current state-of-art graph coloring approaches, we detail the components of the proposed EXTRACOL algorithm. Then, we present computational results and comparisons with the current best performing approaches for the graph coloring problem on 11 largest DIMACS benchmark instances. We analyse the limitation of the proposed EXTRACOL algorithm for the purpose of further improvement.

- Chapter 6 presents the extraction and expansion approach to further improve the EXTRACOL algorithm by proposing additional strategies to remedy the limitation of EXTRACOL. The proposed algorithm integrates an expansion and backward coloring phase which allows the subsequent coloring algorithm to reconsider the extracted independent sets and allows some vertices of these extracted sets to change their colors. Extensive computational experiments are carried out to show the interest of the enhancements implemented in the proposed algorithm.

- In the last chapter, we apply the same idea of the EXTRACOL algorithm for graph coloring to the minimum sum coloring problem, and present the heuristic algorithm based on independent set extraction (EXSCOL) for sum coloring of graphs. The chapter begins with a short introduction of the minimum sum coloring problem. We then present the basic idea and the main components of the proposed algorithm. Finally, we present computational results and comparisons with the current best performing approaches for the minimum sum coloring problem on the sets of DIMACS and COLOR02 benchmark instances

Chapter 1

Introduction

The maximum clique problem (MCP) is an important combinatorial optimization problem with a wide range of practical applications in numerous fields, including information retrieval, signal transmission analysis, classification theory, economics, scheduling, and biomedical engineering. Moreover, a number of combinatorial optimization problems are tightly related to the MCP, such as graph coloring, sum coloring, set packing and optimal winner determination. In this chapter, we first formally introduce the maximum clique problem and some related problems. Then we provide a brief overview of some popular exact and heuristic approaches proposed in the literature for the MCP. Finally, we give some examples of practical application of the MCP.

Contents

1.1 The maximum clique and related problems

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \dots, n\}$ and edge set $E \subset V \times V$. A clique C of G is a subset of V such that every two vertices in C are adjacent, i.e., $\forall u, v \in C, \{u, v\} \in E$. A clique is maximal if it is not contained in any other clique, a clique is maximum if its cardinality is the largest among all the cliques of the graph. The maximum clique problem (MCP) is to determine a maximum clique.

The maximum clique problem is equivalent to two other well-known combinatorial optimization problems: maximum independent set problem (MIS) and minimum vertex cover problem (MVC). Given an undirected graph $G = (V, E)$, an independent set I of G is a subset of V such that every two vertices in I are not connected by an edge, i.e., $\forall u, v \in I, \{u, v\} \notin E$. The maximum independent set problem is to find an independent set of maximum cardinality. A vertex cover V' of G is a subset of V, such that every edge $(i, j) \in E$ has at least one endpoint in V'. The minimum vertex cover problem is to find a vertex cover of minimum cardinality.

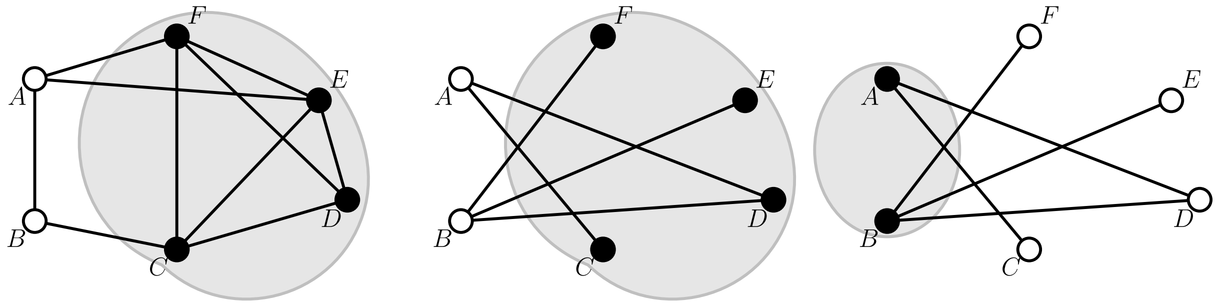


Figure 1.1: An illustration of the relationship between maximum clique (left), maximum independent set (middle) and minimum vertex cover (right).

It is easy to see that C is a maximum clique of G if and only if I is a maximum independent set of $\overline{G}$, where $\overline{G}$ is the complementary graph of G, and if and only if $V \setminus C$ is a minimum vertex cover of $\overline{G}$. An illustration of the relationship between maximum clique , maximum independent set and minimum vertex cover is given in Fig. 1.1. Due to the close relation between the maximum clique and maximum independent set problems, we will operate with both problems while describing the properties and algorithms for the maximum clique problem. In this case, it is clear that a result holding for the maximum clique problem in G will also be true for the maximum independent set problem in $\overline{G}$.

An important generalization of the MCP which is receiving increasing attention arises when positive weights are associated to the vertices of the graph. In this case, the problem is known as the maximum vertex weight clique problem (MVWCP). Let $w : V \rightarrow Z^+$ be a weighting function that assigns to each vertex $i \in V$ a positive value. For a clique C of G, define its weight as $W(C) = \sum_{i \in C} w_i$. The MVWCP is to determine a clique C^* of maximum weight, i.e., $\forall C \in \Omega, W(C^*) \geq W(C)$ where Ω is the set of all possible cliques of the graph. Note that the maximum weight clique is not necessarily a clique of the highest cardinality. However, the classical unweighted version can be viewed as a special case of

MVWCP when the weight of each vertex is set equal to 1.

Another important generalization of the MCP is the maximum edge weight clique problem (MEWCP) [Alidaee *et al.*, 2007] and can be formally defined as follows: Given an integer m and a complete graph $G = (V, E)$, each edge $\{i, j\} \in E$ being associated with a positive weight d_{ij}. The MEWCP is to determine a clique of G with exactly m vertices such that the sum of the weights of the edges in the clique is maximized. During the past three decades, the MEWCP has appeared in previous studies under many different names such as maximum diversity, maxisum dispersion, MAX-AVG dispersion, remote-clique, maximum edge-weighted subgraph, and dense k-subgraph.

The graph coloring problem, which is one of the most studied combinatorial optimization problems, is tightly related to the maximum independent set problem. Given an undirected graph $G = (V, E)$, a legal k-coloring of G is a partition of V into k independent sets (color classes). The graph k-coloring problem is to find a legal k-coloring of G for a given k. The graph coloring problem is to determine the smallest integer k (its chromatic number $\chi(G)$) such that there exists a legal k-coloring of G.

1.2 Complexity Results

The maximum clique and the graph coloring problems are NP-hard [Garey and Johnson, 1979]. Moreover, they are associated with a series of recent results about hardness of approximations. The discovery of a remarkable connection between probabilistically checkable proofs and approximability of combinatorial optimization problems [Arora and Safra, 1992; Arora *et al.*, 1998; Feige *et al.*, 1996] yielded new hardness of approximation results for many problems. Arora and Safra [Arora *et al.*, 1998] proved that for some positive ϵ, the approximation of the maximum clique within a factor of n^{ϵ} is NP-hard. In [Hästad, 1999], Hästad has shown that in fact for any $\delta > 0$, the maximum clique is hard to approximate in polynomial time within a factor $n^{1-\delta}$.

Similar approximation complexity results hold for the graph coloring problem as well. Garey and Johnson [Garey and Johnson, 1976] have shown that obtaining colorings using $s * \chi(G)$ colors, where $s < 2$, is NP-hard. It has been shown by Lund and Yannakakis [Lund and Yannakakis, 1994] that $\chi(G)$ is hard to approximate within n^{ϵ} for positive ϵ, and following Hästad [Hästad, 1999], Feige and Kilian [Feige and Kilian, 1998] have shown that for any $\delta > 0$, the chromatic number is hard to approximate within a factor of $n^{1-\delta}$, unless $NP \subseteq ZPP$.

All of the above facts together with practical evidence [Johnson and Trick, 1996] suggest that the maximum clique and coloring problems are hard to solve even in graphs of moderate sizes.

1.3 Exact Algorithms for MCP

Due to the interest and importance of the MCP, considerable efforts have been devoted to developing both exact and heuristic algorithms for tackling the maximum clique problem. Exact approaches (e.g., branch and bound, Lagrangian relaxation, column generation, cutting planes, etc) are algorithms which guarantee at finding an optimal solution, but with a runtime that often grows forbiddingly large with instance size in case of hard problem instances. Still, highly effective exact algorithms have been proposed in the literature for tackling the maximum clique problem. With the increased speed of modern computers, some of these algorithms are even able to solve large instances with up to 1500 vertices and 568960 edges. In the following, we provide a brief overview of some of the best and most important exact algorithms for the MCP.

There are many exact algorithms for the maximum clique and related problems available in the literature. Most of them are variations of the branch and bound method, which can be defined by different techniques for determining lower and upper bounds and by proper branching strategies. Tarjan and Trojanowski [Tarjan and Trojanowski, 1977] proposed a recursive algorithm for the maximum independent set problem with the time complexity of $O(2^{\frac{n}{3}})$. Later, this result was improved by Robson [Robson, 1986], who modified the algorithm of Tarjan and Trojanowski to obtain the time complexity of $O(2^{0.276n})$. Another important implicit enumerative algorithm for the MCP was developed by Balas and Yu [Balas and Yu, 1986] in 1986. Using an interesting new implementation of the implicit enumeration, they were able to compute maximum cliques in graphs with up to 400 vertices and 30,000 edges.

In 1990, Carraghan and Pardalos [Carraghan and Pardalos, 1990] proposed yet another important branch and bound algorithm for the maximum clique problem based on examining vertices in the order corresponding to the nondecreasing order of their degrees. It is an important exact algorithm for the MCP, since many exact algorithms are based on such a basic and simple branch and bound solver. Despite its simplicity, the approach proved to be very efficient, especially for sparse graphs. Since then, a lot of efforts have been devoted to improving its performance, most attempts to improve on this straightforward algorithm are based on methods for calculating upper bounds (other than from the size of the induced subgraph) during the search for the purpose of more efficiently pruning subtrees. Almost without exceptions, such upper bounds are obtained from the graph coloring problem, based on the following property.

In a graph coloring, adjacent vertices must be assigned different colors. If a graph, or an induced subgraph G, can be colored with, say, k colors, then the subgraph G, cannot contain a clique of size $k + 1$, i.e., $\omega(G) \leq k$ where $\omega(G)$ denotes the cardinality of a maximum clique of the induced subgraph G.

The algorithm of Carraghan and Pardalos [Carraghan and Pardalos, 1990] was further improved by Algorithm Cliquer [Östergård, 2002], a branch-and-bound algorithm which analyzes vertices in order defined by their coloring and employs a new pruning strategy. In [Östergård, 2002], the author compared the performance of this algorithm with several other approaches on random graphs and DIMACS benchmark instances and claimed that his algorithm is superior in many cases.

Fahle [Fahle, 2002] also improves the algorithm of Carraghan and Pardalos [Carraghan and Pardalos, 1990], by using the constructive heuristic DSATUR to color vertices one by one, and by partitioning in parallel the graph into independent sets. As vertices are colored or inserted into an independent set in decreasing and increasing order of their degree respectively, four heuristic solutions of graph coloring are obtained, and the best one is used as the upper bound.

Regin [J.C.Regin, 2003] uses an upper bound based on a matching algorithm. A matching, which corresponds to a set of independent sets of size 2 here, is computed by traversing the vertices of a graph and considering that an edge exists if two vertices are not connected.

MCQ [Tomita and Seki, 2003] colors vertices in a predetermined order. Suppose that the current independent sets are S_1, S_2, ..., S_k (in this order, k is 0 at the beginning of the coloring process), MCQ inserts the current first vertex v into the first S_i such that v is non-connected to all vertices already in S_i. If such a S_i does not exist, a new independent set S_{k+1} is opened and v is inserted here. After all vertices are partitioned into independent sets, they are reordered according to their independent set, vertices in S_i coming before vertices in S_j if $i < j$. This coloring process is executed for G_v after each branching on the vertex v. The predetermined order of vertices in G_v is inherited from G. MCR [Tomita and Kameda, 2007] improves MCQ with a better initial order of vertices in the initial input graph, but uses the same coloring process to compute the upper bound.

MaxCliqueDyn [Konc and Janezic, 2007] is also improved from MCQ. While MCQ (as well as MCR) only computes the degree of vertices at the root of the search tree for the initial input graph, MaxCliqueDyn dynamically recomputes the degree of vertices at some nodes near the root of the search tree chosen using a parameter, and re-orders the vertices in the decreasing order of their degree before coloring these vertices. The dynamic degree computation near the root of the search tree makes MaxCliqueDyn faster than MCQ for random graphs when their density is between 0.7-0.95, but slower than MCQ when the graph density is smaller than 0.7.

In [Li and Quan, 2010], Li and Quan proposed a new encoding from MCP into MaxSAT and used MaxSAT technology to improve the upper bound based on a partition P of a graph into independent sets for a maximum clique of the graph, which cannot be very tight for imperfect graphs. In this way, the strength of specific algorithms for MCP in partitioning a graph and the strength of MaxSAT technology in propositional reasoning are naturally combined to solve the MCP. The authors compared the performance of this algorithm with several other approaches and claimed that their approach is very effective on hard random graphs and on DIMACS benchmarks.

Though exact methods have been frequently used to solve the MCP and these methods have the theoretical advantage of finding optimal solutions to a given problem, their applications are generally limited to instances of moderate sizes. To handle the maximum clique problem on very large graphs, we have to resort to heuristic methods to find sub-optimal solutions within reasonable time. In the following section, we briefly review some representative heuristics for the MCP.

1.4 Heuristic methods for the MCP

The inherent computational complexity of the MCP has led many researches to reassess their strategy for attacking these problems. Using heuristic techniques is a popular alternative which is less bleak than attempting to solve the MCP exactly. Any approach without a formal guarantee of performance can be considered as a “heuristic”. Although such methods are not satisfying mathematically, they have shown to be very useful in practical situations since they are able to provide solutions of acceptable quality with reasonable computing efforts. Essentially, heuristic algorithms for the MCP mainly belong to four main solution approaches including sequential greedy algorithms, local search methods, evolutionary population-based hybrid or distributed approaches and heuristics based on continuous optimization.

1.4.1 Greedy algorithms

Greedy heuristics are simple and intuitive algorithms which make greedy choices to find a solution to an optimization problem. Greedy algorithms construct feasible solutions from scratch by making, in each step, the most favorable choice for a decision variable. Each choice depends on the decisions made in the previous steps, but the effects that this choice will have in the following steps are unknown. Such choices can thus be viewed as local decision rules that generally lead to sub-optimal solutions, since the resulting solution at the end of construction is not known, and future decisions may have a large impact on the resulting quality of the solution.

The majority of greedy algorithms in the literature for the maximum clique problem are called “sequential greedy heuristics”. These heuristics generate a maximal clique through the repeated addition of a vertex into a partial clique, or the repeated deletion of a vertex from a set that is not a clique.

Kopf and Ruhe [Kopf and Ruhe, 1987] named these two classes of heuristics the *Best in* and the *Worst out* heuristics. Decisions on which vertex to be added in or moved out next are based on some particular indicators associated with candidate vertices. For example, a possible *Best in* heuristic constructs a maximal clique by repeatedly adding in a vertex that has the largest degree among candidate vertices. In this case, the indicator is the degree of a vertex. On the other hand, a possible *Worst out* heuristic can start with the whole vertex set V. It will repeatedly remove a vertex out of V until V becomes a clique.

Kopf and Ruhe [Kopf and Ruhe, 1987] further divided the above two classes of heuristics into *New* and *Old* (*Best in* or *Worst out*) heuristics. Namely, if the indicators associated with candidate vertices are updated every time a vertex is added in or moved out, then the heuristic is called a *New* heuristic. Otherwise it is called an *Old* heuristic. We can find in the literature that most heuristics for the maximum clique problem fall in one or the other classes. See for example, the greedy algorithm of Johnson [Johnson, 1974], and the greedy algorithm of Tomita et al [Tomita *et al.*, 1988]. The differences among these heuristics are their choice of indicators and how indicators are updated. These sequential greedy heuristics are very fast by nature but their quality is generally unsatisfactory.

1.4.2 Local search heuristics

1.4.2.1 Local search strategies for the MCP

When designing a local search algorithm for solving a particular problem, one has to define the search space to be explored, the evaluation function to be minimized, and the neighborhood function. This triplet is what we call a search strategy. We propose to classify the search strategies for the MCP into two categories. The first one solves the MCP, while the second one consider the size k of cliques as fixed, and tries to find a clique of a fixed size k.

- The legal strategy: the search space Ω contains all legal cliques and the goal is to find a solution $c \in \Omega$ whose size is as large as possible.

- The k-fixed penalty strategy: the size k of the clique is fixed, the search space Ω contains all (not necessarily legal cliques) the vertex subsets of fixed size k (k-subsets) including both feasible and infeasible cliques, and the goal is to determine a legal clique $c \in \Omega$.

For the k-fixed penalty strategy, as noted in [Friden *et al.*, 1989], the maximum clique problem can be approximated by finding a series of k-cliques for increasing values of k (a k-clique is a clique of size k). Each time a k-clique is found, k is incremented by one and a new (larger) k-clique is sought. This process is repeated until no k-clique can be found. The last k-clique constitutes an approximation of the maximum clique of the graph. Consequently, the maximum clique problem comes down to the problem of finding k-cliques.

In the k-fixed penalty strategy, a solution can be represented as a subset c of vertices of size k. The evaluation function $f(c)$ counts the number of edges induced by c, and the objective can simply be to maximize the number of edges contained in c such that $f(c)$ reaches its maximal value $f(c) = k * (k-1)/2$, which means any two vertices of c are connected by an edge and c is a legal k-clique. Neighbor solutions can be obtained by swapping a vertex in c with another vertex in $V \setminus c$. The k-fixed penalty strategy was explored in [Friden *et al.*, 1989; Fleurent and Ferland, 1996; Geng *et al.*, 2007; Wu and Hao, 2011a].

For the legal strategy [Katayama *et al.*, 2005; Battiti and Protasi, 2001; Pullan and Hoos, 2006; Pullan, 2006; Wu and Hao, 2012f], the admissible search space Ω is the set of all legal cliques. The function to be maximized is the clique size $f(c) = |c|$, and the neighborhood $N(c)$ consists of all cliques that can be obtained from c by adding or dropping a single vertex. For the legal strategy, there exists three basic move operators in the literature: the *add* move which consists in adding a vertex in $V \setminus c$ to c, the *drop* move defined by dropping a vertex from c, and the *swap* move which consisting in swapping a vertex in c with another vertex in $V \setminus c$. Note that the *swap* move can be trivially decomposed into two separate moves, i.e., a *drop* move followed by an *add* move. In previous studies, most local search methods based on the legal strategy use two or three of these move operators. For instances, algorithms like PLS [Pullan, 2006], DLS [Pullan and Hoos, 2006] and KLS [Katayama *et al.*, 2005] employ only the add and swap moves, while the MN/TS algorithm [Wu and Hao, 2012f] and the RLS algorithm [Battiti and Protasi, 2001] use all of these move operators.

1.4.2.2 Tabu search heuristics for the MCP

Tabu search (TS) is a local search metaheuristic whose essential feature is the usage of memory. This history-based heuristic was proposed independently by Glover [Glover, 1989], and by Hansen and Jaumard [Hansen and Jaumard, 1990] who used the term steepest ascent mildest descent (SAMD). In general, TS aims at maximizing a function f by using an iterative modified local search (assume that here we consider a standard maximization problem). At each step of the iterative process, the selected move is the one that produces the highest f value in the neighborhood. This move is executed even if f decreases with respect to the value at the current point, to exit from local optima. As soon as a move is applied, the inverse move is prohibited (i.e., not considered during the neighborhood evaluation) for the next T iterations (called the tabu tenure). Prohibitions can be realized by using a first-in first-out list of length T (the "tabu list"), where the inverse of moves enter immediately after their execution, are shifted at each iteration, and therefore exit after T steps. A move is prohibited at a given iteration if and only if it is located in the "tabu list". The tabu tenure controls the degree of diversification introduced into the search. The longer the tabu list the stronger the introduced diversification.

Various versions of tabu search have been successfully applied to the maximum independent set and maximum clique problems. The first tabu search heuristic for the maximum stable set problem (or maximum independent set problem) is STABULUS, which was proposed by Friden et al. in 1989 [Friden *et al.*, 1989]. STABULUS is based on the k-fixed penalty strategy and aims at finding a legal independent set of size k. The algorithm tries to minimize the number of edges contained in the current subset of k vertices, while aiming at reducing this number to zero. The basic move of STABULUS consists in swapping a vertex in the current subset with another vertex out of the current subset. For determining whether a move is tabu or not, STABULUS first uses one tabu list T_1 containing the $|T_1|$ last solutions. In addition, STABULUS uses two other lists T_2 and T_3 containing the last vertices which were removed from (or introduced into) the independent set for the purpose of preselection. STABULUS was later improved by Fleurent and Ferland [Fleurent and Ferland, 1996].

In [Gendreau *et al.*, 1993], Gendreau et al. considered the legal strategy. They proposed three different versions of TS for the MCP, and successfully compared them with an iterated version of STABULUS. Two of the newly introduced TS versions are deterministic, one (ST) based on a single tabu list of the last $|T_1|$ solutions visited, the other (DT) adding a second list of the last $|T_2|$ vertices deleted. Only additions of vertices to the current clique can be restricted (deletions are always possible). The third version (PT) is stochastic: let S_t be the set of the vertices that can be added to the current clique c, if $|S_t > 0|$, a random sample of S_t is considered for a possible (non-tabu) addition, otherwise, if the current solution c is a local optimum and no vertices can be added, a number of randomly extracted vertices in S_t are removed from it.

In [Battiti and Protasi, 2001], Battiti and Protasi proposed a Reactive Local Search (RLS) algorithm for the MCP. RLS considered also the legal strategy and has been derived from Reactive Tabu Search [Battiti and Tecchiolli, 1994], which is an advanced and general tabu search method that automatically adapts the tabu tenure parameter for determining the appropriate amount of diversification. The amount of diversification with RLS is controlled by the prohibition parameter T which is determined by a feedback from the search history, and an explicit memory-influenced restart which is activated periodically as a long-term diversification tool assuring that each vertex is eventually tried as part of the current clique. RLS shows competitive performance on the set of DIMACS maximum clique instances both in terms of solution quality and computing time.

Very recently, Wu and Hao proposed an adaptive multistart tabu search approach (AMTS) to solve the MCP [Wu and Hao, 2011a]. AMTS is based on the k-fixed penalty strategy and integrates several distinguished features such as a constrained neighborhood for more efficiently exploring the search space, a dynamic tabu tenure mechanism to prevent the algorithm from revisiting previous encountered solutions, and a frequency-based restart strategy to escape from local optima. This algorithm will be presented in Chapter 2 of this thesis.

In [Wu and Hao, 2012f], the authors porpose a multi-neighborhood tabu search (MN/TS) for the maximum vertex weight clique problem. MN/TS is based on the legal strategy and combines three neighborhoods induced by three basic move operators (the add, swap and drop move) for the purpose of effectively exploring the search space. The particularity of the combined neighborhood relies on the union of the underlying neighborhoods instead of the conventional sequential exploration of basic neighborhoods. At each iteration of the algorithm, MN/TS explores the union of these three neighborhoods and selects the overall best admissible neighboring solution. In addition, MN/TS integrates a dedicated tabu mechanism and a randomized restart strategy.

1.4.2.3 Simulated annealing heuristics for the MCP

Simulated annealing [Kirkpatrick *et al.*, 1983] proceeds in the same way as ordinary local search but incorporates some randomization in the move selection to avoid getting trapped in a local optimum by means of non-improving moves. These moves are accepted according to probabilities taken from the analogy with the annealing process.

Geng [Geng *et al.*, 2007] presented a simulated annealing algorithm (SAA) for the MCP. In a given iteration, the SAA method, which was based on the k-fixed penalty strategy, generates a random move (an exchange between a vertex in the current subset and a vertex outside the current subset). If it is an improving move, it is automatically performed; otherwise, it may still be made with a certain probability (according to the Boltzmann distribution) using a parameter called temperature. The algorithm starts with an initial temperature $T_I = 100.0$ which is reduced progressively according to the factor $\alpha_2 = 0.9995$. For each temperature value, $L = 8 * |V|$ moves are generated, evaluated and performed if the acceptation condition is verified. The SAA method terminates when the current temperature $t < T_S$, where T_S is the end temperature which is set equal to 0.001.

1.4.2.4 Variable neighborhood search for the MCP

Variable neighborhood search (VNS) [Hansen and Mladenović, 2003] is based on a simple idea: a systematic change of the neighborhood within a local search algorithm.

In [Hansen *et al.*, 2004], the authors propose a VNS for the MCP. In their implementation, the minimum transversal or minimum vertex cover problem is considered. For VNS, the distance metric between two solutions T and T' is the difference in cardinalities of vertex sets T and T'. Then the neighborhood $N_k(T)$ consists of all solutions at distance k from T. Three types of moves are considered for the sets of selected neighborhood structures N_k, $k = 1, ..., k_{max}$ of VNS: *drop*, *add*, and *interchange*. An initial solution is obtained by a variable neighborhood descent (VND) heuristic, which is later used as the local search procedure of the proposed VNS approach. It combines a greedy selection rule with a simplicial vertex test, and uses *add* moves in its descent phase and *exchange* moves in a plateau phase. Upon reaching a local optimum with respect to both the add and 1-*interchange* neighborhoods, a new solution is randomly generated from the k^{th} neighborhood by dropping k vertices from the current clique.

1.4.2.5 Phased local search for the MCP

Phased local search (PLS) is a stochastic local search algorithm which interleaves different sub-algorithms designed for different types of problem instances. During each phase of the PLS, a sub-algorithm is selected to run.

In [Pullan, 2006], the authors propose a PLS for the MCP. The proposed approach was motivated by the observation that, on the whole set of DIMACS maximum clique instances, "any elevated performance of an algorithm over one class of problems is exactly paid for in performance over another class" [Wolpert and Macready, 1997]. PLS, based on based on the legal strategy, interleaves 3 sub-algorithms which alternate between sequences of iterative improvement, during which suitable vertices are added to the current clique, and plateau search, where vertices of the current clique are swapped with vertices not contained in the current clique. The sub-algorithms differ in their vertex selection techniques in that selection can be solely based on randomly selecting a vertex, randomly selecting within highest vertex degree or randomly selecting within vertex penalties that are dynamically adjusted during the search. In addition, the perturbation mechanism used to overcome search stagnation differs between the sub-algorithms. PLS has no instance dependent parameters and achieves state-of-art performance over on a range of widely studied DIMACS benchmark instances. Moreover, Pullan extends PLS to handle the maximum vertex weight clique problem and the maximum edge weight clique problem [Pullan, 2008].

1.4.2.6 Dynamic local search for the MCP

In [Pullan and Hoos, 2006], Pullan and Hoos introduce DLS-MC, a new stochastic local search algorithm for the MCP. DLS-MC, based on the legal strategy, alternates between a clique expansion phase and a plateau search phase. During the expansion phase, one seeks to expand a clique of size k to a new clique of size k +1 by adding a vertex which is adjacent to all the vertices of the current clique. When the current clique cannot be expended, one switches to plateau search during which vertices of the current partial clique are swapped with vertices outside the partial clique. Once the current clique can be further expanded, one switches back to the expansion phase and so on. The selection of vertices is solely based on vertex penalties that are dynamically adjusted during the search, and a perturbation mechanism is used to overcome search stagnation. The behaviour of DLS-MC is controlled by a single parameter, penalty delay, which controls the frequency at which vertex penalties are reduced. DLS-MC is evaluated on a wide range of commonly used benchmark instances from DIMACS library. Experiments show that DLS-MC archives highly competitive results compared with a number of state-of-the-art algorithms for the MCP.

1.4.2.7 Cooperating local search for the MCP

A recent cooperating local search (CLS) has shown excellent performance for the maximum clique problem on both DIMACS and BOSHLIB benchmarks [Pullan *et al.*, 2011]. CLS is a hyper-heuristic algorithm [Burke *et al.*, 2003], where the term hyper-heuristic defines a heuristic approach that seeks to automate the process of selecting, combining, generating or adapting a number of low level heuristic for a more effective search. CLS is also a parallel method intended to be run on desktop multi-core computers which have dramatically improved the usability of parallel algorithms.

Following the idea of PLS, CLS incorporates four low level heuristics which alternate between sequences of iterated improvement and plateau search. The difference between these low level heuristics lies in their vertex selection techniques and the way they deal with plateaus. CLS controls copies of these low level heuristics, which are allocated to available processor cores. The performance of CLS is improved by passing relevant information between low level heuristics in order to guide the search to particular areas of the search domain. CLS requires no run-time parameters and shows excellent performance for the MCP on both DIMACS and BOSHLIB benchmarks.

1.4.3 Evolutionary algorithms and their hybrids

Inspired by the principles and nature of evolution, several evolutionary algorithms (EAs) have been proposed since the early 1960s, including genetic algorithms [Holland, 1975], evolutionary strategies [Rechenberg, 1973], evolutionary programming [Fogel *et al.*, 1966], distribution estimation algorithms [Larrañaga and Lozano, 2001], scatter search [Laguna and Martrí, 2003] and memetic algorithms [Moscato, 1989; Hao, 2011]. Although each of these constitutes a different approach, they all operate on a population of candidate solutions applying the principle of survival of the fittest to produce a sequence of increasingly better quality individuals (i.e., solutions).

Early attempts to apply genetic algorithms to the maximum independent set and maximum clique problems were made in the beginning of 1990s. Many implementations have appeared in the literature ever since [Murthy *et al.*, 1994; Foster and Soule, 1995; Sakamoto *et al.*, 1997; Thierens, 1999]. Most of the genetic algorithms can be easily parallelized. However, the work of Cater and Park [Carter and Park, 1993; Park and Carter, 1994] shows that a pure genetic algorithm is not effective for the MCP. It is now acknowledged that in order to improve their performances for the MCP, genetic algorithms need to be customized or to incorporate other techniques. Several attempts have been made to combine genetic algorithms with local searches for solving the MCP [Fleurent and Ferland, 1996; Marchiori, 1998; Singh and Gupta, 2006].

In [Marchiori, 2002], the author proposed GENE, a genetic local search method for the MCP. GENE employs roulette wheel selection, uniform crossover and swap mutation, and generates subgraphs, which are then transformed into maximal clique by the local search heuristic. The local search heuristic consists of three steps: perturb, repair and extend. The perturb step modifies the subgraph obtained by the genetic algorithm through removing and adding some vertices quasi randomly. The repair step then transforms this graph into a clique. The extend step extends this clique into a maximal clique. The GENE algorithm outperforms other existing genetic algorithms on the DIMACS benchmark graphs in terms of quality of solutions and speed.

The results of GENE is further improved by EA/G [Zhang *et al.*, 2005], a hybrid evolutionary algorithm with guided mutation for the MCP. In EA/G, guided mutation is first used to generate offsprings, which are then subjected to Marchiori's local search heuristic [Marchiori, 2002] for further improvement. The guided mutation in EA/G can be regarded as a combination of the conventional mutation operator and the offspring generating scheme of the estimation of distribution algorithms (EDAs). The basic idea behind the proposed guided mutation is to combine the global statistical information and location information of the solutions found so far to overcome the shortcoming of GAs and EDAs. Besides guided mutation, EA/G adopts a search strategy which divides the whole search space into several search areas, and in each phase the algorithm focus on a single search area. Experimental results of EA/G on DIMACS graphs indicate that this new algorithm outperforms the GENE algorithm.

Starting from the same evolutionary framework of EA/G, Brunato and Battiti propose a reactive evolutionary algorithm (R-EVO) for the MCP [Brunato and Battiti, 2011]. R-EVO uses an evolutionary scheme in the framework of estimation of distribution algorithms to generate new individuals, which are then subjected to memetic evolution through a simplified Reactive Search Optimization (RSO) method [Battiti and Brunato, 2008]. In this manner, each individual in the population executes a short local search with prohibition. The prohibition period is determined in a simple reactive manner on a specific instance based on the estimated size of the maximum clique. The results obtained by R-EVO demonstrate that R-EVO is competitive with respect to EA/G.

1.4.4 Heuristics based on continuous optimization

As opposite to the combinatorial approaches, various continuous nonconvex formulations of the clique problem are studied in order to develop and apply continuous optimization methods. Massaro et al. [Massaro *et al.*, 2001] presented a PBH algorithm which is based on a linear complementarity formulation of the clique problem. The algorithms, described in [Bomze *et al.*, 2012], and [Kuznetsova and Strekalovsky, 2001] are based on the continuous formulation of MCP given by Bomze [Bomze, 1997]. The Max-AO algorithm by Burer et al. [Burer *et al.*, 2002] employ a low-rank restriction upon the primal semidefinite program computing the Lovász number of a graph. The QUALEX-

MS algorithm of Busygin [Busygin, 2006] follows the idea of finding stationary points of a quadratic function over a sphere. It is based on the new generalized version of the Motzkin-Straus [Motzkin and Straus, 1965] quadratic programming formulation for the MCP. Furthermore, the Motzkin–Straus theorem and its extension were successfully employed in solving the maximum vertex weight clique problem [Bomze *et al.*, 2000] and the maximum edge weight clique problem [Pavan and Pelillo, 2007].

1.4.5 Other popular heuristics

1.4.5.1 Ant colony optimization (ACO)

Ant colony optimization (ACO) [Dorigo *et al.*, 1996; Dorigo and Caro, 1999] is inspired by the effective cooperation of ants in finding, with limited cognitive abilities, the shortest way from their nest to a food source. The first ant algorithm applied to the maximum clique problem is Ant-Clique [Fenet and Solnon., 2003]. Basically, Ant-Clique successively generates maximal cliques through the repeated addition of vertices into partial cliques. ACO is introduced as a heuristic for choosing, at each step, the vertex to enter the clique: this vertex is chosen with respect to a probability that depends on pheromone trails laying between it and the clique under construction, while pheromone trails are deposited by ants proportionally to the quality of the previously computed cliques. The results obtained by Ant-Clique demonstrate that Ant-Clique is able to find larger cliques on a majority of the DIMACS benchmark instances with respect to GENE [Marchiori, 2002]. Later, Ant-Clique is extended and further improved by the authors in another paper [Solnon and Fenet, 2006] in which different strategies for laying pheromone are exploited.

Another version of ant colony optimization for the maximum clique problem (ASMC) was proposed in [Bui and Rizzo, 2004]. ASMC is rather different from Ant-Clique as ants are distributed, i.e., each ant only has local knowledge of the graph, so that the algorithm can be implemented in a distributed system. As a counterpart, performance of ASMC by means of solutions' quality are rather far from performance of Ant-Clique and other non distributed algorithms.

1.4.5.2 Neural Network

Introduced by Hopfield and Tank in 1985 [Hopfield and Tank, 1985], the Neural Network is widely utilized to solve a number of classic combinatorial optimization problems such as the traveling salesman problem. As far as the MCP is concerned, early attempts at encoding the maximum clique and related problems in terms of a neural network were already done in the late 1980's [Ramanujam and Sadayappanv, 1988]. However, little or no experimental results were presented, thereby making it difficult to evaluate the merits of these algorithms. In [Lin and Lee, 1993], Lin and Lee used the quadratic zero-one formulation as the basis for their neural network heuristic. On random graphs with up to 300 vertices, they found their algorithm to be faster than the implicit enumerative algorithm in [Carraghan and Pardalos, 1990], while obtaining slightly worse results in terms of clique size.

Grossman [Grossman, 1993] proposed a discrete, deterministic version of the Hopfield model for the MCP, originally designed for an all-optical implementation. The model has a threshold parameter which determines the character of the stable states of the network. The author suggests an annealing strategy on this parameter, and an adaptive procedure to choose the network's initial state and threshold. On DIMACS graphs the algorithm performs satisfactorily but it does not compare well with more powerful heuristics such as simulated annealing.

Gu and Yu proposed a Chaotic Neural Network for the MCP [Gu and Yu, 2004]. From analyzing the chaotic states of the neuron output and computational energy, the authors reach the conclusion that, unlike the conventional Hopfield neural networks for the MCP, Chaotic Neural Network can avoid getting stuck in local minima and thus yields competitive solutions. Extensive experiments are carried out by the authors to demonstrate that Chaotic Neural Network provides a more effective and efficient approach than conventional Hopfield neural networks to solve the MCP.

1.5 Applications

In practical terms, the maximum clique and related problems have widespread applications in numerous areas [Pardalos and Xue, 1994], as for example:

- project selection [Christofides, 1975].
- classification theory [Barahona *et al.*, 1992].
- fault tolerance [Berman and Pelc, 1990; Lovász, 1979].
- coding theory [Deo, 1974; MacWilliams and Sloane, 1979; Sloane, 1998].
- computer vision [Ballard and Brown, 1982].
- economics [Avondo-Bodeno, 1962].
- information retrieval [Reigold *et al.*, 1977].
- signal transmission theory [Reigold *et al.*, 1977].
- aligning DNA [Miller, 1993; Vingron and Pevzner, 1992; Vingron and Argos, 1991].
- protein sequences [Lecky *et al.*, 1989; Vingron and Pevzner, 1992; Strickland *et al.*, 2005].

In fact, an exhaustive list of (direct or implicit) applications would perhaps contain dozens or hundreds of references, simply because the clique model is very general. The above list comprises only the most representative applications. Next, we provide some relevant practical applications of the MCP. In addition, some problems arising in these applications can be used as test problems for algorithm comparison.

We start with an application in computing the maximum number of codewords in a binary constant-weight code with length n, Hamming distance at least d, and weight w.

This number is called $A(n, d, w)$. Also let $A(n, d)$ be the maximum number of binary codewords of length n and Hamming distance at least d without consideration on weight. To compute $A(n, d)$, one forms the graph with 2^n vertices, corresponding to all possible code-words, joins two vertices by an edge if their Hamming distance is at least d, and finds the maximum clique. Similarly, finding the largest code invariant under a given permutation group, requires finding the maximum clique in a graph with weights attached to the vertices. More details on computing the maximum number of constant weight codes are found in [Brouwer *et al.*, 1990].

The next application occurs in bioinformatics. Computing the similarity between two protein structures is a crucial task in molecular biology, and has been extensively investigated. In [Malod-Dognin *et al.*, 2010], the authors propose a protein structure comparison method based on internal distances which is posed as a maximum clique problem in an alignment graph.

In graph-theoretic language, two proteins P_1 and P_2 can be represented by two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where the sets of vertices V_1 and V_2 stand for residues/SSE, while edges depict contacts/relationships between them. The similarity between P_1 and P_2 can be estimated by finding the longest alignment between the elements of V_1 and V_2. In [Malod-Dognin *et al.*, 2010], this is modeled by an alignment graph $G = (V, E)$ of size $|V_1| \times |V_2|$, where each row corresponds to an element of V_1 and each column corresponds to an element of V_2. A vertex $i \times k$ is in V (i.e. matching $i \longleftrightarrow k$ is possible), only if elements $i \in V_1$ and $k \in V_2$ are compatible. An edge $(i \times k, j \times l)$ is in E if and only if : (1) $i < j$ and $k < l$, for order preserving, and (2) matching $i \times k$ is compatible with matching $j \times l$. A feasible alignment of P_1 and P_2 is then a clique in G, and the longest alignment corresponds to a maximum clique in G.

In addition to its practical applications in real-life problems, the MCP is tightly related to a quite number of important combinatorial optimization problems such as graph coloring, clique partitioning and community detection in complex networks. Moreover, some of them can be directly transformed into the maximum clique problem, such as the set packing problem, the optimal winner determination problem, the set partitioning problem and the double subgraph isomorphism problem. Given the simplicity of the MCP, all these problems can be first mapped onto the MCP and then be modeled by some effective MCP based approaches. In the following, we provide some interesting examples.

The first application occurs in graph coloring. Recall that a legal k-coloring of a given graph $G = (V, E)$ is a partition of V into k independent sets (color classes). Approaches based on independent sets extraction were proposed as early as 1987 in [Chams *et al.*, 1987] and it is now used for coloring large graphs [Chams *et al.*, 1987; Fleurent and Ferland, 1996; Hertz and de Werra, 1987; Johnson *et al.*, 1991; Wu and Hao, 2012a]. As observed in many studies, it is difficult, if not impossible, to find a proper k-coloring of a large graph $G = (V, E)$ (e.g., with 1000 vertices or more) with k close to $\chi(G)$ by applying directly a given coloring algorithm on G. A basic approach to deal with large graphs is to apply the general principle of "reduce-and-solve". This approach is composed of a preprocessing phase followed by a coloring phase.

The preprocessing phase typically identifies and removes some (large) independent sets from the original graph to obtain a reduced subgraph (called "residual" graph). The subsequent coloring phase determines a proper coloring for the residual graph. Given the residual graph is of reduced size, it is expected to be easier to color than the initial graph. Now it suffices to consider each extracted independent set as a new color class (i.e., by assigning a new color to all the vertices of each of these sets). The coloring of the residual graph and all the extracted independent sets give a proper coloring of the initial graph. This approach were explored with success in early studies like [Chams *et al.*, 1987; Fleurent and Ferland, 1996; Hertz and de Werra, 1987; Johnson *et al.*, 1991].

Algorithms based on this approach can use different methods to find a large independent set in the graph. In [Chams *et al.*, 1987], this was achieved with a simple greedy heuristic while in [Fleurent and Ferland, 1996; Hertz and de Werra, 1987], large independent sets were identified by a dedicated tabu search algorithm. In [Johnson *et al.*, 1991], the authors introduced the XRLF heuristic which operates in two steps. First, a number of independent sets are collected using Leighton's Recursive Largest First (RLF) heuristic [Leighton, 1979]. Then, an independent set is iteratively selected and extracted from the graph such that its removal minimizes the density of the reduced graph. This process continues until the residual graph reaches a given threshold.

Another problem which is equivalent to the MCP is the max-min diversity problem. In [Croce *et al.*, 2009], the authors proposed an effective MCP based heuristic approach for the max-min diversity problem. Basically, their approach relies on the equivalence between the max-min diversity problem and the MCP, it solves different decision problems about the existence of cliques with a given size. The idea is rather simple but, according to the experiments and the comparison with the existing literature, appears as particularly promising. In fact, it outperforms, both in quality and CPU time, the existing state of the art algorithms dedicated for the max-min diversity problem.

Given a set of elements $N = \{s_1, s_2, ..., s_n\}$ and an $n \times n$ upper-triangular matrix D whose components d_{ij}, $1 \leq i < j \leq n$, measures the diversity of the two elements s_i and s_j, and an integer m such that $1 \leq m \leq n$. The max-min diversity problem consisting in selecting a subset $S \subseteq N$ such that $|S| = m$ and the objective function

$$f(S) = min\{d_{ij} : s_i, s_j \in S\} \tag{1.1}$$

is as large as possible.

As noted in [Croce *et al.*, 2009], the max-min diversity problem can be approximated by solving a series of decision problems about the existence of the subsets of elements. First, the distinct diversity values in the D matrix are sorted according to the increasing sequence such that:

$$\delta_1 < \delta_2 < ... < \delta_h \tag{1.2}$$

Their approach begins with the smallest $\delta_i (i = 1)$ and then tries to find a subset $S \subseteq N$ such that $|S| = m$ and its objective function value $f(S) \geq \delta_i$, each time such subset is found, i is increased by 1 and a new subset with larger objective function value is sought. The last δ_i constitutes an approximation of the max-min diversity value. Consequently, the max-min diversity problem comes down to a series of decision problems about the existence of the subsets of elements for the given objective function values.

Thus, the decision version of the max-min diversity problem asks whether there exists an m-tuple S with $f(S) \geq \delta_i$ for a given δ_i. This decision problem is in fact equivalent to the k-clique problem (see Section 1.4.2.1). Given δ_i, an instance of the k-clique problem is constructed as follows: we first set $k = m$ and define a graph $G = (V, E)$ where $V = \{1, 2, ..., n\}$ and edge $\{i, j\} \in E$ $(i, j \in V)$ if and only if $d_{ij} \geq \delta_i$. Now it is straightforward to see that if $S = \{s_{i_1}, ..., s_{i_m}\}$ is a m-tuple with $f(S) \geq \delta_i$, then $\{i_1, ..., i_m\}$ is a k-clique in the graph $G = (V, E)$. Consequently, to obtain an m-tuple S with $f(S) \geq \delta_i$, we can determine a k-clique in the corresponding graph $G = (V, E)$.

In addition to the graph coloring problem and the max-min diversity problem, MCP based approaches have been successfully applied to solve several other important combinatorial problems such as graph clustering [Schaeffer, 2007], minimum sum coloring [Wu and Hao, 2012b], mandatory coverage [Brotcorne *et al.*, 2002] and integration of genome mapping data [Harley *et al.*, 2001].

1.6 Benchmarks instances

In this thesis, we are interesting in the maximum clique and related problems, including the maximum vertex weight clique problem, the maximum edge weight clique problem, the graph coloring problem, the minimum sum coloring problem and the set packing problem. For each of these problems, we first developed heuristic approaches and then conducted extensive experimental evaluations and comparisons with the current state-of-art approaches on commonly used benchmark instances.

For the maximum clique problem, we conduct experiments on the following two sets of benchmark instances which are frequently used to assess MCP algorithms in the literature [Wu and Hao, 2011a; Wu and Hao, 2012f; Pullan, 2006; Pullan and Hoos, 2006; Pullan, 2008; Pullan *et al.*, 2011; Battiti and Protasi, 2001; Katayama *et al.*, 2005; Hansen *et al.*, 2004; Marchiori, 1998; Singh and Gupta, 2006; Zhang *et al.*, 2005; Brunato and Battiti, 2011]:

DIMACS benchmark: These instances[1], presented at the Second DIMACS Implementation Challenge, are the most frequently used for comparison and evaluation of MCP algorithms. More specifically, we use the following popular families:

- Brock - instances where the optimal clique is “hidden” by incorporating low-degree vertices.
- pHat - random instances having a wider range of vertex degrees. The generator for these instances is a generalization of the classical uniform random graph generator.
- MANN - clique formulation of the Steiner Triple Problem. These instances are created using Mannino’s code to convert the set covering formulation of the Steiner Triple Problem to clique problems.
- Kel - instances based on Keller’s conjecture on tilings using hypercubes.

[1]http://cs.hbg.psu.edu/txn131/clique.html

- Ham and Joh - graphs steaming from the coding theory.
- Gen - random instances with a unique known optimal solution.
- Other families such as C, San, SanR which are randomly generated using different methods.

The size of the DIMACS instances ranges from less than 50 vertices and 1,000 edges up to more than 3,300 vertices and 5,000,000 edges.

BOSHLIB benchmark: We also consider the more recent frb test instances[2] arising from the SAT'04 Competition. BOSHLIB instances are randomly generated graphs with hidden optimal solutions and appear to be more difficult than most of the instances from the DIMACS suite. The instance size ranges from 450 vertices and 17,794 edges up to 1,534 vertices and 12,7011 edges.

For the maximum vertex weight clique problem, we use the $DIMACS$-W and $BOSHLIB$-W benchmarks which are easily converted from the DIMACS and BOSHLIB instances by allocating weights to vertices in the following way. For vertex i, w_i is set equal to i mod $200 + 1$. This method for constructing weighted instances was initially proposed in [Pullan, 2008]. On the other hand, other maximum weight clique algorithms from the literature [Babel, 2008; Macambira and de Souza, 2000] use randomly generated graphs with randomly generated vertex weights, which makes it impossible to replicate exactly the benchmark experiments for comparisons.

For the maximum edge weight clique problem, which is also known as the maximum diversity problem (MDP), we carry out extensive experiments on the same sets of 120 test instances as in [Palubeckis, 2007], which are frequently used to assess algorithms for MDP. The details of the instance sets are described as follows [Duarte and Martí, 2007; Gallego *et al.*, 2009; Martí *et al.*, 2011]:

- Silva instances [3]: This data set consists of 20 instances, which were generated by Silva et al [Silva *et al.*, 2004]. The instance sizes are such that for $n = 100$, $m = 10$, 20, 30 and 40; for $n = 200$, $m = 20$, 40, 60 and 80; for $n = 300$, $m = 30$, 60, 90 and 120; for $n = 400$, $m = 40$, 80, 120, and 160; and for $n = 500$, $m = 50$, 100, 150 and 200.
- Random Type I instances (Type1_55, Type1_22 and Type1_52) [4]: These instances (60 in total) are based on matrices with real numbers generated from a (0, 10) uniform distribution. Random Type I was introduced by Duarte and Martí [Duarte and Martí, 2007] and includes 3 sets of instances. The first set (Type1_55) consists of 20 instances with $n = 500$ and $m = 50$, the second set (Type1_52) includes 20 instances with $n = 500$ and $m = 200$, and the third set contains 20 instances with $n = 2000$ and $m = 200$.

[2]http://www.nlsde.buaa.edu.cn/∼kexu/benchmarks/graph-benchmarks.htm
[3]http://www.optsicom.es/mdp/
[4]http://www.uv.es/∼rmarti/paper/mdp.html

- Random Type *II* instances (Type2): These instances (20 in total) are based on matrices with real numbers generated from a (0, 1000) uniform distribution. This data set was introduced by Duarte and Martí [Duarte and Martí, 2007] and have size of $n = 500$ and $m = 50$.

- Beasely instances (b2500): This data set consists of 10 instances, which were taken from the OR-Library [Beasley, 1996]. All the instances have 10% density with $m = 2500$ and $n = 1000$. These instances were used in [Palubeckis, 2007; Brimberg *et al.*, 2009; Lozano *et al.*, 2011; Wang *et al.*, 2012] to assess the MDP algorithms.

- Random larger instances (p3000 and p5000): These instances (10 in total) are based on matrices with integer numbers generated from a $[0, 100]$ uniform distribution. The density of the matrix is 10%, 30%, 50%, 80%, 100% respectively. There are five instances with $n = 3000$ and $m = 1500$, and five instances with $n = 5000$ and $m = 2500$. These instances were tested in [Palubeckis, 2007; Brimberg *et al.*, 2009; Lozano *et al.*, 2011; Wang *et al.*, 2012]. See http://www.soften.ktu.lt/~gintaras/max_div.html for the sources of the generator and input files to replicate these instances.

For the graph coloring problem as well as the minimum sum coloring problem, two sets of benchmark graphs from the literature are considered in the experiments. The first set is composed of 32 DIMACS instances [5], which are frequently used to test graph coloring algorithms as well as sum coloring algorithms [Wu and Hao, 2012a; Wu and Hao, 2012b] and can be summarized as follows:

- 12 random graphs (DSJCx.y). The first and second number in the name of each graph represent respectively the number of vertices and the edge density in the graph. The chromatic numbers of these graphs are unknown.

- 6 flat graphs (flat300_20_0, flat300_26_0, flat300_28_0, flat1000_50_0, flat1000_60_0 and flat1000_7_0). They are structured graphs with known chromatic number (respectively 20, 26, 28, 50, 60 and 76).

- 2 large random geometric graphs (R1000.1c, R1000.5). These graphs are generated by picking random points (vertices) in a plane and by linking two points situated within a certain geometrical distance. The chromatic number is unknown for R1000.1c and is equal to 234 for R1000.5.

- 3 very large random graphs (C2000.5, C2000.9, C4000.5). The chromatic numbers of these graphs are unknown. Due to the size and difficulty of these graphs, they are not always used in computational experiments in the literature.

- 1 latin square graph (latin_sqr_10) with unknown chromatic number.

- 8 Leighton graphs (le$X.Y$) with $X = 450$ vertices and with known chromatic number Y (they have a clique of size Y).

[5]http://www.optsicom.es/mdp/

The second set of benchmarks is composed of 27 graphs from the COLOR02 website[6] . Like the first set, these graphs are initially collected for the purpose of the COLOR02 competition.

For the set packing problem (SPP), we use the same 16 large, random SPP instances as in [Alidaee *et al.*, 2008], the size of the SPP instances ranges from 1000 elements to 2000 elements.

[6]http://mat.gsia.cmu.edu/COLOR02/

Chapter 2

An Adaptive Multistart Tabu Search Approach for Maximum Clique

In this chapter, we present an effective adaptive multistart tabu search algorithm (AMTS) for the maximum clique problem. The proposed AMTS algorithm employs the k-fixed penalty strategy, and integrates two complementary neighborhoods, a dynamic tabu tenure mechanism and a guided diversification strategy. To better overcome local optima, the basic TS engine is reinforced with a long term memory based restart strategy. Extensive experimental evaluations on the whole set of 80 DIMACS challenge benchmarks show that the proposed approach competes favorably with the current state-of-art approaches for the MCP. It is able to attain consistently the largest known clique for 79 benchmarks out of the 80 instances. The content of this chapter is published in [Wu and Hao, 2011a].

Contents

2.1 Adaptive multistart tabu search (AMTS) for MCP

In this chapter, we follow the basic idea of [Friden *et al.*, 1989] and develop an effective heuristic algorithm based on tabu search [Glover, 1989]. Like STABULUS (see Section 1.4.2.2), the proposed adaptive multistart tabu search approach is based on the k-fixed penalty strategy, and searches a legal k-clique within a space of subsets S of size k. Yet, the main components of our AMTS algorithm are different from those of STABULUS. In particular, to allow the algorithm to explore more efficiently the search space, the swap operations of AMTS are limited to vertices from two critical subsets A (a constrained subset of the candidate solution S) and B (a constrained subset of $V \backslash S$) (Section 2.1.2.2). To escape from local optima, AMTS applies both a deterministic selection rule (Section 2.1.2.3) and a probabilistic selection rule to occasionally accept deteriorating solutions (Section 2.1.2.4). AMTS uses a dedicated tabu list to prevent the algorithm from revisiting previous encountered solutions (Section 2.1.2.5). Finally, to allow the algorithm to explore more new search regions, AMTS employs a frequency-based restart strategy (Section 2.1.3).

2.1.1 Solution strategy and general procedure

Our adaptive multistart tabu search algorithm follows the k-fixed penalty strategy (see Section 1.4.2.1), and is designed for finding a clique of fixed size k (denoted by k-clique) in a graph $G = (V, E)$. In this section, we describe the general procedure of AMTS while its components are detailed in Section 2.1.2.

For this purpose, we first define the search space Ω that is explored by AMTS. It is composed of all the vertex subsets S of fixed size k (k-subsets) including both feasible and infeasible cliques, i.e.,

$$\Omega = \{S \subset V : |S| = k\} \tag{2.1}$$

For any candidate solution $S \in \Omega$, its quality is assessed by the *evaluation function* $f(S)$ that counts the number of edges induced by S:

$$f(S) = \sum_{u,v \in S} e_{uv} \tag{2.2}$$

where $e_{uv} = 1$ if $\{u, v\} \in E$, $e_{uv} = 0$ otherwise.

Obviously, if a candidate solution S reaches the maximal value of this function, i.e., $f(S) = k * (k-1)/2$, any two vertices of S are connected by an edge and the candidate solution S is a legal k-clique. If $f(S) < k * (k-1)/2$, there must be at least two vertices in S which are not adjacent, consequently S is not a legal k-clique.

The objective of our AMTS algorithm is then to find in Ω a solution S that reaches the maximal value of f such that $f(S) = k * (k-1)/2$. The pseudo-code of AMTS is given in Algorithm 2.1.

AMTS explores the space Ω by employing an optimization procedure based on tabu search (we denote this procedure by TS^0). More specifically, AMTS generates first an initial solution (k-subset) in Ω which is built greedily in k steps from an empty set S. At

Algorithm 2.1: Adaptive multistart tabu search for the MCP

```
Require: Graph $G$, Integer $k$ (clique size), Integer $L$ (search depth), Integer $Iter_{max}$ (maximum
    allowed iterations)
Ensure: $k$-clique if found
1: Begin
2: $S \leftarrow Initialize(k)$ {Initial solution}
3: $Iter \leftarrow 0$ {Iteration counter}
4: while ($Iter < Iter_{max}$) do
5:    $S^* \leftarrow TS^0(S, k, L, Iter)$ {Apply the tabu search procedure TS$^0$ to improve $S$, §2.1.2}
6:    if $S^*$ is a legal $k$-clique then
7:       Return($S^*$) and Stop
8:    else
9:       $S \leftarrow FrequencyBasedInitialize(k)$ {Construction of a new solution $S$, §2.1.3}
10:   end if
11: end while
12: End
13: Return(Failure)
```

each step, a vertex $v \in V \backslash S$ is added to S such that v has the maximum number of edges that are connected to the vertices of S (ties are broken randomly).

From this initial solution S (a k-subset), AMTS runs TS^0 (Section 2.1.2) to improve S by maximizing the function f (Formula 2). During a round of TS^0, the search continues whenever TS^0 finds improved solutions. If the search is judged to be stagnating (the parameter L at line 5 is used for this purpose, see Section 2.1.2.1), the current round of TS^0 is stopped and a new starting solution is constructed whereupon a new round of TS^0 is launched. So a AMTS run is composed of multiple rounds of the TS^0 procedure. While each round of TS^0 examines in detail a region of the search space, each restart displaces the search to a new region.

The AMTS algorithm stops when a legal k-clique is found by TS^0, in which case the found k-clique is returned. AMTS may also stop when the total number $Iter$ of iterations attains a prefixed maximum number ($Iter_{max}$) without finding a legal k-clique. In this case, a failure is reported. $Iter_{max}$ is a user-defined parameter which specifies the maximal search effort allowed to solve a given instance. Next we present in detail the tabu search procedure TS^0.

2.1.2 The tabu search procedure

2.1.2.1 Main idea

Our tabu search procedure TS^0 is based on the well-known tabu search method ([Glover, 1989]) and its main scheme can be summarized in Algorithm 2.2. TS^0 operates on candidate solutions represented by k-subsets. S and S^* designate respectively the current solution and the best solution found so far (according to the evaluation function f defined in Section 2.1.1). I is an iteration counter used for the restart of TS^0 while $Iter$ is the global iteration counter used by AMTS in its stop test (see Algorithm 2.1).

Algorithm 2.2: The tabu search procedure TS^0 for k-clique finding

```
Require: Graph $G$, Initial solution $S$, Integer $k$ (clique size), Integer $L$ (depth of tabu search),
    Integer $Iter$ (iteration counter)
Ensure: The best solution $S^*$ found by the tabu search
 1: Begin
 2: $I \leftarrow 0$ {$I$ is the consecutive iterations during which $f(S)$ is not improved}
 3: $S^* \leftarrow S$ {$S^*$ records the best solution found so far}
 4: while ($I < L$) do
 5:    if There exist improving moves in neighborhood $CN$ then
 6:       Choose a best allowed $swap(u, v)$ {§2.1.2.2 and 2.1.2.3}
 7:    else
 8:       Choose $swap(u, v)$ according to the Prob. Move Select. Rule {§2.1.2.4}
 9:    end if
10:    $S \leftarrow S \backslash \{u\} \cup \{v\}$ {Move to the new solution}
11:    Undate the tabu list {§2.1.2.5}
12:    if $S$ is a legal $k$-clique then
13:       Return $S$ and Stop
14:    end if
15:    $Iter \leftarrow Iter + 1$
16:    if $f(S) > f(S^*)$ then
17:       $S^* \leftarrow S$
18:       $I \leftarrow 0$
19:    else
20:       $I \leftarrow I + 1$
21:    end if
22: end while
23: End
24: Return (Clique $S^*$)
```

For each *while* loop of Algorithm 2.2 (lines 4-23), TS^0 moves from the current solution S (a k-subset in Ω) to a new neighbor solution (another k-subset in Ω). For this, TS^0 uses two different rules to select a dedicated vertex u in S and a specific vertex v outside S (lines 5-6 and 7-8, see Sections 2.1.2.2-2.1.2.4), and then swaps u and v to obtain a new current solution (line 10). These swapped vertices are finally added in the tabu list preventing them from being selected again for the next iterations (line 11, see Section 2.1.2.5). If the new solution is a legal k-clique (i.e., $f(S) = k * (k-1)/2$), the algorithm stops and returns the k-clique found (lines 12-14). Otherwise, if the new solution S is better than the best solution S^* found so far ($f(S) > f(S^*)$), TS^0 updates S^* by S and continues to its next iteration (lines 16-21).

The *while* loop ends if no improved solution is found for L consecutive iterations (L is called the depth of search). In this case, the search is judged to be trapped in a deep local optimum. To escape from this local optimum, AMTS restarts TS^0 from a new starting point (see Section 2.1.3).

In the rest of this section, we provide a detailed description of the main ingredients of the TS^0 procedure while in Section 2.1.3, we explain the solution construction procedure for each restart of the TS^0 procedure.

2.1.2.2 Constrained swap move and neighborhood

To explore the search space Ω of k-subsets (Formula (2.1)), one naive way is to start with any k-subset $S \in \Omega$ and subsequently swap a vertex of S with another vertex of $V \backslash S$. Clearly, such a unconstrained swap (used in [Friden *et al.*, 1989]) induces a neighborhood of size $k * (|V| - k)$ which may be quite large. More importantly such a unconstrained neighborhood is not sufficiently focused and will not enable an efficient exploration of the search space. For this reason, we introduce below the *constrained neighborhood* which is both more focused and smaller-sized.

Let $S \in \Omega$ be a candidate solution (k-subset). For each vertex $v \in V$, let $d(v)$ denote the degree of v relative to the subset S:

$d(v) = |\{i \in S \mid \{i, v\} \in E\}|$

Let *tabu_list* be the tabu list containing the vertices that are currently forbidden for migration (see Section 2.1.2.5).

Let $MinInS = min\{d(u)|\ u \in S,\ u \notin tabu\_list\}$ and

Let $MaxOutS = max\{d(v)|\ v \in V \backslash S,\ v \notin tabu\_list\}$

Define:

$A = \{u \in S \mid u \notin tabu\_list,\ d(u) = MinInS\}$

$B = \{v \in V \backslash S \mid v \notin tabu\_list,\ d(v) = MaxOutS\}$

Now, to obtain a neighbor solution S' from S, we swap one vertex $u \in A$ against a vertex $v \in B$. This transition (from S to S') can conveniently be characterized by a move denoted by $swap(u, v)$ and written formally as: $S' = S \oplus swap(u, v)$ or equivalently $S' = S \backslash \{u\} \cup \{v\}$. All possible swap moves induced by A and B define our constrained neighborhood $CN(S)$, i.e.,

$$CN(S) = \{S' : S' = S \backslash \{u\} \cup \{v\}, u \in A, v \in B\} \tag{2.3}$$

Given the definition of $d(v)$, it is easy to see that function $f(S)$ (Formula (2.2)) can be rewritten as:

$$f(S) = \frac{1}{2} * \sum_{i \in S} d(i) \tag{2.4}$$

For a given $swap(u, v)$, the move gain Δ_{uv}, i.e., the variation in the function value f induced by the swap move, can be conveniently computed by:

$$\Delta_{uv} = f(S') - f(S) = d(v) - d(u) - e_{uv} \tag{2.5}$$

where $e_{uv} = 1$ if $\{u, v\} \in E$, $e_{uv} = 0$ otherwise.

Consequently, for any $u \in A$ and $v \in B$, the following formulation can be concluded:

$$\Delta_{uv} = \begin{cases} MaxOutS - MinInS - 1, & \text{if } \{u, v\} \in E \\ MaxOutS - MinInS, & \text{otherwise.} \end{cases}$$

2.1.2.3 Move selection strategy

Obviously, the moves with $\Delta_{uv} = MaxOutS - MinInS$ are preferable since they give improvement of the evaluation function f mostly. Let T denote those swap moves with the increment value equal to $MaxOutS - MinInS$.

$$T = \{(u, v) : u \in A, v \in B, \{u, v\} \notin E, \Delta_{uv} = MaxOutS - MinInS\}$$

We apply the following strategy to determine the best neighbor solution. If T is not empty, then one pair (u, v) from T is randomly selected for swap. If T is empty, vertex u is randomly selected from A and v is randomly selected from B. Notice that in this latter case, u and v must be two adjacent vertices.

It can be easily showed that the solution $S' = S \backslash \{u\} \cup \{v\}$ obtained by swapping such a pair of vertices (u, v) is one of the best non-tabu neighbor solutions in the neighborhood $CN(S)$, i.e., for any solution $S'' \in CN(S)$, $f(S') \geq f(S'')$. In fact, if $T = \emptyset$, then for each $S'' \in CN(S)$, $f(S'') = f(S) + MaxOutS - MinInS - 1$, i.e., any solution in $CN(S)$ has the same f value and S' is among the best non-tabu solutions. If $T \neq \emptyset$, then $f(S') = f(S) + MaxOutS - MinInS$. For any other solution $S'' \in CN(S)$ (assume that $S'' = S \oplus swap(x, y)$), $f(S'') = f(S) + MaxOutS - MinInS - e_{xy} \leq f(S) + MaxOutS - MinInS = f(S')$. Once again, we can see that S' is one of the best solutions in $CN(S)$.

Finally, to prepare the next iteration of the algorithm, d, A, B, $MinInS$ and $MaxOutS$ are updated accordingly after each $swap(u, v)$ move.

2.1.2.4 Probabilistic diversifying move selection rule

The above move selection rule assures an exhaustive exploration of the constrained neighborhood. To encourage the search to visit new regions in the search space, we additionally employ a strategy that disables the usual move selection rule and prefers occasionally some deteriorating moves. Such an alternative strategy is triggered only in a controlled and probabilistic manner when the current solution S corresponds to a local optimum, i.e., for each allowed $swap(u, v)$, the new solution $S' = S\backslash\{u\} \cup \{v\}$ is not better than the current solution S ($f(S') \leq f(S)$). In this case, we apply the following Probabilistic Move Selection Rule (PMSR).

- With a low probability $P = min\{\frac{l+2}{|V|}, 0.1\}$ where $|V|$ is the order of the graph and $l = k * (k-1)/2 - f(S)$, select a (much worse) $swap(u, v)$ as follows. Pick u at random from S and pick v in $V\backslash S$ such that $d(v) < \lfloor k * \rho \rfloor$, where ρ is the density of the graph.
- With probability 1-P, select one best allowed $swap(u, v)$ according to the usual selection strategy defined in Section 2.1.2.3.

This strategy provides a way to allow the search to occasionally go to another region when no better solution can be found around the current solution.

2.1.2.5 Tabu list and tenure management

To define our tabu list, first recall that a neighbor solution of S is characterized by a pair of (u, v) where u is a specific vertex in $A \subset S$ and v outside S. To prevent the search from revisiting S, when a $swap(u, v)$ move is performed, vertex u is added in a data structure called tabu list and remains in the list for the next T_u iterations (called tabu tenure). We call vertex u tabu and forbid the search to add u back to a solution during the period fixed by T_u. Similarly, vertex v is marked tabu for the next T_v iterations, during which v cannot be removed from the solution. We call a $swap(u, v)$ move tabu if at least one of the two implied vertices is marked tabu.

Inspired by the tabu mechanism proposed in [Galinier and Hao, 1999], the tabu tenures T_u and T_v are dynamically adjusted by a function depending on the evaluation function $f(S)$. More precisely, let $l_1 = k * (k-1)/2 - f(S)$, $l = min\{l_1, 10\}$. Then, T_u and T_v are defined respectively as follows.

$T_u = l + Random(C)$ and

$T_v = 0.6 * l + Random(0.6 * C)$

where $C = max\{\lfloor k/40 \rfloor, 6\}$ are two parameters and the function $Random(X)$ returns randomly an integer number in $\{0, \dots, X-1\}$. It is clear that $T_u > T_v$ holds. (As noted in [Glover, 1989], a tabu tenure to prevent elements from being dropped should typically be smaller than one to prevent elements from being added.)

The first part of the tabu tenure of T_u can be explained by the fact that a solution with a small evaluation function value should have a longer tabu tenure to escape from the local optimum trap. Since the exact value of the tabu tenure is unknown, the second part of T_u and T_u provides a random adjustment.

The reason for $T_u > T_v$ is that preventing vertices in the current solution S from being removed is much more restrictive than preventing vertices outside S from being added to S, since in general there are much fewer vertices contained in S than those outside S. In addition, preventing vertices added to S from being removed for a relatively long time can significantly inhibit available choices. Hence the tenure for the added vertex v should be made smaller by comparison to the removed vertex u.

In order to implement the tabu list, a vector *tabu_list* of $|V|$ elements is used. As suggested in [Glover, 1989], each element $tabu\_list(i)$ $(1 \leq i \leq |V|)$ records $T_i + Iter$, where *Iter* is the current number of iterations and T_i is the tabu tenure for vertex i. In this way, it is very easy to know if a vertex i is tabu or not at iteration j: if $tabu\_list(i) > j$, vertex i is forbidden to move; otherwise, i can be moved without restriction.

Finally, at each iteration, the tabu status of a move is canceled if the move leads to a better solution than the best solution S^* encountered so far.

2.1.3 A frequency-based strategy for new solution generation

To encourage the AMTS algorithm to explore new regions in the search space, we repeat the tabu search procedure TS^0 from different starting points. Recall that a restart is triggered when TS^0 cannot find an improved solution during L consecutive iterations (Section 2.1.2.1).

To build a new initial solution for each TS^0 restart, we devise an informed procedure guided by a long-term frequency memory. In this memory, we keep track of the number of times a vertex has been moved during the search. To maintain the frequency g_i of vertex i, we use the following rules.

1. Initially, set $g_i = 0$ for each vertex $i \in V$.

2. Subsequently, during the search, each time vertex i is removed from or put into the current solution S, the frequency counter g_i of vertex i is incremented, $g_i = g_i + 1$.

3. If for all $i \in V$, $g_i > k$, then we reset $g_i = 0$ for all $i \in V$. This mechanism refreshes the memory over time and avoids the situation where a vertex is definitively prevented from being selected by the solution construction procedure (see below).

Given this frequency information, we create the new initial solution S for a restart as follows. Initialize S by randomly adding a vertex having the smallest frequency value in V and then repeat the above step until S contains exactly k vertices. For each step, select a vertex $v \in V \backslash S$ such that v has the maximum number of edges that connect to S. If several vertices satisfy the above criterion, select the vertex with the smallest frequency value (less moved). If there are still several vertices that satisfy the two criteria, select one of these vertices randomly.

Notice that if ATMS is given a maximum of allowed $Iter_{max}$ iterations, ATMS may perform at most $Iter_{max}/L$ restarts during its run. A small (respectively large) L value implies more (respectively less) restart of the TS^0 procedure. We show a study of the influence of L on the performance of the AMTS algorithm.

2.2 Experimental results

To evaluate the efficiency of our AMTS algorithm, we carry out extensive experiments on the whole set of 80 DIMACS challenge benchmarks (see Section 1.6) and compare AMTS with five state-of-the-art maximum clique algorithms from the literature. Our AMTS algorithm is programmed in C, and compiled using GNU GCC on a PC with 2.83 GHz CPU and 8G RAM.

2.2.1 Experimental settings

We report our computational results based on the parameters values given here, even though fine-tuning the parameters would lead to improved results.

Parameter setting. The two main parameters for AMTS are the number of allowed iterations ($Iter_{max}$) for each run and the search depth L of TS^0 (see Section 2.1.3). Since AMTS stops when a legal k-clique is found, $Iter_{max}$ can safely given a very large value. In this paper, we use $Iter_{max} = 10^8$ as in [Pullan and Hoos, 2006] for their DLS-MC algorithm which is our main reference algorithm. Notice that for many graphs, AMTS attains a legal k-clique with much fewer iterations and stops long before reaching 10^8 iterations.

As to the search depth L, it is set equal to $|V| * k$ except for the structured brock and san graphs for which smaller values $4 * k$ are used. As a general rule, it is preferable to restart more frequently AMTS for structured graphs (by using a small L) in contrast to random graphs for which L should be set to a larger value.

Finally, since a maximum clique in a graph G is a maximum independent set in the complementary graph $\overline{G}$, when the density of G is greater than 0.5, it is transformed to its complement and AMTS is employed to solve the related maximum independent set problem.

2.2.2 Computational results

Given the stochastic nature of our AMTS algorithm, we run the algorithm 100 times on each DIMACS benchmark instance with different random seeds. To run AMTS on a graph, we set k to be the largest known clique size reported in the literature. During an AMTS run, legal cliques of size $k-1$ and $k-2$ are also recorded. These $k-1$ and $k-2$ cliques are reported if no k-clique is found for at least one of the 100 AMTS runs.

Table 2.1: The results obtained by AMTS on the set of 80 DIMACS benchmarks.

Instance	Node	ω	Quality	AvgSize	AvgTime	Iter/sec	TotalTime
brock200_1	200	21*	100-0-0	21	0.0136	280013	13.6
brock200_2	200	12*	100-0-0	12	0.3625	270770	36.25
brock200_3	200	15*	100-0-0	15	0.0105	272734	1.05
brock200_4	200	17*	100-0-0	17	1.7582	272728	175.82
brock400_1	400	27*	100-0-0	27	37.7739	187507	3777.39
brock400_2	400	29*	100-0-0	29	1.1818	187515	118.18
brock400_3	400	31*	100-0-0	31	1.7909	157902	179.09
brock400_4	400	33*	100-0-0	33	0.5956	146373	59.56
brock800_1	800	23*	98-0-2	22.96	234.6277	85714	25326.85
brock800_2	800	24*	100-0-0	24	33.1439	85649	3314.39
brock800_3	800	25*	100-0-0	25	52.3981	78950	5239.81
brock800_4	800	26*	100-0-0	26	15.2340	70768	1523.40
C125.9	125	34*	100-0-0	34	0.0018	400214	0.18
C250.9	250	44*	100-0-0	44	0.0058	336700	0.58
C500.9	500	57	100-0-0	57	0.1263	206611	12.63
C1000.9	1000	68	100-0-0	68	1.1471	181180	114.71

Table 2.1 – continued from previous page

Instance	Node	ω	Quality	AvgSize	AvgTime	Iter/sec	TotalTime
C2000.5	2000	16	100-0-0	16	0.6611	31685	66.11
C2000.9	2000	80	1-93-6	78.95	450.0996	86199	115300.62
C4000.5	4000	18	100-0-0	18	126.6315	15422	12663.15
DSJC500.5	500	13*	100-0-0	13	0.0071	106723	0.71
DSJC1000.5	1000	15*	100-0-0	15	0.3113	59241	31.13
keller4	171	11*	100-0-0	11	< 0.0001	212000	0.01
keller5	776	27	100-0-0	27	0.0565	120772	5.65
keller6	3361	59	100-0-0	59	10.8103	47755	1081.03
MANN_a9	45	16*	100-0-0	16	0.0161	835681	1.61
MANN_a27	378	126*	100-0-0	126	0.0707	715188	7.07
MANN_a45	1035	345*	4-96-0	344.04	112.8498	436381	22450.52
MANN_a81	3321	1100	0-0-100	1098	27.5524	332219	2755.24
hamming6-2	64	32*	100-0-0	32	< 0.0001	581395	0.01
hamming6-4	64	4*	100-0-0	4	< 0.0001	245700	0.01
hamming8-2	256	128*	100-0-0	128	0.0005	236966	0.05
hamming8-4	256	16*	100-0-0	16	< 0.0001	177935	0.01
hamming10-2	1024	512*	100-0-0	512	0.3116	71123	31.16
hamming10-4	1024	40	100-0-0	40	0.9167	130548	91.67
gen200_p0.9_44	200	44*	100-0-0	44	0.0074	375939	0.74
gen200_p0.9_55	200	55*	100-0-0	55	0.0006	531914	0.06
gen400_p0.9_55	400	55	100-0-0	55	0.5476	211914	54.76
gen400_p0.9_65	400	65	100-0-0	65	0.0123	355871	1.23
gen400_p0.9_75	400	75	100-0-0	75	0.0415	200512	4.15
c-fat200-1	200	12*	100-0-0	12	0.0014	108675	0.14
c-fat200-2	200	24*	100-0-0	24	0.1742	91407	17.42
c-fat200-5	200	58*	100-0-0	58	0.1102	87719	11.02
c-fat500-1	500	14*	100-0-0	14	0.1354	47755	13.54
c-fat500-2	500	26*	100-0-0	26	0.2253	44150	22.53
c-fat500-5	500	64*	100-0-0	64	0.1009	39510	10.09
c-fat500-10	500	126*	100-0-0	126	2.6587	29629	265.87
johnson8-2-4	28	4*	100-0-0	4	< 0.0001	375939	0.01
johnson8-4-4	70	14*	100-0-0	14	< 0.0001	425531	0.01
johnson16-2-4	120	8*	100-0-0	8	< 0.0001	96993	0.01
johnson32-2-4	496	16*	100-0-0	16	< 0.0001	22857	0.01
p_hat300-1	300	8*	100-0-0	8	0.0008	130548	0.08
p_hat300-2	300	25*	100-0-0	25	0.0007	220750	0.07
p_hat300-3	300	36*	100-0-0	36	0.0016	255754	0.16
p_hat500-1	500	9*	100-0-0	9	0.0011	84175	0.11
p_hat500-2	500	36*	100-0-0	36	0.0008	165213	0.08
p_hat500-3	500	50	100-0-0	50	0.0053	284419	0.53
p_hat700-1	700	11*	100-0-0	11	0.0098	60518	0.98
p_hat700-2	700	44*	100-0-0	44	0.0012	155470	0.12
p_hat700-3	700	62	100-0-0	62	0.0053	233798	0.53
p_hat1000-1	1000	10	100-0-0	10	0.0008	45202	0.08
p_hat1000-2	1000	46	100-0-0	46	0.0009	105470	0.09
p_hat1000-3	1000	68	100-0-0	68	0.0813	200348	8.13
p_hat1500-1	1500	12*	100-0-0	12	2.1815	31628	218.15
p_hat1500-2	1500	65	100-0-0	65	0.3284	80123	32.84
p_hat1500-3	1500	94	100-0-0	94	0.3153	139885	31.53
san200_0.7_1	200	30*	100-0-0	30	0.2074	100102	20.74
san200_0.7_2	200	18*	100-0-0	18	0.2420	88909	24.20
san200_0.9_1	200	70*	100-0-0	70	0.1676	170024	16.76
san200_0.9_2	200	60*	100-0-0	60	0.1322	300293	13.22
san200_0.9_3	200	44*	100-0-0	44	0.0757	300263	7.57
san400_0.5_1	400	13*	100-0-0	13	11.4577	33336	1145.77
san400_0.7_1	400	40*	100-0-0	40	8.7633	40032	876.33
san400_0.7_2	400	30*	100-0-0	30	29.9791	42873	2997.91
san400_0.7_3	400	22*	100-0-0	22	56.2885	45024	5628.85
san400_0.9_1	400	100*	100-0-0	100	1.8674	42888	186.74

Table 2.1 – continued from previous page

Instance	Node	ω	Quality	AvgSize	AvgTime	Iter/sec	TotalTime
san1000	1000	15*	100-0-0	15	315.1698	37273	31516.98
sanr200-0.7	200	18*	100-0-0	18	0.0009	290697	0.09
sanr200-0.9	200	42*	100-0-0	42	0.0047	336700	0.47
sanr400-0.5	400	13*	100-0-0	13	0.0137	130548	1.37
sanr400-0.7	400	21	100-0-0	21	0.0048	182815	0.48

To report our results, we follow the common practice of the literature on the maximum clique problem such as ([Pullan, 2006; Katayama *et al.*, 2005; Singh and Gupta, 2006; Battiti and Protasi, 2001]). Table 2.1 gives the computational statistics.

For each instance, we show in column 4 the solution quality by a triple $a - b - c$, where a is the number of runs (out of the 100 runs) in which a clique size of ω (ω is the maximum known clique size reported in the literature) is found, b is the number of runs in which the algorithm fails to find a clique size of ω, but attains a clique size of $\omega - 1$, c is the number of runs where only cliques of size $\omega - 2$ or worse are found. The next three columns provide other information: the averaged clique size over 100 runs, averaged CPU time in seconds over the successful runs and the average iterations per second. The last column indicates the *total run time of the 100 runs* of AMTS to solve an instance. Notice that it is generally much shorter than the time needed to execute $100 \times Iter_{max} = 10^8$ because very often AMTS finds a legal k-clique long before attaining $Iter_{max}$ iterations.

Table 2.1 discloses that AMTS can find cliques of the largest known size for 79 out of the 80 benchmarks. The only instance for which AMTS fails to find the best known solution ($\omega = 1100$) is MANN_a81. For this instance, AMTS obtains consistently cliques of size 1098 in 100 of all the 100 runs. (The average time provided in Table 2.1 for the instance MANN_a81 is the average time to find cliques size of 1098.)

Of the 79 instances where AMTS attains the best known solutions, in 76 cases it finds such a solution with a success rate of 100%. For only three instances (brock800_1, C2000.9, MANN_a45), the success rate is less than 100%. For C2000.9 which has a largest known clique size of $\omega = 80$, the success rate is only 1%, but AMTS obtains consistently cliques of size 79 in 93 of 100 runs, while the remaining 6 runs finds cliques of size 78 (see Table 2.2). To the best of our knowledge, cliques of size 80 for C2000.9 have only been reported recently in [Grosso *et al.*, 2008]. Not only AMTS attains this result, but also it can easily attain cliques of size 79 in reasonable time. Table 2.2 displays additionnal results of AMTS on C2000.9, statistics are based on 100 independent runs. Table 2.1 shows that for 58 out of the 80 DIMACS instances, the average CPU time for attaining the best known solution is within 1 CPU second. An average CPU time of 10 or more CPU seconds are required for 15 of the 22 remaining instances.

2.2.3 Comparative results

In this section, we attempt to compare AMTS with 5 representative state-of-the-art methods from the literature. The main comparison criterion is the quality of the solutions found.

Table 2.2: The performance of AMTS on the C2000.9 instance.

clique size(k)	AvgTime	AvgIter	success rate
80	450.09	38797622	1
79	338.39	29169205	93
78	33.52	2890191	100

Due to the differences among the programming languages, data structures, compiler options and computers, computing times are provided only for indicative purposes.

First, we recall the hardware and basic experimental conditions used by these reference methods.

- DLS-MC (Stochastic local search ([Pullan, 2006])). The results of DLS-MC were based on a dedicated 2.2 GHz Pentium IV machine with 512KB L2 cache and 512MB RAM. For each instance, DLS-MC was run 100 times, each run being allowed 10^8 iterations like in our case.

- KLS (k-opt variable depth search algorithm ([Katayama *et al.*, 2005])). The results of of KLS were based on a Sun Blade 1000 Workstation (UltraSPARC-III 900 MHz, 2 GB memory). For each instance, KLS was run 100 trials. For each trial, KLS was repeatedly executed n times, where n was the number of nodes of a given graph.

- HSSGA (Heuristic based steady-state genetic algorithm ([Singh and Gupta, 2006])). HSSGA was run on a Pentium-III 1GHz Linux based system with 384 MB RAM. HSSGA was run 10 times on each graph instance. For each run, HSSGA was run until either the optimum solution value was found or a maximum of 20 000 generations was reached.

- RLS (Reactive local search ([Battiti and Protasi, 2001])). RLS was run on a Pentium-II (450MHz CPU, 384MB RAM) machine. For each instance, 100 runs were performed, for each run, the number of iterations was fixed to $20000 \times n$.

- QUALEX-MS (Quick Almost Exact Motzkin-Straus-based search ([Busygin, 2006])). QUALEX-MS was run on a Pentium IV 1.4GHz computer under Red Hat Linux.

In Table 2.3, we first compare our AMTS method with DLS-MC which is the current best maximum clique algorithm. The comparison mainly focuses on solution quality, i.e., the largest clique size found (averaged size is given in parenthesis if it is different from the largest one). As explained above, computing times are provided only as complementary information. Notice moreover that the results of DLS-MC were obtained after fine-tuning its parameter ([Pullan, 2006]) on an instance-by-instance basis.

Table 2.3 shows that AMTS compares favorably with DLS-MC in terms of the best clique size. Indeed, AMTS can find the largest clique sizes for all the 80 instances except MANN_a81 while DLS-MC can find the best known solutions for all the instances except MANN_a81, MANN_a45 and C2000.9. The difference between AMTS and DLS-MC can

Table 2.3: Comparative results between AMTS and DLS-MC.

	AMTS		DLS-MC			AMTS		DLS-MC	
Instance	Clique size	CPU(s)	Clique size	CPU(s)	Instance	Clique size	CPU(s)	Clique size	CPU(s)
brock200_1	21	0.0136	21	0.0182	johnson32_2_4	16	$< \epsilon$	16	$< \epsilon$
brock200_2	12	0.3625	12	0.0242	johnson8_2_4	4	$< \epsilon$	4	$< \epsilon$
brock200_3	15	0.0105	15	0.0367	johnson8_4_4	14	$< \epsilon$	14	$< \epsilon$
brock200_4	17	1.7582	17	0.0468	keller4	11	$< \epsilon$	11	$< \epsilon$
brock400_1	27	37.774	27	2.2299	keller5	27	0.0565	27	0.0201
brock400_2	29	1.1818	29	0.4774	keller6	59	10.810	59	170.483
brock400_3	31	1.7909	31	0.1758	MANN_a9	16	0.0161	16	$< \epsilon$
brock400_4	33	0.5956	33	0.0673	MANN_a27	126	0.0707	126	0.0476
brock800_1	23(22.96)	234.628	23	56.497	MANN_a45	345(344.04)	112.850	344	51.960
brock800_2	24	33.144	24	15.734	MANN_a81	1098	27.552	1098(1097.96)	264.009
brock800_3	25	52.398	25	21.920	p_hat300_1	8	0.0008	8	0.0007
brock800_4	26	15.234	26	8.8807	p_hat300_2	25	0.0007	25	0.0002
C125.9	34	0.0018	34	$< \epsilon$	p_hat300_3	36	0.0016	36	0.0007
C250.9	44	0.0058	44	0.0009	p_hat500_1	9	0.0011	9	0.0010
C500.9	57	0.1263	57	0.1272	p_hat500_2	36	0.0008	36	0.0005
C1000.9	68	17.547	68	4.440	p_hat500_3	50	0.0053	50	0.0023
C2000.5	16	0.6611	16	0.9697	p_hat700_1	11	0.0098	11	0.0194
C2000.9	80(78.95)	450.100	78(77.93)	193.224	p_hat700_2	44	0.0012	44	0.0010
C4000.5	18	126.632	18	181.234	p_hat700_3	62	0.0053	62	0.0015
DSJC500.5	13	0.0071	13	0.0138	p_hat1000_1	10	0.0008	10	0.0034
DSJC1000.5	15	0.3113	15	0.7990	p_hat1000_2	46	0.0009	46	0.0024
c-fat200-1	12	0.0014	12	0.0002	p_hat1000_3	68	0.0813	68	0.0062
c-fat200-2	24	0.1742	24	0.0010	p_hat1500_1	12	2.1815	12	2.7064
c-fat200-5	58	0.1102	58	0.0002	p_hat1500_2	65	0.3284	65	0.0061
c-fat500-1	14	0.1354	14	0.0004	p_hat1500_3	94	0.3153	94	0.0103
c-fat500-2	26	0.2253	26	0.0004	san200_0.7_1	30	0.2074	30	0.0029
c-fat500-5	64	0.1009	64	0.0020	san200_0.7_2	18	0.2420	18	0.0684
c-fat500-10	126	2.6587	126	0.0015	san200_0.9_1	70	0.1676	70	0.0003
gen200-P0.9-44	44	0.0074	44	0.0010	san200_0.9_2	60	0.1322	60	0.0002
gen200-P0.9-55	55	0.0006	55	0.0003	san200_0.9_3	44	0.0757	44	0.0015
gen400-P0.9-55	55	0.5476	55	0.0268	san400_0.5_1	13	11.458	13	0.1641
gen400-P0.9-65	65	0.0123	65	0.0010	san400_0.7_1	40	8.7366	40	0.1088
gen400-P0.9-75	75	0.0415	75	0.0005	san400_0.7_2	30	29.979	30	0.2111
hamming6-2	32	$< \epsilon$	32	$< \epsilon$	san400_0.7_3	22	56.289	22	0.4249
hamming6-4	4	$< \epsilon$	4	$< \epsilon$	san400_0.9_1	100	1.8674	100	0.0029
hamming8-2	128	0.0005	128	0.0003	san1000	15	315.170	15	8.3636
hamming8-4	16	$< \epsilon$	16	$< \epsilon$	sanr200_0.7	18	0.0009	18	0.0020
hamming10-2	512	0.3116	512	0.0008	sanr200_0.9	42	0.0047	42	0.0127
hamming10-4	40	0.9167	40	0.0089	sanr400_0.5	13	0.0137	13	0.0393
johnson16-2-4	8	$< \epsilon$	8	$< \epsilon$	sanr400_0.7	21	0.0048	21	0.0230

Table 2.4: Comparative results of AMTS with four other leading clique algorithms.

Instance	Node	Best	Max-Clique Algorithm									
			AMTS		KLS		HSSGA		RLS		QUALEX-MS	
			size	time	size	time	size	time	size	time	size	time
brock200_2	200	12*	12	0.3625	11	0.0035	12	0.29	12	9.605	12	< 1
brock200_4	200	17*	17	1.7582	16	0.0066	17(16.7)	1.14	17	19.491	17	< 1
brock400_2	400	29*	29	1.1818	25(24.84)	0.1334	29(25.1)	2.35	29(26.063)	42.091	29	3
brock400_4	400	33*	33	0.5956	25	0.0174	33(27.0)	2.76	33(32.423)	108.638	33	2
brock800_2	800	24*	24	33.144	21(20.86)	0.4993	21(20.7)	10.72	21	4.739	24	18
brock800_4	800	26*	26	15.234	21(20.67)	1.2160	21(20.1)	3.04	21	6.696	26	18
C125.9	125	34*	34	0.0018	34	0.0011	34	0.06	34	0.004	34	< 1
C250.9	250	44*	44	0.0058	44	0.0278	44(43.8)	0.34	44	0.029	44	1
C500.9	500	57	57	0.1263	57(56.15)	0.2699	56(54.2)	4.17	57	3.124	55	4
C1000.9	1000	68	68	1.1471	68(66.38)	2.0049	66(64.1)	14.27	68	41.660	64	27
C2000.5	2000	16	16	0.6611	16	2.8971	16(15.4)	27.52	16	9.976	16	278
C2000.9	2000	80	80(78.95)	450.10	77(74.90)	14.715	74(71.0)	117.66	78(77.575)	823.358	72	215
C4000.5	4000	18	18	126.63	18(17.02)	23.802	17(16.8)	158.42	18	2183.089	17	2345
DSJC500.5	500	13*	13	0.0071	13	0.0256	13	0.71	13	0.194	13	5
DSJC1000.5	1000	15*	15	0.3113	15(14.93)	0.6711	15(14.7)	7.38	15	6.453	14	36
keller4	171	11*	11	0.0001	11	0.0003	11	0.01	11	0.002	11	1
keller5	776	27	27	0.0565	27	0.0399	27(26.9)	4.04	27	0.171	26	16
keller6	3361	59	59	10.810	57(55.59)	52.364	57(54.2)	314.65	59	189.814	53	1291
MANN_a27	378	126*	126	0.0707	126	0.0178	126(125.5)	3.17	126	3.116	125	1
MANN_a45	1035	345*	345(344.04)	112.85	345(343.88)	6.2014	343(342.6)	65.25	345(343.602)	398.770	342	17
MANN_a81	3321	1100	1098	27.552	1100(1098.07)	39.484	1095(1094.2)	3996.65	1098	2830.820	1096	477
hamming8-4	256	16*	16	0.0001	16	0.0004	16	0.01	16	0.003	16	1
hamming10-4	1024	40	40	0.9167	40	0.2209	40(39.0)	10.21	40	0.078	40	45
gen200_p0.9_44	200	44*	44	0.0074	44	0.0317	44(43.1)	1.07	44	0.037	42	< 1
gen200_p0.9_55	200	55*	55	0.0006	55	0.0065	55	0.29	55	0.016	55	1
gen400_p0.9_55	400	55	55	0.5476	53(52.21)	0.2089	53(51.4)	1.83	55	1.204	51	2
gen400_p0.9_65	400	65	65	0.0123	65	0.0647	65(63.8)	1.71	65	0.050	65	2
gen400_p0.9_75	400	75	75	0.0415	75	0.0425	75	1.93	75	0.051	75	2
p_hat300-1	300	8*	8	0.0008	8	0.0021	8	0.02	8	0.018	8	1
p_hat300-2	300	25*	25	0.0007	25	0.0012	25	0.02	25	0.006	25	1
p_hat300-3	300	36*	36	0.0016	36	0.0118	36(35.9)	0.18	36	0.021	35	1
p_hat700-1	700	11*	11	0.0098	11	0.1245	11	1.02	11	0.186	11	10
p_hat700-2	700	44*	44	0.0012	44	0.0077	44	0.19	44	0.028	44	12
p_hat700-3	700	62	62	0.0053	62	0.0158	62(61.7)	2.01	62	0.035	62	11
p_hat1500-1	1500	12*	12	2.1815	12	2.6054	12(11.5)	14.62	12	30.274	12	95
p_hat1500-2	1500	65	65	0.3284	65	0.0625	65(64.9)	2.03	65	0.158	64	111
p_hat1500-3	1500	94	94	0.3153	94	0.4286	94(93.1)	2.91	94	0.192	91	108

also be observed in terms of the average clique size obtained by the two algorithms; AMTS finds larger average clique size on three large and hard instances (C2000.9, MANN_a45 and MANN_a81) while the result of DLS-MC is better for one instance (brook800_1).

In terms of solution speed, DLS-MC shows better performance than AMTS on the brock and san graphs. Indeed, for all the instances of these two families, both algorithms can attain the best known solutions, but DLS-MC needs much less computing time.

In Table 2.4, we report the best and the average clique size obtained by AMTS in comparison with the other four algorithms (KLS, HSSGA, RLS and QUALEX-MS) on 37 DIMACS benchmark instances which are used by these reference algorithms. Table 2.5 summarizes the comparative results in terms of the number of instances on which these algorithms performs better or worse than AMTS.

Tables 2.4 and 2.5 show that AMTS finds larger cliques than KLS for 9 graphs, while

Table 2.5: Comparison result of AMTS with KLS, HSSGA, RLS and QUALEX-MS.

	Best clique size		Average clique size	
	Better than AMTS	Worse than AMTS	Better than AMTS	Worse than AMTS
KLS	1	9	1	14
HSSGA	0	10	0	26
RLS	0	3	0	6
QUALEX-MS	0	14	-	-

Table 2.5: Comparison result of AMTS with KLS, HSSGA, RLS and QUALEX-MS in terms of number of instances on which AMTS found better (or worse) results out of the 37 DIMACS benchmark instances. The symbol '-' used for QUALEX-MS indicates that the average clique size is not available.

the reverse is true only for one graph. Moreover, the average clique size found by AMTS is better than that of KLS on 14 instances whereas KLS outperforms AMTS on one instance. Regarding the other three algorithms (HSSGA, RLS and QUALEX-MS), AMTS can find an equal or better solution than these reference algorithms on each of the 37 benchmark instances.

2.3 Analysis of critical components of AMTS

2.3.1 Influence of restart

Recall that for each run of the algorithm, ATMS restarts from a new solution if the current solution is not improved for L consecutive iterations. So a small (large) value of L leads to more (less) frequent restart. To analyze the influence of the restart strategy on the performance of the AMTS algorithm, we focus on the effect of L and study the running profile of the evaluation function f (Formula 2.2, Section 2.1.2.1) by varying the value of L.

Experiments in this study are performed on a *structured* instance (brock800_2) and a *random* instance (C2000.9). To solve these instances, we consider 3 different values $L =$ 100, 1000 and 10000. For each of these values, we perform 100 runs of AMTS, each run being given a maximum of $Iter_{max} = 10^7$ iterations.

The running profile is defined by the function $i \longmapsto f_*(i)$ where i is the number of iterations (counter $Iter$) and $f_*(i)$ is the best evaluation function value known at iteration i, averaged over 100 runs. More precisely, let j denote the jth run of AMTS ($j = 1...100$), $f_i^j(S^*)$ the value of the evaluation function f (defined by Formula 2.2 in Section 2.1.2.1) of the best solution S^* known at iteration i of AMTS jth run. For each plotted iteration i in the running profile, $f_*(i)$ is equal to $\sum_{j=1}^{100} f_i^j(S^*)/100$. Such a profile gives a natural way to observe the evolution of the best values of the objective function during a search ([Galinier and Hao, 1999]).

Figure 2.1 shows the running profiles of AMTS on the graph brock800_2 with $k = 24$. The figure shows also the running profile of AMTS without restart, i.e., with $L = Iter_{max}$ (we call this version basic TS). From Figure 2.1, we observe that AMTS with $L = 100$ dominates AMTS with $L = 1000$ and $L = 10000$. Since smaller L implies more restarts, this experiment suggests a frequent restart is quite useful for the instance brock800_2 (in fact for other special structured instances). One also notices that AMTS without restart performs the worst.

Figure 2.2 shows the running profiles of AMTS on C2000.9 with $k = 79$. It is interesting to observe that for this graph, AMTS performs better with large values $L = 1000$ or $L = 10000$ than with $L = 100$. AMTS without restart performs here quite well. This suggests that for C2000.9 (in fact for many random instances) a long search with the basic TS^0 engine is more effective than a search with frequent restarts.

The above observations are confirmed by the results reported in Table 2.6. In this

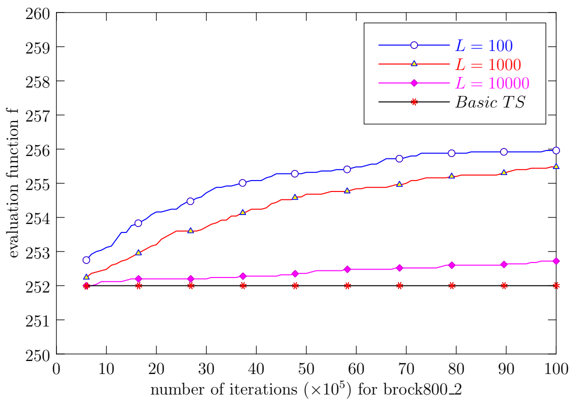


Figure 2.1: Running profile of AMTS with $L = 100$, 1000 and 10000 as well as AMTS without restart (basic TS) on brock800_2.

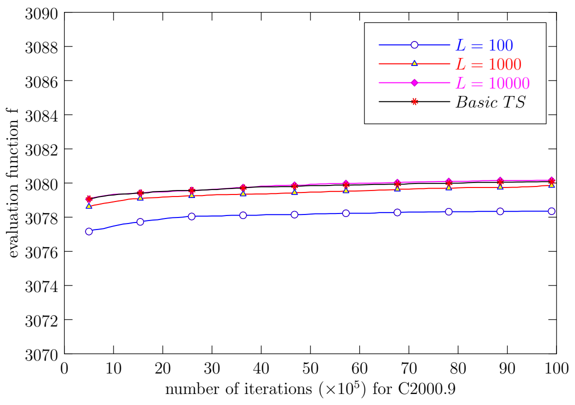


Figure 2.2: Running profile of AMTS with $L = 100$, 1000 and 10000 as well as AMTS without restart (basic TS) on C2000.9.

Table 2.6: Success rate of ATMS with different values of L and AMTS without restart (basic TS) for brock800_2 and C2000.9.

Graph	L=100	L=1000	L=10000	Basic TS
brock800_2	99	87	18	0
C2000.9	0	6	19	17

table, we show the number of runs (out of the 100 runs) where a clique size of k is found successfully by AMTS with these L values and ATMS without restart. For brock800_2, search with frequent restarts makes ATMS more effective and robust whereas the reverse is true for C2000.9.

More generally, various experiments suggest that for some structured graphs, relatively smaller L values are preferable whereas for random graphs, it is advantageous to use relatively larger L values. This experiment also explains the choice of the L values used in Section 5.4.2. In sum, compared to the $Iter_{max}$ parameter, L is more sensitive to the structure of the graph and should be tuned with more care.

2.3.2 The tabu list

As explained in Section 2.1.2.5, each time a $swap(u, v)$ move is performed, both the dropped vertex u and the added vertex v are marked tabu for respectively T_u and T_v iterations. We experiment here two additional tabu strategies which are summarized together with the previous one as follows.

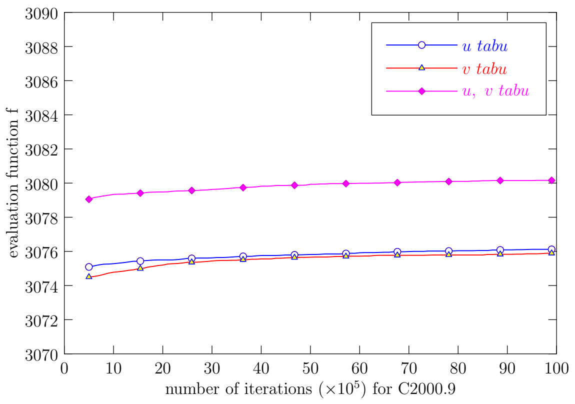


Figure 2.3: The basic TS with three tabu strategies.

- *Strategy* 1: Only preventing the dropped vertex u from being put back into S in the next T_u iterations.
- *Strategy* 2: Only preventing the added vertex v from being removed from S in the next T_v iterations.

- *Strategy* 3: Preventing u from being put back into S in the next T_u iterations while preventing v from being removed from S in the next T_v iterations. This strategy is used in this paper.

We test these three strategies on C2000.9 with $k = 79$. Figure 2.3 shows the running profiles. From the figure, we can observe that strategy 3, which is used by our proposed AMTS algorithm, largely dominates strategy 1 and strategy 2 throughout the search.

2.4 Conclusions

In this chapter, we have presented AMTS, an adaptive multistart tabu search for approximating the maximum clique problem. AMTS seeks a clique of fixed size k by effectively exploring subsets of vertices of size k. For this purpose, AMTS combines a TS procedure with a guided restart strategy. The TS engine is based a constrained neighborhood and an adaptive technique for tuning the double tabu tenures. To enable an effective exploration of the search space, AMTS uses an informed multistart strategy which relies on a long term memory (move frequencies) to regenerate new initial starting solutions.

AMTS shows an excellent performance on the complete set of 80 standard DIMACS benchmark instances. AMTS finds the current best known solutions for all the instances except one case (MANN_a81 for which cliques of size 1098 are found easily). The competitiveness of AMTS is further confirmed when it is compared with five state-of-the-art maximum clique procedures.

Most of the current top-performing algorithms for the maximum clique problem are based on an expansion and plateau search model. The proposed method constitutes an interesting alternative approach that probably merits more attention and research efforts.

Finally, the AMTS algorithm has been applied very recently with success to solve two combinatorial problems: graph coloring [Wu and Hao, 2012a] and graph sum coloring [Wu and Hao, 2012b] (see also Chapter 5 and Chapter 7). In the next chapter, we will discuss one important generalization of the classic MCP when a positive weight is associated to each vertex in the graph: the maximum vertex weight clique problem.

Chapter 3

Multi-neighborhood tabu search for the maximum vertex weight clique problem

In this chapter, we consider the maximum vertex weight clique problem (MVWCP) which is one important generalization of the MCP. Given an undirected graph $G = (V, E)$ with vertex set $V = \{1, ..., n\}$ and edge set $E \subseteq V \times V$. Let $w : V \rightarrow Z^+$ be a weighting function that assigns to each vertex $i \in V$ a positive integer. The MVWCP is to determine a clique of maximum weight. This chapter introduces a tabu search heuristic whose key features include a combined neighborhood, a dedicated tabu mechanism and a randomized restart strategy. The proposed algorithm is evaluated on a total of 136 benchmark instances from different sources (DIMACS, BHOSLIB and set packing). Computational results disclose that our tabu search algorithm outperforms the leading algorithm for the MVWCP, and in addition rivals the performance of the best methods for the unweighted version of the problem without being specialized to exploit this problem class. The content of this chapter is published in [Wu and Hao, 2012f].

Contents

3.1 Introduction

An important generalization of MCP is the maximum vertex weight clique problem (MVWCP) where a positive weight is associated to each vertex. A formal definition of MVWCP is provided in Section 1.1. It is clear that the classical unweighted version (MCP) turns out to be a special case of MVWCP when the weight of each vertex is set equal to 1. For this reason, the MVWCP has at least the same computational complexity as the MCP. Applications of the MVWCP arise in a number of domains like computer vision, pattern recognition and robotics [Ballard and Brown, 1982].

To solve the MVWCP, a number of exact algorithms have been reported in the literature (see e.g., [Babel, 2008; Östergård, 2001]) which can be applied to instances of small sizes. For large problems, various heuristic methods have been proposed to find approximative solutions. For instance, in [Pullan, 2008], an efficient local search algorithm is presented which is based on the PLS algorithm (see Section 1.4.2.5) developed for the unweighted case [Pullan, 2006]. Along with the proposed algorithm, a set of MVWCP benchmark instances using the DIMACS graphs are also introduced (see Section 1.6, we use these instances in our experiments). In [Mannino and Stefanutti, 1999], an augmentation algorithm is described which is based on edge projections for the equivalent maximum weight stable set problem. Other representative studies include a deterministic iterated greedy construction algorithm using a nonlinear programming formulation [Busygin, 2006], and a distributed computational network algorithm [Bomze *et al.*, 2000].

In this chapter, we present a multi-neighborhood tabu search approach (denoted by MN/TS) for the MVWCP. In order to effectively explore the search space, the proposed algorithm combines three neighborhoods induced by three types of moves. The particularity of the combined neighborhood relies on the *union* of the underlying neighborhoods instead of the conventional sequential exploration of basic neighborhoods (Sections 3.2.3 and 3.2.4). At each iteration of the algorithm, our tabu search approach explores the union of these three neighborhoods and selects the overall best admissible neighboring solution. The algorithm integrates a dedicated tabu mechanism (Section 3.2.5) and a randomized restart strategy (Section 3.2.6).

The performance of the proposed MN/TS algorithm is assessed on a large set of benchmarks from the well-known DIMACS and BHOSLIB libraries and the set packing problem (Section 3.4). Extensive experimental tests disclose that the proposed approach finds new best solutions for 26 DIMACS instances of the MVWCP, while matching the best known solution on all but one of the others. For the unweighted case, MN/TS is able to attain the best known solutions for the 120 tested instances except for only two cases, rivaling the performance of the best algorithms for the MCP problem without specializing our method to exploit the unweighted class. For an additional set of 16 instances derived from the set packing problem, MN/TS is able to attain the current best-known results while discovering 2 improved results, again without being designed to exploit the set packing structure (in contrast to the methods that have produced the previous best results). An analysis is also provided to show the relevance of the union exploration of the underlying neighborhoods (Section 3.5).

3.2 Multi-neighborhood tabu search for the MVWCP

In this section, we present our multi-neighborhood tabu search (MN/TS) approach for the general MVWCP. Based on the legal strategy, MN/TS integrates several features which are responsible for its effectiveness, including three complementary neighborhoods defined by three basic move operators. These neighborhoods are explored in a *combined manner* employing a rule that selects the most favorable neighboring solution that is admissible subject to the tabu conditions. The method is driven by a dedicated tabu list strategy employing a restart mechanism for diversification.

3.2.1 Search space and evaluation function

For a given MVWCP instance $G = (V, E, w)$, our MN/TS algorithm considers the legal strategy (see Section 1.4.2.1), and explores a search space Ω composed of all possible cliques of G, i.e., $\Omega = \{C : C \subset V$ such that $\forall i, j \in C, i \neq j, \{i, j\} \in E\}$. For any solution $C \in \Omega$, its *quality* is evaluated by its weight $W(C) = \sum_{i \in C} w_i$. Given two solutions C and C', C' is better than C if and only if $W(C') > W(C)$. Our objective function (to be maximized) is thus given by: $W : \Omega \rightarrow Z^+$.

3.2.2 Randomized procedure for initial solutions

Our algorithm starts from an initial clique $C \in \Omega$ and then uses the tabu search procedure (Sections 3.2.3-3.2.5) to improve C by maximizing its weights. The initial solution C is constructed as follows. We first select randomly a seeding vertex i from the graph and set the current clique C to the set consisting of this single vertex. We then randomly pick another vertex $v \notin C$ subject to the stipulation that v is connected to *all* the vertices of C (i.e., v is taken from the set $\{v : v \in V \backslash C, \{v, i\} \in E, \forall i \in C\}$). This process is repeated until no such vertex v exists. This procedure is also used to initialize each restart during a run of the MN/TS algorithm (see Section 3.2.6). This procedure has the advantage of being simple and fast, leading to diversified initial solutions for each round of the tabu search procedure.

3.2.3 Basic move operators and neighborhoods

In local search, a neighborhood is typically defined by a move operator mv, which transforms a given solution C to generate a neighboring solution C', denoted by $C' = C \oplus mv$. Let $M(C)$ be the set of all possible moves which can be applied to C, then the neighborhood N of C is defined by: $N(C) = \{C' : C' = C \oplus mv, mv \in M(C)\}$.

Our MN/TS algorithm explores jointly three neighborhoods which are defined by three basic move operators (denoted by *ADD*, *SWAP* and *DROP*). These move operators are based on the definition of two vertex subsets: PA and OM relative to a given clique C.

PA is composed of the vertices that are excluded from the clique C and connected to *all* the vertices of C: $PA = \{v : v \in V \backslash C, \{v, i\} \in E, \forall i \in C\}$.

OM contains the vertices that are excluded from the clique C and connected to *all but one* vertex of C: $OM = \{v : v \in V \backslash C, |A(v) \bigcap C| = |C| - 1\}$ where $A(v) = \{j : j \in V, \{j, v\} \in E\}$ is the set of vertices adjacent to v.

The relationship between a clique C and the associated subsets PA and OM is illustrated in Fig. 3.1.

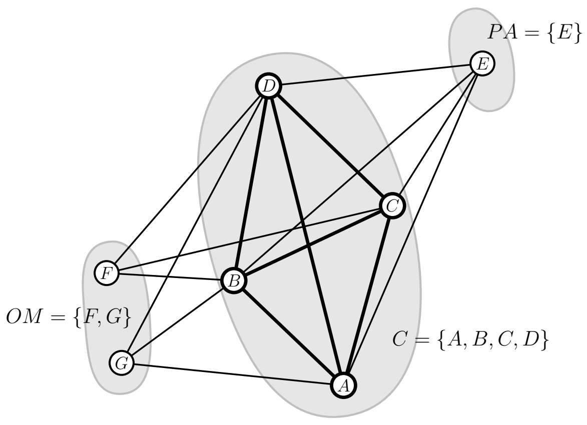


Figure 3.1: A clique and its two associated subsets: $C = \{1, 2, 3, 4\}$, $PA = \{5\}$ and $OM = \{6, 7\}$.

The two subsets PA and OM just described form the basis for defining the ADD and $SWAP$ move operators while the $DROP$ move operator is defined independently of these subsets, as follows.

- $ADD(i)$: This move operator (which applies when PA is not empty) consists in adding a vertex i from the set PA to the current clique C. The neighborhood defined by this move operator is given by $N_1 = \{C' : C \oplus ADD(i), i \in PA\}$.

  After a $ADD(i)$ move, the change in the clique weight (i.e., the move gain denoted by Δ_i) is given by the following expression:

  $$\Delta_i = w_i \tag{3.1}$$

  where w_i is the weight associated to vertex i. Since the move gain is always positive for a ADD move, such a move always leads to an improved neighboring solution. The size of this neighborhood is clearly bounded by $O(n)$.

- *SWAP(i,j)*: This move operator (which applies when OM is not empty) consists in exchanging a vertex i from the set OM with the only vertex j of C which is not connected to i in C. The neighborhood defined by this move operator is given by $N_2 = \{C' : C \oplus SWAP(i, j),\ i \in OM, j \in C, \{i, j\} \notin E\}$.

  For a given *SWAP(i,j)* move, the move gain Δ_{ij} can be conveniently computed by:

  $$\Delta_{ij} = w_i - w_j \tag{3.2}$$

Since Δ_{ij} can be either positive or negative, a *SWAP* move can improve or deteriorate the quality of the current solution. The size of this neighborhood is bounded by $O(n)$.

- *DROP(i)*: This move operator removes a vertex i from the current clique C. The neighborhood induced by the *DROP* move can be formally defined by $N_3 = \{C' : C\backslash\{i\}, i \in C\}$.

  The move gain Δ_i of dropping vertex i can be calculated by:

  $$\Delta_i = -w_i \tag{3.3}$$

  We can see that a *DROP* move always leads to a decrease to the objective function.

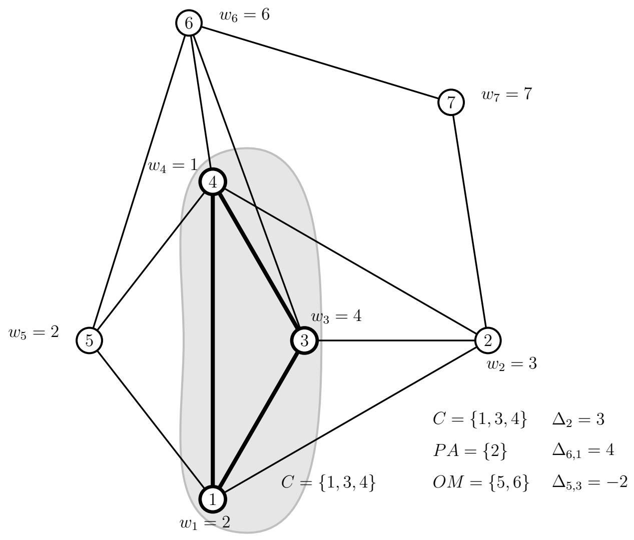

$w_6 = 6$
$w_7 = 7$
$w_4 = 1$
$w_3 = 4$
$w_5 = 2$
$w_2 = 3$
$w_1 = 2$
$C = \{1,3,4\}$
$C = \{1,3,4\}$ $\Delta_2 = 3$
$PA = \{2\}$ $\Delta_{6,1} = 4$
$OM = \{5,6\}$ $\Delta_{5,3} = -2$

Figure 3.2: The relationship between the add move and the swap move **(From clique $C = \{1, 3, 4\}$, the *SWAP* move between vertices 6 and 1 ($SWAP(6, 1)$) leads to a solution $C_1 = \{3, 4, 6\}$, which is better than the solution $C_2 = \{1, 2, 3, 4\}$ obtained by the *ADD* move ($ADD(2)$).)**

3.2.4 Combined neighborhood and neighbor selection strategy

In the case of the unweighted maximum clique problem, *ADD* moves are always preferable to other moves (in a local sense) since they invariably increase the clique weight. However, for the case of MVWCP, a *SWAP* move may lead to a solution better than any solution that can be obtained by a *ADD* move. Fig. 3.2 shows an illustrative example where the current clique C contains three vertices. From Fig. 3.2, we can see that swapping vertices 6 and 1 ($\Delta_{6,1} = w_6 - w_1 = 4$) leads to the solution $C_1 = \{3, 4, 6\}$, which is better than the solution $C_2 = \{1, 2, 3, 4\}$ obtained by adding vertex 2 ($\Delta_2 = 3$) to C.

Moreover, when no *ADD* move is possible ($PA = \emptyset$), a *DROP* move may lead to a solution which is better than any solution that can be obtained by a *SWAP* move (see Fig. 3.3 for an illustrative example). To summarize, for the MVWCP, there is no absolute dominance of one move operator (and its neighborhood) over another move operator. The best move operator to be applied depends on the current search context and should be determined according to the context.

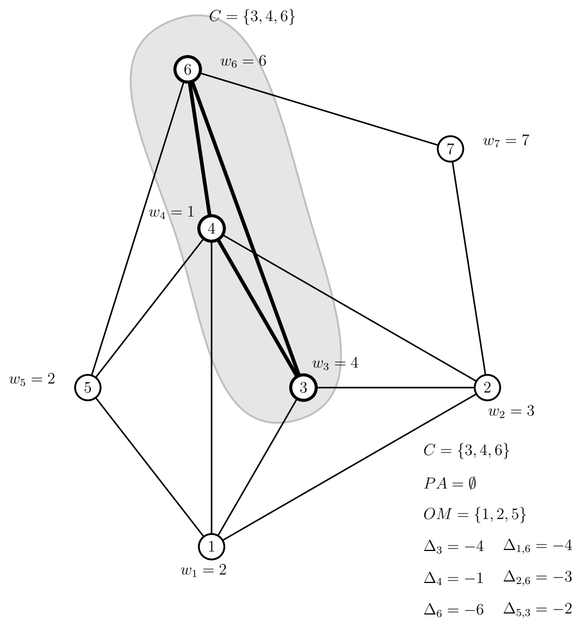


Figure 3.3: The relationship between the drop move and the swap move **(From clique $C = \{3, 4, 6\}$, the *DROP* move of dropping vertex 4 leads to a neighbor solution ($\{3, 6\}$) better than any other neighbor solution obtained by the *SWAP* moves.)**

These observations lead us to create a combined neighborhood $\mathcal{N}$ which corresponds to the *union* of the three neighborhoods N_1, N_2 and N_3, denoted by $\mathcal{N} = N_1 \bigcup N_2 \bigcup N_3$. Using this union neighborhood, our tabu search algorithm selects at each iteration the most favorable move (i.e., with the largest Δ value) among all the *ADD*, *SWAP* and *DROP* moves to generate the next solution. Ties are broken at random.

3.2.5 Tabu list and tabu tenure management

Tabu search characteristically introduces a tabu list to forbid recently visited solutions from being revisited. In our MN/TS algorithm, we adopt the following general prohibition rule: a vertex that leaves the current clique C (by a *SWAP* or *DROP* move) is forbidden to move back to C for the next tt iterations (tabu tenure). A vertex that joins the clique C (by an *ADD* or *SWAP* move) is free to be removed from C without restriction.

With this prohibition rule, no tabu list is needed for the *ADD* moves. This choice can be intuitively explained by the fact that due to the objective of maximizing the clique weight, an added vertex has little chance to be removed anyway. (As noted in [Glover, 1989], a tabu tenure to prevent elements from being dropped should typically be smaller than one to prevent elements from being added. We have simply elected to make the smaller tenure 0.)

For the $SWAP$ move, when a vertex $i \in OM$ is swapped with the only node $j \in C$ not connected to i, j is prohibited to be moved back to C for the next T_{swap} iterations while no tabu status is assigned to i. T_{swap} is tuned dynamically according to the cardinality of OM:

$$T_{swap} = random(|OM|) + T_1 \tag{3.4}$$

where T_1 is set equal to 7 and $random(|OM|)$ takes a random integer in the range $[1, ..., |OM|]$.

For the $DROP$ move, each time a vertex i is removed from C, moving i back to C is declared tabu for the next T_1 iterations where $T_1 = 7$.

Our tabu restrictions (like most of those employed by tabu search) apply to attributes of solutions that are affected by the moves - in this case, the vertices affected by the moves. We call a move tabu if one of its attributes is tabu (hence in this case the vertex that would be added to C), and employ the common aspiration criterion that permits a move to be accepted in spite of being tabu if it produces a solution better than any found so far. A move that is not tabu or that satisfies the aspiration criterion is called *admissible.*

3.2.6 Multistart strategy and stop criteria

Our MN/TS algorithm examines at each iteration the three neighborhoods and selects an admissible move that produces the most favorable neighboring solution. The inclusion of all three neighborhoods allows the algorithm to make a more thorough examination of the solutions around each solution. On the other hand, the tabu restrictions provide a form of local diversification by forcing the search to leave the regions already examined. To establish a more global form of diversification, and thereby reinforce the capacity of the algorithm to visit unexplored areas in the search space, we employ a multistart strategy to restart the search from new starting points. A restart is triggered each time the current search is judged to be trapped in a deep local optimum, a condition that is deemed to occur upon exceeding a maximum allowable number of consecutive iterations without improving the clique weight. We call this number the depth of the search (denoted by L).

Basically, our multistart tabu search algorithm iterates the following two steps until the stop criterion is satisfied:

1. Generate a new start point C (see Section 3.2.2).
2. Apply the tabu search procedure to improve the solution C until the fixed depth L is reached.

The algorithm stops when it attains a predetermined maximum number of iterations ($Iter_{max}$). The complete MN/TS algorithm is described as Algorithm 3.1. Each outside *while* loop triggers a restart of the tabu search procedure which is realized in the inner

Algorithm 3.1: The multi-neighborhood tabu search approach for MVWCP

```
Require: A weighted graph $G = (V, E, w)$, integer $L$ (search depth), $Iter_{max}$ (max. allowed
    iterations)
Ensure: A clique $C^*$ with its weight $W(C^*)$
 1: Begin
 2: $Iter = 0$ {Iteration counter}
 3: $C^* = \emptyset$
    {Each loop triggers a restart of the tabu search procedure}
 4: while ($Iter < Iter_{max}$) do
 5:   $C = Initialize()$ {Section 3.2.2}
 6:   Initiate $tabu\_list$ {Section 3.2.5}
 7:   $NI = 0$ {$NI$ is the consecutive iterations during which $W(C)$ is not improved}
 8:   $C_{local\_best} = C$ {$C_{local\_best}$ is the best solution during the inner while loop}
      {The inner while loop corresponds to a round of the tabu search procedure}
 9:   while ($NI < L$) do
10:     Construct neighborhoods $N_1$, $N_2$ and $N_3$ from $C$ {Section 3.2.3}
11:     Choose an overall best allowed neighbor $C' \in N_1 \bigcup N_2 \bigcup N_3$ according to max gain
        criterion {Section 3.2.4}
12:     $C = C'$ {Move to the new solution}
13:     $NI = NI + 1$
14:     $Iter = Iter + 1$
15:     Update $tabu\_list$ {Section 3.2.5}
16:     if ($W(C) > W(C_{local\_best})$) then
17:       $NI = 0$
18:       $C_{local\_best} = C$
19:     end if
20:   end while
21:   if ($W(C_{local\_best}) > W(C^*)$) then
22:     $C^* = C_{local\_best}$
23:   end if
24: end while
25: End
26: Return (Clique $C^*$))
```

while loop. The variables C and C' designate respectively the current solution and one of its neighboring solution. C_{local_best} is the best solution found during one inner *while* loop while C^* is the overall best solution found by the algorithm.

3.3 Discussion

The move operators ADD, $SWAP$ and $DROP$ (and particularly ADD and $SWAP$, see Section 1.4.2.1) have been widely used in previous studies for both the MVWCP and MCP. However, previous studies have applied these operators independently and sequentially rather than making reference to their union as done here.

For instance, the PLS approach for the MVWCP ([Pullan, 2008]) alternates between a greedy expansion phase during which suitable vertices are added to the current clique followed by a plateau phase where vertices of the current clique are swapped with some vertices out of the clique. This strategy implicitly causes PLS to give a higher priority to ADD moves even if a $SWAP$ move may lead to a solution better than any ADD move (see Fig. 2 for an example). Such a sequential application of ADD and $SWAP$ moves can miss favorable neighboring solutions. In short, the union neighborhood explored by MN/TS ensures a more aggressive and intensified examination of the search space, increasing the chance to find solutions of better quality. In Section 3.5, we give computational evidence of this assertion. Another difference between our method and PLS is that MN/TS picks the two vertices for a *SWAP(i,j)*, according to the move gain, while PLS selects a vertex i with the largest weight w_i in OM to exchange with the only vertex not connected to i in C.

Finally, for the unweighted MCP, most local search methods (such as [Battiti and Protasi, 2001; Gendreau *et al.*, 1993; Grosso *et al.*, 2004; Katayama *et al.*, 2005; Pullan, 2006]) use these moves in manner similar to the way they are employed in PLS. These algorithms differ from each other chiefly in: (1) the strategies for exploring the neighborhoods, (2) the scheme of vertex selection and (3) the prohibition mechanism applied to the performed moves.

3.4 Experimental results

This section is dedicated to an intensive evaluation of the proposed algorithm.[1] For this purpose, we present computational results on a large panel of benchmark instances and show comparisons with state of the art algorithms when such comparisons are possible.

3.4.1 Benchmark instances and experimental settings

Our MN/TS algorithm is programmed in C and compiled using GNU GCC on a PC with 2.83 GHz CPU and 8G RAM. Like [Pullan, 2008] and given the stochastic nature of the MN/TS algorithm, each instance is solved 100 times independently by MN/TS with

[1]The source code of our MN/TS algorithm is publically available at: http://www.info.univ-angers.fr/pub/hao/clique.html.

different random seeds. The maximum allowed iterations $Iter_{max}$ (see Algorithm 3.1) per run and per instance is set equal to 10^8. For the search depth L (see Section 3.2.6), we use $L = 4000$ for the instances of the weighted case (MVWCP). For the unweighted case (MCP), we use $L = 10^4$ except for the brock and san families (DIMACS) for which L is equal to 100.

To evaluate the performance of the proposed MN/TS algorithm, three sets of test problems are considered in the experiments, in total constituting 136 instances. The first set of benchmarks is composed of 80 DIMACS-W benchmarks. The second set of benchmarks consists of a set of 40 BHOSLIB-W instances. The third set of benchmarks include 16 instances derived from the set packing problem [18] with sizes ranging from 1000 to 2000. A detailed description for these three sets of benchmarks can be found in Section 1.6.

3.4.2 Experimental results for the maximum vertex weight clique problem

In Tables 3.1 and 3.2, we show respectively the computational results of our algorithm on the set of 80 DIMACS-W instances and on the set of 40 BHOSLIB-W benchmarks. Columns 2–3 give the features of each tested instance: the number of vertices (Node) and the largest known clique size for the graph. In columns 4–9, we give the same computational statistics as in [Pullan, 2008] (our main reference algorithm): the maximum weight obtained by MN/TS over the 100 independent trials (W_{best}), the cardinality of the obtained maximum weighted clique ($|C|$), the average weight over the 100 trials (W_{avg}), the number of successful trials in which MN/TS reached W_{best} (*Success*), the average time (*AvgTime*) and the average iterations (*Iteration*) over these successful trials.

Table 3.1: Results obtained by MN/TS on the 80 DIMACS-W benchmarks.

Instance	Node	ω	W_{best}	$\|C\|$	W_{avg}	Success	AvgTime	Iteration
brock200_1	200	21*	2821	19	2821	100	< 0.01	2341
brock200_2	200	12*	1428	9	1428	100	< 0.01	2120
brock200_3	200	15*	2062	13	2062	100	< 0.01	2025
brock200_4	200	17*	2107	13	2107	100	< 0.01	2482
brock400_1	400	27*	3422	21	3422	100	0.03	7283
brock400_2	400	29*	3350	21	3350	100	0.03	7601
brock400_3	400	31*	3471	23	3471	100	0.03	7887
brock400_4	400	33*	3626	33	3626	100	4.70	1955245
brock800_1	800	23*	3121	20	3121	100	0.05	7658
brock800_2	800	24*	3043	18	3043	100	0.20	28159
brock800_3	800	25*	3076	20	3076	100	0.08	14002
brock800_4	800	26*	2971	26	2971	100	49.70	9033704
C125.9	125	34*	2529	30	2529	100	0.02	15579
C250.9	250	44*	5092	40	5092	100	0.06	23974
C500.9	500	57	6955	48	6955	100	0.07	21806
C1000.9	1000	68	9254	61	9254	100	8.9	3378709
C2000.5	2000	16	2466	14	2466	100	1.84	62469
C2000.9	2000	80	10999	72	10971.92	22	168.11	36667727
C4000.5	4000	18	2792	16	2792	100	80.56	1344781
DSJC500.5	500	13*	1725	12	1725	100	0.04	5471
DSJC1000.5	1000	15*	2186	13	2186	100	0.20	14184
keller4	171	11*	1153	11	1153	100	0.03	10257
keller5	776	27	3317	27	3317	100	3.17	445956
keller6	3361	59	8062	56	7939.49	5	606.15	53687525
MANN_a9	45	16*	372	16	372	100	< 0.01	2813
MANN_a27	378	126*	12281	126	12273.28	1	88.28	15724873
MANN_a45	1035	345*	34192	340	34172.9	1	390.58	25104191
MANN_a81	3321	1100	111128	1094	111108.31	1	832.24	18550423

Table 3.1 – continued from previous page

Instance	Node	ω	W_{best}	$\|C\|$	W_{avg}	Success	AvgTime	Iteration
hamming6-2	64	32*	1072	32	1072	100	< 0.01	2305
hamming6-4	64	4*	134	4	134	100	< 0.01	2011
hamming8-2	256	128*	10976	128	10976	100	< 0.01	10742
hamming8-4	256	16*	1472	16	1472	100	< 0.01	2086
hamming10-2	1024	512*	50512	512	50512	100	0.92	118180
hamming10-4	1024	40	5129	35	5129	100	2.21	407528
gen200_p0.9_44	200	44*	5043	37	5043	100	< 0.01	3203
gen200_p0.9_55	200	55*	5416	52	5416	100	0.33	146200
gen400_p0.9_55	400	55	6718	47	6718	100	0.15	47748
gen400_p0.9_65	400	65	6940	48	6940	100	0.04	12861
gen400_p0.9_75	400	75	8006	75	8006	100	0.88	264342
c-fat200-1	200	12*	1284	12	1284	100	0.14	74106
c-fat200-2	200	24*	2411	23	2411	100	0.06	33207
c-fat200-5	200	58*	5887	58	5887	100	0.02	13569
c-fat500-1	500	14*	1354	12	1354	100	0.73	162912
c-fat500-2	500	26*	2628	24	2628	100	0.33	71620
c-fat500-5	500	64*	5841	62	5841	100	0.14	34146
c-fat500-10	500	126*	11586	124	11586	100	0.06	18163
johnson8-2-4	28	4*	66	4	66	100	< 0.01	3762
johnson8-4-4	70	14*	511	14	511	100	< 0.01	2038
johnson16-2-4	120	8*	548	8	548	100	0.23	101807
johnson32-2-4	496	16*	2033	16	2033	100	0.53	113880
p_hat300-1	300	8*	1057	7	1057	100	0.02	2986
p_hat300-2	300	25*	2487	20	2487	100	< 0.01	3141
p_hat300-3	300	36*	3774	29	3774	100	0.02	6862
p_hat500-1	500	9*	1231	8	1231	100	0.03	2707
p_hat500-2	500	36*	3920	31	3920	100	< 0.01	732
p_hat500-3	500	50	5375	42	5375	100	0.10	28840
p_hat700-1	700	11*	1441	9	1441	100	0.03	2446
p_hat700-2	700	44*	5290	40	5290	100	0.02	3542
p_hat700-3	700	62	7565	58	7565	100	0.38	90841
p_hat1000-1	1000	10	1514	9	1514	100	0.08	6153
p_hat1000-2	1000	46	5777	40	5777	100	0.11	13405
p_hat1000-3	1000	68	8111	58	8111	100	1.23	235604
p_hat1500-1	1500	12*	1619	10	1619	100	0.06	3271
p_hat1500-2	1500	65	7360	58	7360	100	0.82	75206
p_hat1500-3	1500	94	10321	84	10319.92	96	188.38	18432419
san200_0.7_1	200	30*	3370	30	3370	100	0.17	44988
san200_0.7_2	200	18*	2422	14	2422	100	0.02	4127
san200_0.9_1	200	70*	6825	70	6825	100	0.13	57024
san200_0.9_2	200	60*	6082	60	6082	100	0.21	79785
san200_0.9_3	200	44*	4748	34	4748	100	< 0.01	5745
san400_0.5_1	400	13*	1455	8	1455	100	0.06	5685
san400_0.7_1	400	40*	3941	40	3941	100	13.68	1688081
san400_0.7_2	400	30*	3110	30	3110	100	43.34	5330560
san400_0.7_3	400	22*	2771	18	2771	100	0.05	9267
san400_0.9_1	400	100*	9776	100	9776	100	1.29	306069
san1000	1000	15*	1716	9	1716	100	13.01	471338
sanr200-0.7	200	18*	2325	15	2325	100	< 0.01	2049
sanr200-0.9	200	42*	5126	36	5126	100	< 0.01	2168
sanr400-0.5	400	13*	1835	11	1835	100	0.02	2941
sanr400-0.7	400	21	2992	18	2992	100	< 0.01	2745

For problems in the MVWCP class, studies in the literature are often based on DIMACS-W instances (different weighting functions may be used). We are unaware of studies reporting computational results on the BHOSLIB-W benchmarks. For this reason, our comparisons reported in the next section are based on DIMACS-W benchmarks (as well as a set of instances from the set packing problem) while our results on the BHOSLIB-W benchmarks can serve as a basis for performance assessment of other MVWCP algorithms.

Table 3.2: Results obtained by MN/TS on the 40 BHOSLIB-W benchmarks.

Instance	Node	ω	W_{best}	$\|C\|$	W_{avg}	Success	AvgTime	Iteration
frb30-15-1	450	30*	2990	27	2990	100	0.35	162927
frb30-15-2	450	30*	3006	28	3006	100	3.45	1628915
frb30-15-3	450	30*	2995	27	2995	100	4.72	2147505
frb30-15-4	450	30*	3032	28	3032	100	0.12	53148
frb30-15-5	450	30*	3011	27	3011	100	3.01	1404617

Table 3.2 – continued from previous page

Instance	Node	ω	W_{best}	$\|C\|$	W_{avg}	Success	AvgTime	Iteration
frb35-17-1	595	35*	3650	33	3650	100	25.80	10949043
frb35-17-2	595	35*	3738	33	3736.84	96	72.09	36780076
frb35-17-3	595	35*	3716	33	3716	100	7.72	3208297
frb35-17-4	595	35*	3683	35	3678.31	77	94.03	46627497
frb35-17-5	595	35*	3686	33	3686	100	8.09	3306241
frb40-19-1	760	40*	4063	37	4062.15	83	85.57	12557557
frb40-19-2	760	40*	4112	36	4111.16	87	134.58	29716520
frb40-19-3	760	40*	4115	36	4108.30	19	215.98	44792105
frb40-19-4	760	40*	4136	37	4135.56	89	96.65	13321879
frb40-19-5	760	40*	4118	36	4117.6	90	178.89	31692738
frb45-21-1	945	45*	4760	41	4748.66	44	126.26	41702954
frb45-21-2	945	45*	4784	42	4775.86	47	228.03	42332553
frb45-21-3	945	45*	4765	43	4756.90	26	125.35	42132692
frb45-21-4	945	45*	4799	42	4772.41	43	174.73	34953953
frb45-21-5	945	45*	4779	43	4777.38	82	193.82	35802284
frb50-23-1	1150	50*	5494	47	5484.74	6	186.62	52803333
frb50-23-2	1150	50*	5462	47	5434.14	3	149.66	45053333
frb50-23-3	1150	50*	5486	47	5480.29	53	158.71	45289811
frb50-23-4	1150	50*	5454	46	5451.69	9	176.41	49915555
frb50-23-5	1150	50*	5498	47	5495.70	89	110.85	36065699
frb53-24-1	1272	53*	5670	50	5637.94	5	233.22	54638030
frb53-24-2	1272	53*	5707	48	5676.56	6	145.22	40515069
frb53-24-3	1272	53*	5640	49	5610.79	15	215.79	62672666
frb53-24-4	1272	53*	5714	50	5645.61	7	449.39	73105032
frb53-24-5	1272	53*	5659	49	5628.77	5	294.00	47012340
frb56-25-1	1400	56*	5916	53	5836.85	3	308.90	49212581
frb56-25-2	1400	56*	5872	52	5807.70	1	73.25	17174823
frb56-25-3	1400	56*	5859	51	5799.38	1	181.93	47664235
frb56-25-4	1400	56*	5892	51	5839.16	3	104.58	28605284
frb56-25-5	1400	56*	5839	52	5768.39	1	322.70	91502378
frb59-26-1	1534	59*	6591	55	6547.53	3	166.20	42284765
frb59-26-2	1534	59*	6645	56	6567.07	3	212.49	54746666
frb59-26-3	1534	59*	6608	55	6514.18	1	232.77	60188544
frb59-26-4	1534	59*	6592	54	6498.37	1	318.39	47624522
frb59-26-5	1534	59*	6584	53	6522.57	1	161.47	30820580

3.4.3 Comparative results for the maximum *vertex weight* clique problem

In order to show the relative effectiveness of our MN/TS for the MVWCP, we first compare MN/TS with two state of the art algorithms from the literature [Pullan, 2008; Mannino and Stefanutti, 1999]. The main comparison criterion is the quality of the solutions found. Due to the differences among the programming languages, data structures, compiler options and computers, computing times are provided only for indicative purposes. Since the reference algorithms report results only for DIMCAS-W benchmarks, our first comparisons are based on this set of instances.

Table 3.3: Comparative results between MNTS and PLS on the set of 80 DIMACS-W benchmarks.

Instance	TS			PLS			$\Delta(TS-PLS)$
	W_{best}	$Success$	$CPU(s)$	W_{best}	$Success$	$CPU(s)$	
brock200_1	2821	100	< 0.01	2821	100	0.19	0
brock200_2	1428	100	< 0.01	1428	100	0.02	0
brock200_3	2062	100	< 0.01	2062	100	0.01	0
brock200_4	2107	100	< 0.01	2107	100	0.70	0
brock400_1	3422	100	0.03	3422	32	437.19	0
brock400_2	3350	100	0.03	3350	61	415.95	0
brock400_3	3471	100	0.03	3471	100	12.04	0
brock400_4	3626	100	4.70	3626	100	0.05	0
brock800_1	3121	100	0.05	3121	100	31.46	0
brock800_2	3043	100	0.20	3043	69	893.42	0
brock800_3	3076	100	0.08	3076	100	3.35	0
brock800_4	2971	100	49.70	2971	100	3.77	0

Table 3.3 – continued from previous page

Instance	TS			PLS			$\Delta(TS - PLS)$
	W_{best}	*Success*	$CPU(s)$	W_{best}	*Success*	$CPU(s)$	
C125.9	2529	100	0.02	2529	100	8.08	0
C250.9	5092	100	0.06	5092	17	247.69	0
C500.9	**6955**	100	0.07	6822	-	-	133
C1000.9	**9254**	100	8.90	8965	5	344.74	289
C2000.5	2466	100	1.84	2466	18	711.27	0
C2000.9	**10999**	22	168.11	10028	-	-	971
C4000.5	2792	100	80.56	2792	-	-	0
DSJC500.5	1725	100	0.04	1725	100	0.95	0
DSJC1000.5	2186	100	0.20	2186	100	47.76	0
keller4	1153	100	0.03	1153	100	0.02	0
keller5	3317	100	3.17	3317	100	119.24	0
keller6	**8062**	5	606.15	7382	-	-	680
MANN_a9	372	100	< 0.01	372	100	$< \epsilon$	0
MANN_a27	**12281**	1	88.28	12264	-	-	17
MANN_a45	**34192**	1	390.58	34129	-	-	63
MANN_a81	**111128**	1	832.24	110564	-	-	564
hamming6-2	1072	100	< 0.01	1072	100	$< \epsilon$	0
hamming6-4	134	100	< 0.01	134	100	$< \epsilon$	0
hamming8-2	10976	100	< 0.01	10976	100	$< \epsilon$	0
hamming8-4	1472	100	< 0.01	1472	100	$< \epsilon$	0
hamming10-2	50512	100	0.92	50512	100	$< \epsilon$	0
hamming10-4	**5129**	100	2.21	5086	1	1433.07	43
gen200_p0.9_44	5043	100	< 0.01	5043	100	4.44	0
gen200_p0.9_55	5416	100	0.33	5416	100	0.05	0
gen400_p0.9_55	6718	100	0.15	6718	2	340.11	0
gen400_p0.9_65	**6940**	100	0.04	6935	4	200.79	5
gen400_p0.9_75	8006	100	0.88	8006	100	$< \epsilon$	0
c-fat200-1	1284	100	0.14	1284	100	$< \epsilon$	0
c-fat200-2	2411	100	0.06	2411	100	$< \epsilon$	0
c-fat200-5	5887	100	0.02	5887	100	$< \epsilon$	0
c-fat500-1	1354	100	0.73	1354	100	$< \epsilon$	0
c-fat500-2	2628	100	0.33	2628	100	0.01	0
c-fat500-5	5841	100	0.14	5841	100	$< \epsilon$	0
c-fat500-10	11586	100	0.06	11586	100	$< \epsilon$	0
johnson8-2-4	66	100	< 0.01	66	100	$< \epsilon$	0
johnson8-4-4	511	100	< 0.01	511	100	$< \epsilon$	0
johnson16-2-4	548	100	0.23	548	100	$< \epsilon$	0
johnson32-2-4	2033	100	0.53	2033	100	44.68	0
p_hat300-1	1057	100	0.02	1057	100	0.01	0
p_hat300-2	2487	100	< 0.01	2487	100	19.36	0
p_hat300-3	3774	100	0.02	3774	47	418.11	0
p_hat500-1	1231	100	0.03	1231	100	0.42	0
p_hat500-2	3920	100	< 0.01	**3925**	-	-	-5
p_hat500-3	**5375**	100	0.10	5361	-	-	14
p_hat700-1	1441	100	0.03	1441	100	0.20	0
p_hat700-2	5290	100	0.02	5290	100	78.51	0
p_hat700-3	7565	100	0.38	7565	12	718.40	0
p_hat1000-1	1514	100	0.08	1514	100	7.61	0
p_hat1000-2	5777	100	0.11	5777	87	940.62	0
p_hat1000-3	**8111**	100	1.23	7986	-	-	125
p_hat1500-1	1619	100	0.06	1619	100	48.91	0
p_hat1500-2	**7360**	100	0.82	7328	4	1056.19	32
p_hat1500-3	**10321**	96	188.38	10014	-	-	307
san200_0.7_1	3370	100	0.17	3370	100	$< \epsilon$	0
san200_0.7_2	2422	100	0.02	2422	66	397.38	0
san200_0.9_1	6825	100	0.13	6825	100	$< \epsilon$	0
san200_0.9_2	6082	100	0.21	6082	100	$< \epsilon$	0
san200_0.9_3	4748	100	< 0.01	4748	72	219.68	0
san400_0.5_1	1455	100	0.06	1455	100	200.44	0
san400_0.7_1	3941	100	13.68	3941	100	0.03	0
san400_0.7_2	3110	100	43.34	3110	100	0.05	0
san400_0.7_3	2771	100	0.05	2771	100	4.41	0
san400_0.9_1	9776	100	1.29	9776	100	$< \epsilon$	0
san1000	1716	100	13.01	1716	-	-	0
sanr200-0.7	2325	100	< 0.01	2325	100	0.62	0
sanr200-0.9	5126	100	< 0.01	5126	5	182.54	0
sanr400-0.5	1835	100	0.02	1835	100	0.67	0
sanr400-0.7	2992	100	< 0.01	2992	100	141.50	0

The weight of each vertex i is set equal to $(i \bmod 200) + 1$. Statistics are based on 100 trials of each algorithm. An entry with "-" for PLS means that PLS was terminated because of excessive CPU time. An entry with "$< \epsilon$" signifies that the average CPU time required by PLS was less than 0.01 seconds. MNTS finds improved solutions for 13 instances (in bold).

Table 3.3 summarizes the comparative results between our MN/TS and the well-known

PLS algorithm [Pullan, 2008]. For both algorithms, the number of the trials devoted to solving each instance was 100. In Table 3, we indicate the largest weights obtained by the two algorithms for each graph over the 100 independent trials (W_{best}), the number of successful trials where an algorithm reached W_{best} ($Success$), the average time (CPU) over these successful trials. Finally, column 8 indicates the difference in the largest weights obtained by MN/TS and PLS.

Table 3.3 discloses that, over the 80 instances tested, the quality of solutions obtained by our MN/TS algorithm matches or exceeds that of solutions obtained by the PLS algorithm except in one case (p_hat500-2) where our method obtained a slightly worse solution. By contrast, the MN/TS method obtained strictly superior solutions on 13 out of the 80 instances (C500.9, C1000.9, C2000.9, keller6, MANN_a27, MANN_a45, MANN_a81, hamming10-4, gen400_p0.9_65, p_hat500-3, p_hat1000-3, p_hat1500-2, p_hat1500-3). For all of the remaining 66 instances on which the two algorithms attain the same largest weight (W_{best}), MN/TS has a success rate of 100%, while PLS has a 100% success rate on 52 of these instances.

According to [Pullan, 2008], the experiments of PLS were performed on a computer that, when executing the DIMACS MC Machine Benchmark program (ftp://dimacs.rutgers.edu in directory /pub/dsj/clique), required respectively 0.31, 1.93 and 7.35 CPU seconds for the graphs r300.5, r400.5 and r500.5. Running this benchmark program on our computer leads to respectively 0.46, 2.79 and 10.44 CPU seconds for these three graphs. In other words, the computer used by PLS is slightly faster than the computer we used for our experiments. Table 3.3 shows that our MN/TS algorithm required in most cases less computing time to obtain solutions of the same or better quality.

To augment the above comparison, Table 3.4 contrasts the results of our MN/TS with those of the AugSearch algorithm reported in [Mannino and Stefanutti, 1999]. The authors of the AugSearch algorithm used a subset of 36 DIMACS graphs with weighting function in which the weight w_i of vertex i is set equal to $(i \; mod \; 10)+1$. We have run our algorithm 100 times to solve each of these instances and report the computational statistics in Table 3.4. As demonstrated, our MN/TS algorithm attains easily all the best objective values W_{best} reported in [Mannino and Stefanutti, 1999] with a short computing time ranging from less than 1 second to 13 minutes. (The computing times of AugSearch are based on an IBM-RISC SYSTEM 6000 POWER station 375.) In addition, MN/TS obtains solutions better than those found by AugSearch in 11 cases out of the 36 instances.

3.4.4 Computational results on structured instances from set packing

In this section, we report the outcomes of testing the MN/TS algorithm on the set of 16 structured instances derived from the set packing problem [Alidaee *et al.*, 2008; Delorme *et al.*, 2004], which have sizes ranging from 1000 to 2000. In [Alidaee *et al.*, 2008], the set packing problem is solved via a unconstrained quadratic formulation while in [Delorme *et al.*, 2004], a dedicated GRASP heuristic was used. For the approach to convert a set packing problem into a maximum weight independent set problem, interested readers are referred to [Kwon, 2005]. (A maximum weight independent set in a graph corresponds to a maximum weight clique in the complement of the graph.)

Table 3.4: Comparative results between MN/TS and AugSearch on 36 DIMACS *weighted* instances.

Instance	MN/TS		AugSearch		$\Delta(MN/TS - AugSearch)$
	W_{best}	Time	W_{best}	Time	
C125.9	215	< 0.01	215	5.28	0
C250.9	304	0.01	304	12.93	0
C500.9	**390**	0.08	385	3363.74	5
C1000.9	**491**	6.08	470	553.53	21
C2000.9	**585**	15.21	531	2430.77	54
C2000.5	**129**	2.43	113	324.07	16
DSJC500.5	**98**	0.05	94	19.19	4
DSJC1000.5	**114**	0.87	102	732.74	12
MANN_a27	867	0.50	867	0.01	0
MANN_a45	2403	100.23	2403	0.01	0
MANN_a81	7794	327.22	(18250)	0.01	-
brock200_2	**77**	< 0.01	76	0.19	1
brock200_4	114	< 0.01	114	114	0
brock400_2	178	< 0.01	178	1255.78	0
brock400_4	175	< 0.01	175	113.50	0
brock800_2	**159**	0.16	155	1841	4
brock800_4	**158**	0.13	153	803.11	5
gen200_p0.9_44	277	< 0.01	277	48.20	0
gen200_p0.9_55	293	0.57	293	37.26	0
gen400_p0.9_55	374	0.06	374	2220.10	0
gen400_p0.9_65	391	0.65	391	208.30	0
gen400_p0.9_75	401	88.88	401	2578.57	0
hamming8-4	106	< 0.01	106	0.1	0
hamming10-4	**294**	0.03	291	1499.09	3
keller4	87	< 0.01	87	2.48	0
keller5	**201**	0.21	195	1307.16	6
keller6	445	821.4	(1205)	922.18	-
p_hat300-1	71	< 0.01	71	2.38	0
p_hat300-2	180	< 0.01	180	0.40	0
p_hat300-3	260	< 0.01	260	6.60	0
p_hat700-1	77	0.02	77	36.17	0
p_hat700-2	284	0.01	284	200.45	0
p_hat700-3	418	< 0.01	418	42.95	0
p_hat1500-1	89	< 0.01	89	1113.43	0
p_hat1500-2	416	6.50	416	758.03	0
p_hat1500-3	595	1.46	595	517.60	0

The weight of each vertex i is set equal to $(i \ mod \ 10)+1$ according to [Mannino and Stefanutti, 1999]. Notice that the results reported in [Mannino and Stefanutti, 1999] for two instances (MANN_a81 and keller6) are wrong since their respective W_{best} values are superior to their respective upper bounds (11000 and 590). MN/TS finds improved solutions for 11 instances (in bold).

Table 3.5: Computational results on the 16 weighted maximum clique instances from the set packing problem.

Instance	n	m	BKR	MN/TS				
				W_{best}	W_{avg}	Success	AvgTime	Iteration
1	1000	5000	67	67	67	100	< 0.01	987
2	1000	5000	4	4	4	100	< 0.01	2466
3	1000	5000	661	661	661	100	0.05	14412
4	1000	5000	48	48	48	100	0.05	11367
5	1000	1000	222	222	222	100	0.02	2160
6	1000	1000	15	15	15	100	0.09	9794
7	1000	1000	2260	2260	2259.38	63	164.12	54803313
8	1000	1000	175	175	175	100	0.53	81249
9	2000	10000	40	40	40	100	< 0.01	40
10	2000	10000	2	2	2	100	< 0.01	4
11	2000	10000	478	478	478	100	2.02	116784
12	2000	10000	32	32	32	100	4.17	229540
13	2000	2000	140	140	140	100	0.07	3136
14	2000	2000	9	9	9	100	0.06	1333
15	2000	2000	1784	**1811**	1806.81	6	304.52	51346666
16	2000	2000	131	**135**	135	100	4.96	1111260

The results of this experiment are summarized in Table 3.5. Columns 1–3 give the problem identification [Alidaee *et al.*, 2008]. Column 4 gives the previous best known results reported in [Alidaee *et al.*, 2008] and [Delorme *et al.*, 2004], and columns 5–9 show the computational statistics of the MN/TS algorithm. Table 3.5 shows that MN/TS attains the previous best known results for all these 16 tested instances. Moreover, our algorithm discovers new best results for two instances (ID15 and ID16). This performance is surprising given that our MN/TS algorithm is not specifically designed for the set packing problem.

3.4.5 Experimental results for the *unweighted* maximum clique problem

The results on the weighted instances have shown the efficacy of the MN/TS algorithm for the maximum vertex weight clique problem. In this section, we additionally test the MN/TS algorithm on the unweighted maximum clique problem, using the DIMACS and BHOSLIB benchmark instances. For this experiment, the search depth L is set equal to 10^4 except for the brock and san graphs (DIMACS) for which L is set equal to 100.

Table 3.6: The computational results obtained by MN/TS on the 80 *unweighted* DIMACS benchmarks.

Instance	Node	ω	W_{best}	W_{avg}	Success	AvgTime	Iteration
brock200_1	200	21*	21	21	100	< 0.01	3639
brock200_2	200	12*	12	12	100	0.06	27460
brock200_3	200	15*	15	15	100	0.07	37184
brock200_4	200	17*	17	17	100	0.09	53893
brock400_1	400	27*	27	27	100	10.27	2571929
brock400_2	400	29*	29	29	100	1.34	551643
brock400_3	400	31*	31	31	100	0.63	259892
brock400_4	400	33*	33	33	100	0.28	121510
brock800_1	800	23*	23	22.72	86	188.14	29580466
brock800_2	800	24*	24	23.88	96	156.47	26960764
brock800_3	800	25*	25	25	100	118.57	20949527
brock800_4	800	26*	26	26	100	62.38	10861330
C125.9	125	34*	34	34	100	< 0.01	114
C250.9	250	44*	44	44	100	< 0.01	706

Table 3.6 – continued from previous page

Instance	Node	ω	W_{best}	W_{avg}	Success	AvgTime	Iteration
C500.9	500	57	57	57	100	0.06	28868
C1000.9	1000	68	68	68	100	0.63	197084
C2000.5	2000	16	16	16	100	0.07	33553
C2000.9	2000	80	80	78.37	1	563.70	99176504
C4000.5	4000	18	18	18	100	144.37	3779319
DSJC500.5	500	13*	13	13	100	0.24	30471
DSJC1000.5	1000	15*	15	15	100	0.61	44184
keller4	171	11*	11	11	100	< 0.01	123
keller5	776	27	27	27	100	0.05	10431
keller6	3361	59	59	59	100	97.87	7407809
MANN_a9	45	16*	16	16	100	< 0.01	1853
MANN_a27	378	126*	126	126	100	3.42	564530
MANN_a45	1035	345*	340	340	6	90.58	2510419
MANN_a81	3321	1100	1090	1090	8	632.24	8550423
hamming6-2	64	32*	32	32	100	< 0.01	128
hamming6-4	64	4*	4	4	100	< 0.01	4
hamming8-2	256	128*	128	128	100	< 0.01	222
hamming8-4	256	16*	16	16	100	< 0.01	23
hamming10-2	1024	512*	512	512	100	< 0.01	1400
hamming10-4	1024	40	40	40	100	< 0.01	795
gen200_p0.9_44	200	44*	44	44	100	< 0.01	1116
gen200_p0.9_55	200	55*	55	55	100	< 0.01	396
gen400_p0.9_55	400	55	55	55	100	0.03	13285
gen400_p0.9_65	400	65	65	65	100	< 0.01	852
gen400_p0.9_75	400	75	75	75	100	< 0.01	511
c-fat200-1	200	12*	12	12	100	< 0.01	861
c-fat200-2	200	24*	24	24	100	0.07	35744
c-fat200-5	200	58*	58	58	100	< 0.01	2698
c-fat500-1	500	14*	14	14	100	0.02	4638
c-fat500-2	500	26*	26	26	100	< 0.01	1586
c-fat500-5	500	64*	64	64	100	0.02	6764
c-fat500-10	500	126*	126	126	100	< 0.01	3666
johnson8-2-4	28	4*	4	4	100	< 0.01	4
johnson8-4-4	70	14*	14	14	100	< 0.01	14
johnson16-2-4	120	8*	8	8	100	< 0.01	8
johnson32-2-4	496	16*	16	16	100	< 0.01	16
p_hat300-1	300	8*	8	8	100	< 0.01	94
p_hat300-2	300	25*	25	25	100	< 0.01	77
p_hat300-3	300	36*	36	36	100	< 0.01	346
p_hat500-1	500	9*	9	9	100	< 0.01	84
p_hat500-2	500	36*	36	36	100	< 0.01	124
p_hat500-3	500	50	50	50	100	< 0.01	616
p_hat700-1	700	11*	11	11	100	< 0.01	1071
p_hat700-2	700	44*	44	44	100	< 0.01	143
p_hat700-3	700	62	62	62	100	< 0.01	249
p_hat1000-1	1000	10	10	10	100	< 0.01	180
p_hat1000-2	1000	46	46	46	100	< 0.01	216
p_hat1000-3	1000	68	68	68	100	< 0.01	1630
p_hat1500-1	1500	12*	12	12	100	1.42	75661
p_hat1500-2	1500	65	65	65	100	< 0.01	998
p_hat1500-3	1500	94	94	94	100	< 0.01	1029
san200_0.7_1	200	30*	30	30	100	< 0.01	2716
san200_0.7_2	200	18*	18	18	100	0.05	16594
san200_0.9_1	200	70*	70	70	100	< 0.01	838
san200_0.9_2	200	60*	60	60	100	< 0.01	731
san200_0.9_3	200	44*	44	44	100	< 0.01	1749
san400_0.5_1	400	13*	13	13	100	0.34	37789
san400_0.7_1	400	40*	40	40	100	0.21	37048
san400_0.7_2	400	30*	30	30	100	0.35	61817
san400_0.7_3	400	22*	22	22	100	0.17	39006
san400_0.9_1	400	100*	100	100	100	< 0.01	2728
san1000	1000	15*	15	15	100	54.30	2215110
sanr200-0.7	200	18*	18	18	100	< 0.01	292
sanr200-0.9	200	42*	42	42	100	< 0.01	1251
sanr400-0.5	400	13*	13	13	100	< 0.01	3351
sanr400-0.7	400	21	21	21	100	< 0.01	1609

Table 3.6 shows the performance of MN/TS on the 80 DIMACS benchmarks. The different columns have the same interpretation as before. W_{best} (column 4) identifies the largest clique found by MN/TS. For 78 of the 80 instances, MN/TS finds the previous best known results in less than seven minutes. This performance matches the current best

MCP algorithms like [Pullan *et al.*, 2011; Pullan, 2006; Wu and Hao, 2011a] and dominates other methods.

The outcomes of applying MN/TS to the BHOSLIB benchmark instances are displayed in Table 3.7, reinforcing these findings. In particular, for all of these 40 instances, MN/TS successfully obtains the known optimal solutions. In addition, for 23 instances, MN/TS finds optimal solutions with a success rate of 100%.

In sum, Tables 3.1 to 3.7 together demonstrate that our MN/TS algorithm is not only very effective for the maximum weight clique problem, but also very competitive for the conventional unweighted case for which it was not specially designed.

3.5 Influence of neighborhood combination

One of the most important features of a local search algorithm is certainly the definition of its neighborhood. When several neighborhoods are available, the issue of effective ways for using these neighborhoods becomes relevant [Gaspero and Schaerf, 2006; Lü *et al.*, 2011]. As previously noted, the three neighborhoods N_1, N_2 and N_3 induced respectively by the ADD, $SWAP$ and $DROP$ moves are natural components to embody in an overall choice strategy. In our case, at each iteration of our tabu search approach, we have elected to employ the combined neighborhood $N_1 \bigcup N_2 \bigcup N_3$, from which we select the admissible move (non-tabu or globally improving) yielding the largest move gain.

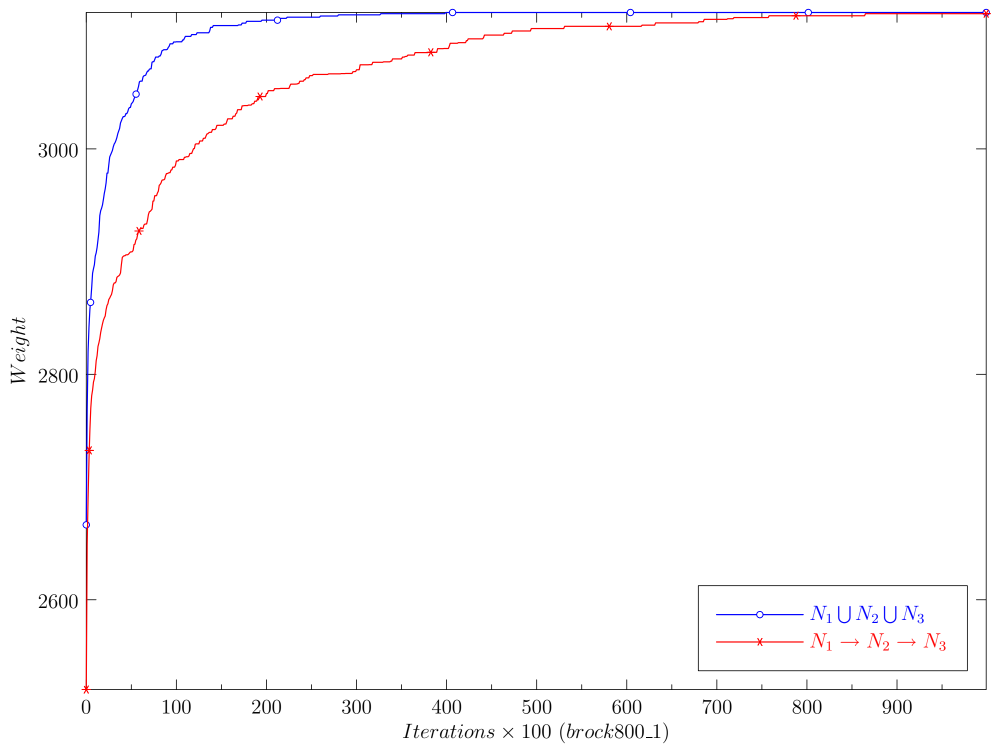
For traditional approaches which have been previously applied to the unweighted maximum clique problem, the basic moves consist of the addition or removal of a single vertex from the current clique. (Swap moves thus trivially decompose into two separate moves [Battiti and Mascia, 2010]). Within this setting of traditional methods for the MCP, the ADD moves are applied whenever possible as they are the only moves that augment the current clique. $DROP$ moves are considered only when no ADD or $SWAP$ move exists. In this section, we perform tests to apply this traditional way of combining neighborhoods to the MVWCP: When admissible ADD moves are present, we select the one yielding the largest move gain (i.e., drawing the move from N_1). Otherwise, if admissible $SWAP$ moves are present, we similarly select one of these moves with the largest gain (drawing the move from neighborhood N_2). If none of these two types of moves is available, a $DROP$ move is applied to remove from C the vertex with the minimum weight (N3). In this approach, the neighborhoods are explored sequentially, as denoted by $N_1 \rightarrow N_2 \rightarrow N_3$.

We apply our MN/TS algorithm to 10 MVWCP instances from the DIMACS-W and BHOSLIB-W benchmarks to compare our $N_1 \bigcup N_2 \bigcup N_3$ neighborhood combination with the $N_1 \rightarrow N_2 \rightarrow N_3$ combination. Each version of the algorithm was run 100 times on each instance with $Iter_{max} = 10^8$. Table 7.2 shows that on all these 10 instances, MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ matches or outperforms MN/TS with $N_1 \rightarrow N_2 \rightarrow N_3$. For three instances (C2000.9, Keller6 and frb59-26-4), MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ achieves a better weight (W_{best}) than MN/TS with $N_1 \rightarrow N_2 \rightarrow N_3$. One also observes that MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ requires significantly fewer iterations to reach the same W_{best}.

To augment these observations, we show in Fig. 3.4 the running profiles of our algorithms with $N_1 \bigcup N_2 \bigcup N_3$ and $N_1 \rightarrow N_2 \rightarrow N_3$ on the instances C1000.9 and brock800_1.

Table 3.7: The computational results obtained by MN/TS on the 40 *unweighted* BHOSLIB benchmark instances.

Instance	Node	ω^*	W_{best}	W_{avg}	Success	AvgTime	Iteration
frb30-15-1	450	30	30	30	100	0.04	11933
frb30-15-2	450	30	30	30	100	0.06	16148
frb30-15-3	450	30	30	30	100	0.57	157523
frb30-15-4	450	30	30	30	100	0.03	8366
frb30-15-5	450	30	30	30	100	0.37	103338
frb35-17-1	595	35	35	35	100	1.56	600704
frb35-17-2	595	35	35	35	100	0.55	215200
frb35-17-3	595	35	35	35	100	0.08	28925
frb35-17-4	595	35	35	35	100	2.08	821830
frb35-17-5	595	35	35	35	100	0.02	88714
frb40-19-1	760	40	40	40	100	0.32	111699
frb40-19-2	760	40	40	40	100	9.15	3146725
frb40-19-3	760	40	40	40	100	1.63	560773
frb40-19-4	760	40	40	40	100	6.72	2267514
frb40-19-5	760	40	40	40	100	45.17	14755785
frb45-21-1	945	45	45	45	100	7.89	2223000
frb45-21-2	945	45	45	45	100	21.71	5824014
frb45-21-3	945	45	45	45	100	53.67	15251125
frb45-21-4	945	45	45	45	100	10.40	2867683
frb45-21-5	945	45	45	45	100	37.22	10616570
frb50-23-1	1150	50	50	49.84	84	116.92	30626106
frb50-23-2	1150	50	50	49.47	47	161.77	43081194
frb50-23-3	1150	50	50	49.15	15	214.58	55481196
frb50-23-4	1150	50	50	50	100	11.91	3136579
frb50-23-5	1150	50	50	50	100	50.90	8056548
frb53-24-1	1272	53	53	52.03	3	240.36	56242252
frb53-24-2	1272	53	53	52.30	30	209.89	48183139
frb53-24-3	1272	53	53	52.91	91	253.96	33611417
frb53-24-4	1272	53	53	52.45	45	178.01	42556535
frb53-24-5	1272	53	53	52.90	90	278.31	39408553
frb56-25-1	1400	56	56	55.22	22	174.02	40273969
frb56-25-2	1400	56	56	55.12	12	127.16	30201302
frb56-25-3	1400	56	56	55.25	25	209.48	47435029
frb56-25-4	1400	56	56	55.85	85	158.14	36531402
frb56-25-5	1400	56	56	56	100	85.57	18921353
frb59-26-1	1534	59	59	58.05	5	242.75	53070428
frb59-26-2	1534	59	59	58.01	1	396.38	86144885
frb59-26-3	1534	59	59	58.23	23	197.36	43876938
frb59-26-4	1534	59	59	58.10	11	192.45	41319157
frb59-26-5	1534	59	59	58.99	99	96.09	20416819



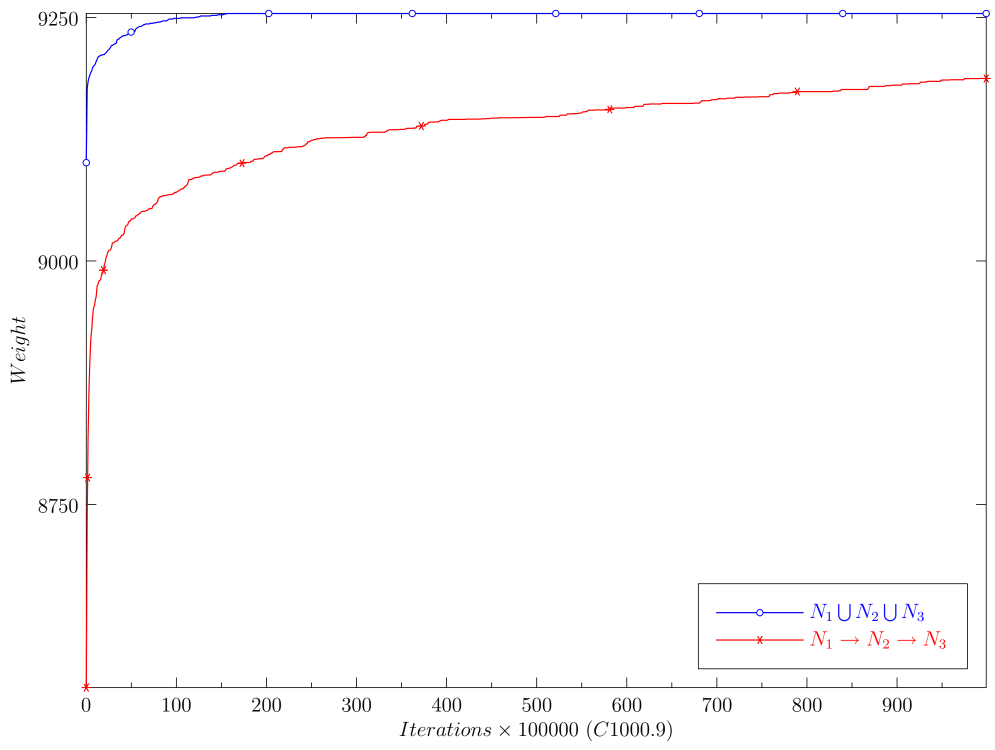


Figure 3.4: Running profile of the two algorithms base on $N_1 \bigcup N_2 \bigcup N_3$ and $N_1 \rightarrow N_2 \rightarrow N_3$ on C1000.9 and brock800_1

Table 3.8: The comparative results between two neighborhood combinations

Instance	$N_1 \bigcup N_2 \bigcup N_3$				$N_1 \rightarrow N_2 \rightarrow N_3$			
	W_{best}	W_{avg}	*Success*	*Iteration*	W_{best}	W_{avg}	*Success*	*Iteration*
brock800_1	3121	3121	100	7658	3121	3121	100	28550
C1000.9	9254	9254	100	3378709	9254	9187.99	43	49203658
C2000.5	2466	2466	100	62469	2466	2466	100	324754
C2000.9	10999	10971.92	22	36667727	10891	10758.24	1	96090684
keller6	8062	7939.49	5	53687525	7706	7456.32	1	68412368
hamming10-4	5129	5129	100	407528	5129	5128.96	97	30752818
p_hat500-2	3920	3920	100	732	3290	3290	100	10702
p_hat1000-3	8111	8111	100	235604	8111	8111	100	8255957
frb59-26-1	6591	6547.53	3	42284765	6591	6556.92	9	53125555
frb59-26-4	6592	6498.37	1	47624522	6575	6534.39	18	35506666

A running profile is defined by the function $i \mapsto f_*(i)$ where i is the number of iterations and $f_*(i)$ is the best value of the objective function (averaged over 100 runs) known at iteration i. Such a profile gives a natural way to observe the evolution of the best values of the objective function during a search.

Fig. 3.4 shows that MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ strongly dominates MN/TS with $N_1 \rightarrow N_2 \rightarrow N_3$ on these two test instances by obtaining a faster and better convergence to the best result.

3.6 Conclusion

A natural concern in local search is to identify how to exploit several different neighborhoods so as to increase the ability of the algorithm to explore the search space more effectively. In this chapter, we have presented a tabu search algorithm for the maximum vertex weight clique problem based on a combined neighborhood induced by three types of moves. The algorithm explores all these moves at each iteration and selects the best admissible (non-tabu or globally improving) solution that yields the largest weight gain. The tabu mechanism creates an effective local diversification and a multistart strategy is employed to create a global diversification.

Our proposed algorithm is evaluated on a large number of MVWCP benchmarks from the BHOSLIB-W and DIMACS-W test sets (containing 40 instances and 80 instances, respectively) and and is also applied to 16 instances derived from the set partitioning problem. Compared with leading reference algorithms from the literature, our MN/TS algorithm finds new best solutions in 26 cases (24 DIMACS-W instances and 2 set packing instances). Moreover, our MN/TS approach exhibits an excellent performance when applied to the classical maximum clique problem, obtaining the best-known solutions for all the BHOSLIB instances and for 78 out of the 80 DIMACS instances. All these results are achieved with a computing time ranging from less than one second to 15 minutes on a standard laptop.

We also provided an analysis to show the relevance of the union combination of the underlying neighborhoods by comparing it to the sequential exploration of these neighborhoods. The outcomes suggest that the union combination of neighborhoods plays a key role in contributing to the effectiveness of the proposed algorithm.

This chapter is dedicated to the maximum vertex weight clique problem. In the next chapter, we will consider the maximum edge weight clique problem, which is another important generalization of MCP, and develop an effective hybrid metaheuristic method to solve the problem.

Chapter 4

A hybrid metaheuristic method for the maximum edge weight clique Problem

The maximum edge weight clique problem (MEWCP) is another important generalization of MCP, which is also known as the maximum diversity problem in the literature. Given an integer m and a complete graph $G = (V, E)$ in which every pair of distinct vertices is connected by a unique edge, each edge $\{i, j\} \in E$ being associated with a positive weight d_{ij}. The MEWCP is to determine a clique of G with exactly m vertices such that the sum of the weights of the edges in the clique is maximized. We present a hybrid metaheuristic algorithm (denoted by MAMEP) for MEWCP. The algorithm uses a dedicated crossover operator to generate new solutions and a constrained neighborhood tabu search procedure for local optimization. MAMEP applies also a distance-and-quality based replacement strategy to maintain population diversity. Extensive evaluations on a large set of 120 benchmark instances show that the proposed approach competes very favorably with the current state-of-art methods for MAMEP. In particular, it consistently and easily attains all the best known results. Moreover, it yields improved results for 6 large MEWCP instances. The key components of MAMEP are analyzed to shed light on their functioning. The content of this chapter is presented in [Wu and Hao, 2012c] which is currently under review.

Contents

4.1 Introduction

Given an integer m and a complete graph $G = (V, E)$ in which every pair of distinct vertices is connected by a unique edge, each edge $\{i, j\} \in E$ being associated with a positive weight d_{ij}. The maximum edge weight clique problem (MEWCP) [Macambira and de Souza, 2000; Sorensen, 2004; Macambira, 2003; Alidaee *et al.*, 2007] is to find a clique C in G such that the sum of the weights of the edges in the clique C is maximized and the number of vertices in C is equal to m. The maximum edge weight clique problem is also called the maximum diversity problem (MDP) [Martí *et al.*, 2011; Palubeckis, 2007], which is more frequently studied in the literature. More formally, the MEWCP may be stated as the following quadratic zero-one integer program [Alidaee *et al.*, 2007]:

$$Maximize\ f(x) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_i x_j \tag{4.1}$$

subject to

$$\sum_{i=1}^{n} x_i = m \tag{4.2}$$

where $n = |V|$ and x_i is a binary variable indicating whether a vertex i is selected to be a member of the clique C.

The decision version of the MEWCP problem is NP-complete (see [Ausiello *et al.*, 1999]), as far as it reduces to the maximum clique problem. In addition to its theoretical significance as a difficult combinatorial problem, MEWCP is notable for its ability to formulate a number of practical applications: location of undesirable or mutually competing facilities [Hunting *et al.*, 2001], decision analysis with multiple objectives [Palubeckis, 2007], composing jury panels [Lozano *et al.*, 2011], genetic engineering [Martí *et al.*, 2011], medical and social sciences [Kuo *et al.*, 1993], and product design [Glover *et al.*, 1998]. During the past three decades, MEWCP has been studied under many different names such as maximum diversity [Martí *et al.*, 2011; Palubeckis, 2007], maxisum dispersion [Kuby, 1987], MAX-AVG dispersion [Ravi *et al.*, 1994], remote-clique [Chandra and Halldórsson, 2001], maximum edge-weighted subgraph [Macambira, 2003], and dense k-subgraph [Brimberg *et al.*, 2009; Feige *et al.*, 2001].

The computational challenge of the MEWCP has motivated a variety of solution approaches including exact methods, approximation algorithms and metaheuristic methods. Examples of approximation algorithms are described in [Hunting *et al.*, 2001; Feige *et al.*, 2001; Hassin *et al.*, 1997]. These approaches provide a performance guarantee, but do not compete well with other methods in computational testing. Two recent examples of exact methods are described in [Sorensen, 2004; Dijkhuizen and Faigle, 1993; Park *et al.*, 1996]. While these methods have the theoretical advantage of finding optimal solutions to a given problem, their applications are generally limited to problems with no more than 100 vertices.

For larger problem instances, heuristics and metaheuristics are often used to find approximate solutions of good quality with a reasonable computing time. This includes tabu search [Alidaee *et al.*, 2007; Macambira, 2003; Aringhieri *et al.*, 2008; Aringhieri and Cordone, 2011; Wang *et al.*, 2012], iterated tabu search [Palubeckis, 2007], simulated annealing [Kincaid, 1992], iterated greedy algorithm [Lozano *et al.*, 2011], genetic algorithms [Feng *et al.*, 2010], variable neighborhood search [Brimberg *et al.*, 2009], scatter search [Gallego *et al.*, 2009; Gortázar *et al.*, 2010], path-relinking method [de Andrade *et al.*, 2005] and memetic search [Katayama and Narihisa, 2004]. Another approach that has received considerable attention in the solution of the MEWCP is greedy randomized adaptive search procedure (GRASP) [de Andrade *et al.*, 2003; de Andrade *et al.*, 2005; Duarte and Martí, 2007; Ghosh, 1996; Silva *et al.*, 2004; Silva *et al.*, 2007]. Finally, a comprehensive survey of the most significant heuristic and metaheuristic methods for MEWCP can be found in [Aringhieri and Cordone, 2011; Martí *et al.*, 2011].

This paper presents MAMEP, a hybrid metaheuristic algorithm integrating a tabu search procedure with a population-based evolutionary algorithm for solving the maximum diversity problem. The proposed algorithm integrates three complementary key components to ensure the high efficiency of the search process. First, to generate promising new solutions, we introduce a dedicated crossover operator which tries to preserve common vertices that are shared by parent solutions. The design of this crossover operator is motivated by an experimental observation that high quality solutions share a large number of common vertices. Second, to allow the algorithm to explore efficiently the search space around each newly generated solution by crossover, we devise a tabu search optimization procedure which relies on a constrained neighborhood and a dynamic tabu list management strategy. Finally, to maintain the population diversity, we employ a quality-and-distance replacement strategy for population updates.

To assess the performance and the competitiveness of our algorithm in terms of both solution quality and computing efficiency, we provide computational results on a total of 120 MEWCP benchmark instances with up to 5000 vertices, showing that the proposed algorithm achieves highly competitive results with respect to the best existing MEWCP heuristics. Moreover, for 6 large MEWCP instances, the proposed algorithm is able to provide new improved results.

4.2 A hybrid metaheuristic algorithm for MEWCP

4.2.1 Outline of the Memetic Algorithm

Memetic algorithms are known to be an effective approach in solving a number of hard combinatorial optimization problems [Moscato and Cotta, 2003; Hao, 2011; Wu and Hao, 2012e; Wu and Hao, 2013]. Typically, a memetic approach repeatedly alternates between a recombination (or crossover) operator to generate solutions located in promising regions in the search space and a local optimization procedure to search around the newly generated solutions. It is commonly admitted that the success of this approach depends critically on the recombination operator. In order to be effective, the recombination operator must be adapted to the problem being solved and should be able to transmit meaningful features from parents to offspring.

The general procedure of our hybrid metaheuristic algorithm for MEWCP (called

MAMEP) is summarized in Algorithm 4.1. It is composed of four main basic components: a population initialing procedure, a tabu search procedure, a crossover operator and a population management strategy. Starting from an initial population of local optima obtained by the tabu search procedure (Section 4.2.3), MAMEP performs a series of cycles called generations. At each generation, two solutions S^1 and S^2 are randomly chosen in the population to serve as parents. The crossover is then used to produce an offspring solution S^0 from S^1 and S^2 (Section 4.2.5). The tabu search procedure is applied to improve S^0 for a fixed number of iterations (Section 4.2.4). Afterward, the population updating rule decides whether the improved solution S^0 should be inserted into the population and which existing solution should be replaced (Section 4.2.6). This process repeats until a stop condition is verified, such as a time limit or a fixed number of generation (Section 4.3.2). In the following, we describe the four main components of the proposed algorithm.

Algorithm 4.1: Memetic algorithm for the maximum edge weight clique problem

```
Require: An edge weighted graph $G = (V, E)$, cardinality of the clique $m$ ($m < |V|$), population size $p$
Ensure: The best solution $S^*$ found
 1: Initialize population $Pop = \{S_1, ..., S_p\}$ /* Section 4.2.3 */
 2: $S^* \leftarrow Best(Pop)$ /* $S^*$ records the best solution encountered until now */
 3: while Stop condition is not verified do
 4:     Randomly select 2 parent solutions $S^1$ and $S^2$ from $Pop = \{S_1, ..., S_p\}$
 5:     $S^0 = Cross\_Over(S^1, S^2)$ /* Section 4.2.5, generate a new solution from parents */
 6:     $S^0 \leftarrow Tabu\_Search(S^0)$ /* Section 4.2.4, improve the offspring */
 7:     if $f(S^0) > f(S^*)$ then
 8:         $S^* \leftarrow S^0$ /* Update the best solution found so far */
 9:     end if
10:     $Pop \leftarrow PopulationUpdate(S^0, Pop)$ /* Section 4.2.6, update population using a
        distance-and-quality rule */
11: end while
```

4.2.2 Search space and evaluation function

Before presenting the components of the MAMEP algorithm, we define first the search space explored by the algorithm as well as the evaluation function to measure the quality of a candidate solution.

Given that the objective of MEWCP is to determine a subset $M \subset V$ of size m (called m-subset) while maximizing the sum of the weights between every two vertices in M, we define our search space Ω to be the collection of all possible subsets of V of cardinality m, i.e., $\Omega = \{S \subset V : |S| = m\}$. It is clear that Ω has a size of $P(n, m)$ which may be very large for reasonable values of n and m.

To evaluate the quality of a solution $S \in \Omega$ (i.e., a m-subset), we just sum up the weights between every two vertices in S as follows:

$$f(S) = \sum_{u,v \in S, u<v} d_{uv} \tag{4.3}$$

It is easy to see that this function is strictly equivalent to the function defined in Formula (4.1).

Finally, we mention that to represent each subset S of Ω, we use a binary vector $\overline{S}$ of length n containing exactly m 1s: $\overline{S}[i] = 1$ if vertex $s_i \in N$ belongs to subset S, $\overline{S}[i] = 0$ otherwise. For simplicity reasons, hereafter we will use the set notion S, instead of its vector representation $\overline{S}$, to designate a solution of Ω even if both are semantically equivalent.

4.2.3 Generation of initial solutions

Our algorithm begins with an initial population composed of p solutions (p is the population size which is fixed by a parameter). There are different ways to obtain the initial population. One basic technique is random generation which, though easy to apply, can hardly lead to initial solutions of good quality. In this paper, we initialize the population with locally optimal solutions as follows. Starting from a random m-subset S ($S \in \Omega$), we apply the tabu search procedure (see Section 4.2.4) to improve S until a local optimum is reached. The local optimum is then added to the population. This procedure is repeated until the population fills up with $3 \times p$ solutions from which we finally retain the p best ones with the largest objective values to form the initial population. This procedure allows us to obtain an initial population of relatively high quality.

4.2.4 The constrained neighborhood tabu search procedure

One key element of our hybrid MAMEP algorithm is its tabu search procedure which ensures the critical role of intensified search of a limited region. In addition to being applied to generate the initial population as explained in the last section, the tabu procedure is in particular used to improve the offspring solutions created by the crossover operator (see Section 4.2.5). Adopting the general method of tabu search, our tabu procedure (see Algorithm 4.2) is specifically adapted to the MEWCP problem by introducing a dedicated constrained neighborhood and a dynamic tabu list management mechanism, which are developed in this section.

4.2.4.1 Constrained swap move and neighborhood

As explained in Section 4.2.2, our search space Ω is composed of all possible m-subsets from the given vertex set V. To explore this space, one simple and basic way is to start with any initial m-subset S and subsequently swap an vertex of S with another vertex of $V \setminus S$ such that the objective value is improved. One advantage of this swap move is that it maintains solution feasibility. Nevertheless, this unconstrained swap leads to a large neighborhood of size $m.(n-m)$. This unconstrained swap and its associated neighborhood were used in [Ghosh, 1996] and further explored in [Duarte and Martí, 2007]. As indicated in [Duarte and Martí, 2007], although this unconstrained swap is valuable for local search algorithms, the evaluation of the neighborhood can be computationally expensive. This is particularly the case with tabu search given that at each iteration of the algorithm, we wish to select the *best* swap move among all $m.(n-m)$ possible moves induced by S and $V \setminus S$.

To reduce the computing time needed to examine neighboring solutions and improve the computational efficiency of our tabu search procedure, we devise a *constrained* neighborhood which is both more focused and smaller-sized. A similar constrained neighborhood for MCP is explored in Chapter 1 and the basic idea of our constrained neighborhood is to limit the swap move to two specifically identified subsets $X \subseteq S$ and $Y \subseteq V \setminus S$ such that $|X|$ and $|Y|$ are as small as possible, and the resulting neighborhood contains always the best solutions of the unconstrained neighborhood induced by S and $V \setminus S$.

The constrained neighborhood is based on the notion of *contribution* defined for each vertex of the current solution. Precisely, let $S \in \Omega$ be a solution (i.e., a m-subset of V), we define, for each vertex $i \in V$, its contribution d_i to the objective value $f(S)$ as follows:

$$d_i = \sum_{j \in S} d_{ij}, for\ i \in V \tag{4.4}$$

Let $swap(u, v)$ designate the move which swaps $u \in S$ and $v \in V \setminus S$. Then, when $swap(u, v)$ is applied, the objective variation Δ_{uv}, also called the 'move gain', can be conveniently computed by:

$$\Delta_{uv} = f(S') - f(S) = d_v - d_u - d_{uv} \tag{4.5}$$

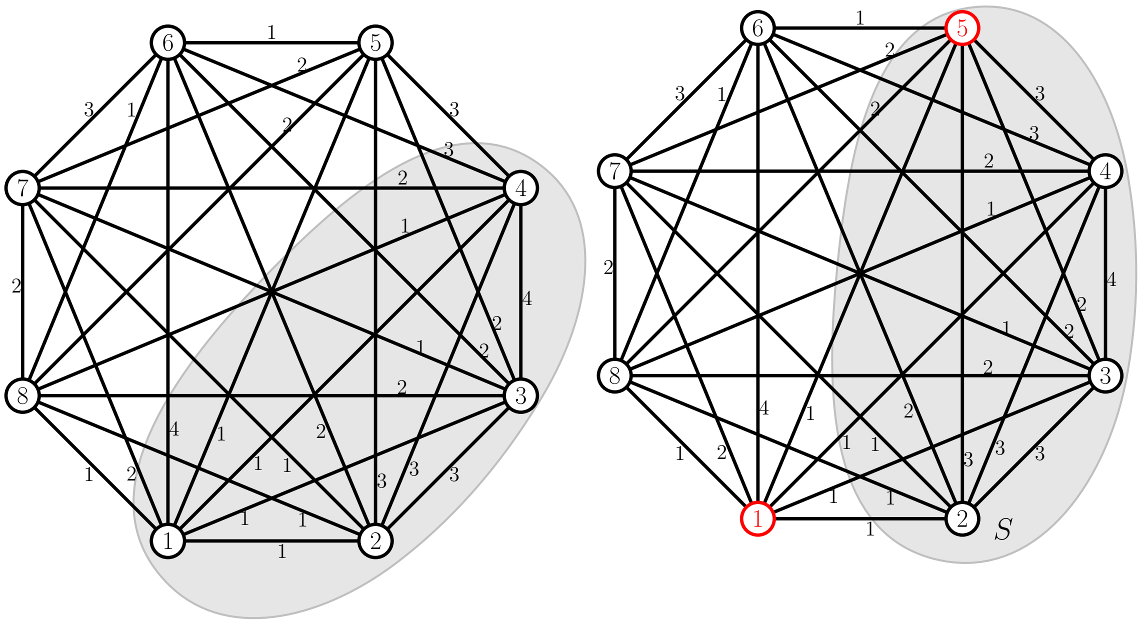
where $S' = S \setminus \{u\} \cup \{v\}$ while d_v and d_u are contributions of v and u according to Formula (4.4).

From Formula 4.5, we observe that the move gain Δ_{uv} of $swap(u, v)$ depends on d_v, d_u and d_{uv}. For the purpose of maximizing the objective function f, we should prefer a vertex $v \in V \setminus S$ with a large contribution and inversely a vertex $u \in S$ with a small contribution. In addition, we also need to consider the weight d_{uv} between u and v. To maximize f, we constraint u to belong to a specific subset $X \subseteq S$ containing the vertices in S with small contributions, and v to belong to a specific subset $Y \subseteq V \setminus S$ including

Algorithm 4.2: Constrained neighborhood tabu search for MEWCP

```
Require: An edge weighted graph $G = (V, E)$, initial solution $S$, number $MaxIter$ of tabu search
    iterations
Ensure: The best solution $S^*$ found and $f(S^*)$
 1: $S^* \leftarrow S$   /* Records the best solution found so far */
 2: $Iter \leftarrow 0$   /* Iteration counter */
 3: Compute the contribution value $d_v$ according to Eq. 4.4 for each vertices $v \in V$.
 4: Initiate the tabu list and tabu tenure
 5: while $Iter < MaxIter$ do
 6:    Identity the two critical subsets $X \subseteq S$ and $Y \subseteq V \setminus S$
 7:    Choose a best allowed $swap(u, v)$ from the constrained neighborhood $CN(S)$ defined by $X$ and $Y$
 8:    $S = S \setminus \{u\} \cup \{v\}$ /* Move to the new solution */
 9:    Update the tabu list and the contribution value $d_v$ for each $v \in V$
10:    if $f(S) > f(S^*)$ then
11:       $S^* \leftarrow S$     /* Update the best solution found so far */
12:    end if
13:    $Iter \leftarrow Iter + 1$
14: end while
```

the vertices in $V \setminus S$ with large contributions. Formally, let *tabu_list* be the tabu list containing the vertices that are currently forbidden for swap (these vertices are excluded for consideration, see Section 4.2.4.2).



$d_1 = 3,\ d_2 = 7,\ d_3 = 8, d_4 = 8,\ d_5 = 9,\ d_6 = 11,\ d_7 = 6,\ d_8 = 5$
$S = \{1,2,3,4\},\ V \setminus S = \{5,6,7,8\},\ dmax = 4$
$d_{MinInS} = 3,\ d_{MaxOutS} = 11,\ X = \{1,2\},\ Y = \{5,6\}$

$\Delta_{15} = 5,\ \Delta_{16} = 4,\ \Delta_{25} = -1, \Delta_{26} = 2$
$swap(1,5)$ *is the best swap move*

Figure 4.1: An example for the constrained neighborhood defined by swap move.

Let $dMinInS = min\{d_i \mid i \in S,\ i \notin tabu\_list\}$ and
Let $dMaxOutS = max\{d_i \mid i \in V \setminus S,\ i \notin tabu\_list\}$.
Then we define subsets X and Y as follows:
$X = \{i \in S \mid i \notin tabu\_list,\ d_i \leq dMinInS + dmax\}$ and
$Y = \{i \in N \setminus S,\ \mid i \notin tabu\_list,\ d_i \geq dMaxOutS - dmax\}$
where $dmax$ is the maximum weight between two vertices in V, i.e., $dmax = max\{d_{ij} \mid 1 \leq i < j \leq |V|\ \}$.

To obtain a neighboring solution S' from S, we swap one vertex $u \in X$ with another vertex $v \in Y$. All possible swap moves induced by X and Y define our constrained neighborhood $CN(S)$, i.e.,

$CN(S) = \{S' \mid S' = S \backslash \{u\} \cup \{v\},\ u \in X, v \in Y\}$.

Fig. 4.1 shows an illustrative example where S has four possible neighboring solutions and we assume that the tabu list is empty (i.e., no vertex is forbidden for swap).

Our tabu search procedure explores the search space Ω by following this constrained neighborhood. At each iteration, instead of examining all the swap moves induced by S and $V \setminus S$, our tabu search first identifies the two subsets X and Y associated to S and then selects the best $swap(u,v)$ ($u \in X$, $v \in Y$) with the highest move gain Δ_{uv} (ties broken randomly). The resulting solution replaces S to becomes the new current solution.

We can see that our constrained neighborhood is a strict subset of the unconstrained neighborhood. Furthermore, assume that S'' is a best non-tabu neighboring solution in the unconstrained neighborhood, it is easy to verify that $S'' \in CN(S)$. In other words, our constrained neighborhood $CN(S)$ contains all the best (non-tabu) neighboring solutions in the unconstrained neighborhood, while its size is generally much smaller than the unconstrained neighborhood.

When a *swap*(u, v) move is performed to give a new solution, the contribution associated with each vertex i in V is updated using the following formula:

$$d_i = \begin{cases} d_i + d_{iv}, & if\ i = u, \\ d_i - d_{iu}, & if\ i = v, \\ d_i + d_{iv} - d_{iu}, & if\ i \neq u \text{ and } i \neq v. \end{cases}$$

Thus, the updating of the contribution values associated with the $|V|$ vertices in V can be performed in linear time $O(n)$.

4.2.4.2 Tabu list and tenure management

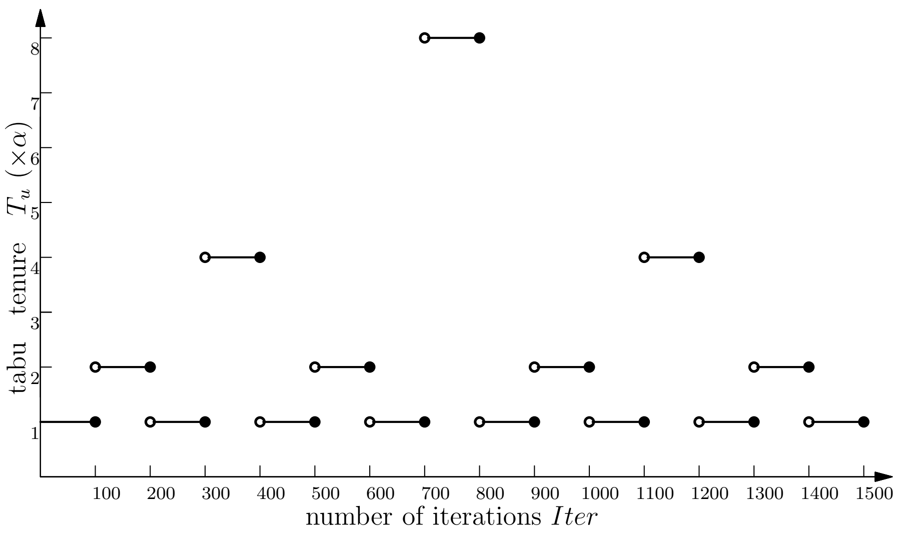


Figure 4.2: An illustration of the step function (one period) used for tuning the tabu tenure T_u.

As previously explained, a neighboring solution of S is obtained by a *swap*(u, v) move. To prevent the search from short-term cycling, when such a move is performed, vertex u is marked tabu for the next T_u (called tabu tenure) iterations, during which u cannot be put back into solution S. Similarly, vertex v is also marked tabu for the next T_v iterations and v cannot be removed from S during this period.

It is well known that the performance of a tabu search algorithm depends on the way the tabu tenure is determined [Glover, 1989]. A too short tabu tenure may leads the search to revisit solutions previously encountered while a too long tabu tenure may exclude high quality solutions during the search. In this chapter, for the MEWCP, we adopt a dynamic tabu list management technique which was first proposed in [Galinier *et al.*, 2011] and recently explored in [Wu and Hao, 2013]. With this technique, the tabu tenure T_u are dynamically adjusted by a periodic step function $T_u(Iter)$ defined over the number of iterations $Iter$: $Iter \rightarrow T_u(Iter)$ where $Iter$ is the number of iterations and $T_u(Iter)$ is the tabu tenure for T_u at iteration $Iter$. Each period of the step function is composed of 1500 iterations divided into 15 steps.

Fig. 4.2 provides an illustration of this step function. As shown in Fig. 4.2, the tabu tenure T_u is equal to α (α is a parameter) for the first 100 iterations [1..100], then $2 \times \alpha$ for iterations from [101..200], followed by α again for iterations [201..300] and $4 \times \alpha$ for iterations [401..500] etc. After reaching the largest value $8 \times \alpha$ for iterations [701..800], T_u drops again to α for the next 100 iterations and so on. This function repeats periodically this variation scheme every 1500 iterations. Similarly, we use the same strategy to tune the tabu tenure T_v. At each iteration of the tabu search, T_v is set equal to $0.7 * T_u$.

One notices that the tabu tenure T_u for u (the vertex leaving S) is larger than the tabu tenure T_v for v (the vertex joining S). This can be explained by the same fact provided in Section 2.1.2.5

4.2.5 Crossover operator

Within the hybrid memetic framework, the crossover operator constitutes another important search operator [Neri *et al.*, 2012]. Its main goal is to create new promising candidate solutions by blending existing parent solutions, a solution being promising if it can potentially lead the search process to new search regions where better solutions may be found. For this reason, crossover plays basically an exploratory role which comes to complement the intensification role of the tabu search procedure. It is well known that even if random crossover operators (uniform, one-point etc.) can be easily applied in the context of binary representation, such a blind operator can rarely guide the search process to effectively explore the most promising search regions. To be effective, a meaningful crossover operator is usually based on some pertinent properties (building blocks) of the given problem and a recombination mechanism to preserve these properties from parents to offspring [Hao, 2011].

To identify "good properties" for MEWCP, we carried out a detailed analysis of samples of locally optimal solutions (see Section 4.4.2). This analysis discloses that high quality solutions share a large number of common vertices that have high chances to be part of an optimal solution. Therefore, given two high quality solutions, it seems pertinent to preserve the shared vertices (building blocks). Our proposed crossover operator follows this idea and operates as follows.

Given 2 parent solutions S^1 and S^2 which are chosen randomly from the population, one offspring solution S^0 is constructed as follows. We build first a partial solution by preserving the common vertices shared by the two selected parents, i.e., $S^0 = S^1 \cap S^2$. Then we complete S^0 to obtain a feasible solution with a greedy procedure based on the contribution values defined in Section 4.2.4.1. The greedy procedure extends S^0 in a step-by-step way by adding at each step one vertex to S^0 until S^0 contains exactly m vertices.

At the first step, we examine all the vertices in $S^1 \setminus S^0$ to identify the vertex with the highest contribution with respect to S^0 and displace it from $S^1 \setminus S^0$ to S^0. Afterward, the vertices in $S^2 \setminus S^0$ are considered, we identify the vertex with the largest contribution in $S^2 \setminus S^0$ and add it to S^0. Then at each step of this greedy procedure, the vertices in $S^1 \setminus S^0$ and $S^2 \setminus S^0$ are considered in turn until S^0 reaches the size of m. This offspring solution is usually of relatively high quality and has approximatively the same distance to its two parent solutions.

4.2.6 Population updating rule

The population updating rule decides whether an offspring solution, which is generated by the crossover operator and further improved by the tabu search procedure, should become a member of the population, and if affirmative, which existing solution of the population should be replaced. Population management is an important issue since the population updating rule impacts directly the population diversity which conditions the convergence of the search algorithm.

In our case, we wish to maintain a healthy diversity of the population during the search. For this purpose, we adopt a updating strategy which takes into account both quality and the distance between the solutions of the population [Lü *et al.*, 2010; Porumbel *et al.*, 2010a]. While the notion of quality can be easily understood with respect to the objective function, we need to formally define the notion of distance between solutions.

Definition 1. Distance between two solutions: Given two solutions S^a and S^b, the distance $dis(S^a, S^b)$ between S^a and S^b is the minimum number of swap moves necessary to transform S^b to S^a, i.e., $dis(S^a, S^b) = m - |S^a \cap S^b|$.

Definition 2. Distance between one solution and a population: Given a population $Pop = \{S^1, ..., S^p\}$ and the distance $dis(S^i, S^j)$ between any two solutions S^i and S^j $(i, j \in \{1, ..., p\}$ and $i \neq j)$, the distance between a solution S^i $(i = 1, ..., p)$ and the population Pop is defined as the minimum distance between S^i and any other solution in the population:

$$D_{S^i,Pop} = min\{dis(S^i, S^j) | S^j \in Pop, S^j \neq S^i\} \tag{4.6}$$

To update the population, we adopt the strategy which was originally proposed in [Lü *et al.*, 2010], and has shown to be very effective in maintaining the balance between the diversity and quality of the population. This strategy uses a quality-and-distance scoring function (denoted by $H(S^i, Pop')$) to rank the solutions of the population $Pop' = Pop \cup \{S^0\}$. Formally, $H(S^i, Pop')$ is defined as follows:

$$H(S^i, Pop') = \beta \widetilde{A}(f(S^i)) + (1 - \beta)\widetilde{A}(D_{S^i,Pop'}) \tag{4.7}$$

where β is a parameter set to 0.6 according to [Lü *et al.*, 2010], $f(S^i)$ is the objective value of S^i, and $\widetilde{A}(.)$ is a normalized function defined as follows:

$$\widetilde{A}(y) = \frac{y - y_{min}}{y_{max} - y_{min} + 1} \tag{4.8}$$

where y_{max} and y_{min} are respectively the maximum and minimum of y in the population *Pop*. "+1" is used for the purpose of avoiding the possibility of a 0 denominator.

Our population updating strategy is described in Algorithm 4.3. To update the population, we first tentatively add S^0 to the population Pop, then we use the scoring function H to identify the solution S^w with the smallest goodness score $H(S^i, Pop')$ in Pop'. If S^w is not the offspring S^0, then the population is updated by replacing S^w with S^0.

4.3 Computational experiments

In this section, we present an extensive assessment of the proposed MAMEP. For this purpose, we show experimental results obtained by MAMEP on a large collection of benchmark instances and make comparisons with the best performing MEWCP algorithms published in the literature.

4.3.1 Benchmark instances and Parameter settings

To evaluate the efficiency of the proposed approach, we carry out extensive experiments on the same sets of 120 test instances as in [Palubeckis, 2007], which are frequently used to assess algorithms for MEWCP. The details of the instance sets are described in Section 1.6.

The settings of the parameters required by our MAMEP algorithm are given in Table 4.1. These parameter values have been determined by performing a preliminary experiment on a set of 20 random *Type I* instances ($Type1\_22$) with $|V| = 2000$ and $m = 200$ and another set of 20 random *Type II* instances ($Type2$) with $|V| = 500$ and $m = 50$. To determine the parameter values, we first began with the parameters required by the TS algorithm ($MaxIter$, α) and then determined the parameters used by the memetic algorithm (p,β). For the TS algorithm, we tested values for α in the range [5..25] and β in the range [5000..200000]. For each instance, we run the TS algorithm for 20 seconds. Table 4.2 reports, for each set of instances and each parameter combination, the average evaluation function value obtained with the TS algorithm.

From Table 4.2, we observe that the combination ($MaxIter = 50000$, $\alpha = 15$) globally leads to the best results, giving the evidence for the parameter settings of our TS algorithm

Algorithm 4.3: Population updating strategy

```
Require: Offspring solution $S^0$ and Population $Pop = \{S^1, ..., S^p\}$
Ensure: Updated population $Pop = \{S^1, ..., S^p\}$
1: Tentatively add $S^0$ to the population: $Pop' = Pop \cup \{S^0\}$
2: for $i = 0, ..., p$ do
3:     Calculate the distance between $S^i$ and $Pop'$ according to Eq. (4.6)
4:     Calculate the goodness score $H(S^i, Pop')$ of $S^i$ according to Eq. (4.7)
5: end for
6: Identify the solution $S^w$ with the smallest goodness score in $Pop'$:
   $S^w = arg\ min\{H(S^i, Pop')|i = 0, ..., p\}$
7: if ($S^w \neq S^0$) then
8:     Replace $S^w$ with $S^0$: $Pop = Pop \cup \{S^0\} \backslash \{S^w\}$
9: end if
```

Table 4.1: Settings of parameters

Parameters	Section	Description	Values
p	4.2.1	size of population	10
α	4.2.4	tabu tenure management factor	15
$MaxIter$	4.2.4	number of TS iterations after recombination	50000
β	4.2.6	goodness score coefficient	0.6 [Lü *et al.*, 2010]

Table 4.2: Parameter tuning

Experiment number	Parameter combination		Type1_22(AVG)	Type2(AVG)
	$MaxIter$	α		
1	5000	5	114238.75	777406.28
2	5000	10	114251.65	777406.28
3	5000	15	114253.25	777406.28
4	5000	20	114248.00	777406.28
5	5000	25	114242.00	777406.28
6	20000	5	114238.15	777406.28
7	20000	10	114251.40	777406.28
8	20000	15	114253.15	777406.28
9	20000	20	114248.30	777406.28
10	20000	25	114240.60	777406.28
11	50000	5	114228.70	777406.28
12	50000	10	114254.45	777406.28
13	50000	15	114255.65	777406.28
14	50000	20	114248.55	777406.28
15	50000	25	114243.25	777406.28
16	100000	5	114234.90	777406.28
17	100000	10	114255.40	777406.28
18	100000	15	114254.45	777406.28
19	100000	20	114248.30	777406.28
20	100000	25	114222.05	777406.28
21	200000	5	114239.50	777406.28
22	200000	10	114250.40	777406.28
23	200000	15	114250.10	777406.28
24	200000	20	114248.15	777406.28
25	200000	25	114233.30	777406.28

in Table 4.1. In addition to $MaxIter$ and α, our algorithm requires two other parameters p and β. We fixed $p = 10$ (small population are often used in memetic algorithms) while $\beta = 0.6$ is chosen according to [Lü *et al.*, 2010]. As we see below, the adopted parameter settings are good and robust enough to allow our algorithm to yield very competitive results for the sets of the tested instances compared with those reported in the literature.

4.3.2 Reference Algorithms and Experimental Protocol

Our MAMEP algorithm is programmed in C and compiled using GNU GCC. All the experiments were carried out on a PC running Windows XP with an Intel Xeon E5440 processor (2.83 GHz and 8G RAM).

In order to evaluate the relative effectiveness and efficiency of our proposed algorithm, we compared our MAMEP algorithm with 4 recent and best-performing MEWCP algorithms in the literature:

- ITS: Iterated robust tabu search algorithm (2007) [Palubeckis, 2007].
- VNS: Variable neighborhood search algorithm (2009) [Brimberg *et al.*, 2009].
- TIG: An fine-tuning iterated greedy algorithm (2011) [Lozano *et al.*, 2011].

Table 4.3: Experimental Protocol

Instance Family	Time limit(s)	Independent runs
Silva	20	30
Type1_55	20	30
Type1_22	20	30
Type1_52	20	30
Type2	20	30
b2500	300	30
p3000	600	15
p5000	1800	15

- LTS-EDA: Robust learnable tabu search guided by estimation of distribution algorithm (2012) [Wang *et al.*, 2012].

As reviewed and compared in the most recent survey [Martí *et al.*, 2011], ITS and VNS seem to be the most powerful algorithms for the MEWCP among 30 heuristic algorithms. TIG and LTS-EDA are two recently proposed algorithms and thus not included in the recent review [Martí *et al.*, 2011]. However, experimental results show that TIG and LTS-EDA obtain better or competitive performance than VNS and ITS. Especially, LTS-EDA is able to reach new best solutions [Wang *et al.*, 2012] for some larger random instances. Thus, these 4 reference algorithms are among the most successful approaches for solving MEWCP actually available in the literature.

Moreover, these 4 reference algorithms are tested and compared very recently in [Wang *et al.*, 2012] under the same time limit on a Pentium(R) with 2.70 GHz CPU and 2.0 G RAM. For a fair comparison, we use for our MAMEP algorithm the same running (stopping) conditions as described in [Wang *et al.*, 2012], which are summarized in Table 4.3.

4.3.3 Computational results

Tables 4.4 to 4.10 respectively show the computational statistics of the MAMEP algorithm on the 7 sets of 120 benchmark instances. In all tables, columns 1 and 2 respectively give the name of the instance and the previous best objective values (f_{prev}). Note that in Tables 4.4 to 4.10, the previous best objective values (f_{prev}) are compiled from Tables 1-7 from [Palubeckis, 2007] and Tables 6 and 11 from [Wang *et al.*, 2012]. To the best of our knowledge, they are the current best known results for these 120 problem instances.

Columns 3 to 6 give our results: the best objective value (f_{best}), the best solution gap to the previous best known evaluation function values g_{best}(i.e., $f_{prev} - f_{best}$), the average solution gap to the previous best evaluation function values g_{avg} (i.e., $f_{prev} - f_{avg}$) (where f_{avg} represents the average objective value), and the standard deviation over the tested runs.

From Tables 4.4 to 4.10, we observe that for *all* these 120 instances, our MAMEP algorithm can reach the previous best known results within the time limits given in Table 4.3. Specifically, for 101 out of 120 instances (84%), MAMEP has a successful rate of 100%, attaining the best known objective value for each of its runs. For the 19 cases (16%) where the successful rate is below 100%, the standard deviations remain small ranging from 0.17 to 27.93. More importantly, for 6 random large and very challenging instances

Table 4.4: Performance of MAMEP on the 20 Silva instances

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
Silva_100_10	333	333	0	0	0
Silva_100_20	1195	1195	0	0	0
Silva_100_30	2457	2457	0	0	0
Silva_100_40	4142	4142	0	0	0
Silva_200_20	1247	1247	0	0	0
Silva_200_40	4450	4450	0	0	0
Silva_200_60	9437	9437	0	0	0
Silva_200_80	16,225	16,225	0	0	0
Silva_300_30	2694	2694	0	0	0
Silva_300_60	9689	9689	0	0	0
Silva_300_90	20,743	20,743	0	0	0
Silva_300_120	35,881	35,881	0	0	0
Silva_400_40	4658	4658	0	0	0
Silva_400_80	16,956	16,956	0	0	0
Silva_400_120	36,317	36,317	0	0	0
Silva_400_160	62,487	62,487	0	0	0
Silva_500_50	7141	7141	0	0	0
Silva_500_100	26,258	26,258	0	0	0
Silva_500_150	56,572	56,572	0	0	0
Silva_500_200	97,344	97,344	0	0	0

Table 4.5: Performance of MAMEP on the 20 random Type1_55 instances ($|V| = 500; m = 50$)

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
Type1_55.1	7833.83	7833.83	0	0	0
Type1_55.2	7771.66	7771.66	0	0	0
Type1_55.3	7759.36	7759.36	0	0	0
Type1_55.4	7770.24	7770.24	0	0	0
Type1_55.5	7755.23	7755.23	0	0	0
Type1_55.6	7773.71	7773.71	0	0	0
Type1_55.7	7771.73	7771.73	0	0	0
Type1_55.8	7750.88	7750.88	0	0	0
Type1_55.9	7770.07	7770.07	0	0	0
Type1_55.10	7780.35	7780.35	0	0	0
Type1_55.11	7770.95	7770.95	0	0	0
Type1_55.12	7757.65	7757.65	0	0	0
Type1_55.13	7798.43	7798.43	0	0	0
Type1_55.14	7795.63	7795.63	0	0	0
Type1_55.15	7736.84	7736.84	0	0	0
Type1_55.16	7792.77	7792.77	0	0	0
Type1_55.17	7787.20	7787.20	0	0	0
Type1_55.18	7756.26	7756.26	0	0	0
Type1_55.19	7755.41	7755.41	0	0	0
Type1_55.20	7733.86	7733.86	0	0	0

Table 4.6: Performance of MAMEP on the 20 random Type1_52 instances ($|V| = 500; m = 200$)

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
Type1_52.1	107394.58	107394.58	0	0	0
Type1_52.2	107251.75	107251.75	0	0	0
Type1_52.3	107260.39	107260.39	0	0	0
Type1_52.4	107010.90	107010.90	0	0	0
Type1_52.5	106944.55	106944.55	0	0	0
Type1_52.6	107167.36	107167.36	0	0	0
Type1_52.7	107079.44	107079.44	0	0	0
Type1_52.8	107077.45	107077.45	0	0	0
Type1_52.9	107482.71	107482.71	0	0	0
Type1_52.10	107265.81	107265.81	0	0	0
Type1_52.11	107193.08	107193.08	0	0	0
Type1_52.12	106853.46	106853.46	0	0	0
Type1_52.13	107647.28	107647.28	0	0	0
Type1_52.14	107427.17	107427.17	0	0	0
Type1_52.15	107054.79	107054.79	0	0	0
Type1_52.16	107420.66	107420.66	0	0	0
Type1_52.17	107111.01	107111.01	0	0	0
Type1_52.18	107006.35	107006.35	0	0	0
Type1_52.19	107052.95	107052.95	0	0	0
Type1_52.20	106815.65	106815.65	0	0	0

Table 4.7: Performance of MAMEP on the 20 random Type1_22 instances ($|V| = 2000; m = 200$)

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
Type1_22.1	114,271	114,271	0	2.83	2.29
Type1_22.2	114,327	114,327	0	0	0
Type1_22.3	114,195	114,195	0	5.46	5.36
Type1_22.4	114,093	114,093	0	7.13	7.65
Type1_22.5	114,196	114,196	0	22.23	24.09
Type1_22.6	114,265	114,265	0	5.70	8.87
Type1_22.7	114,361	114,361	0	0.03	0.17
Type1_22.8	114,327	114,327	0	1.26	1.26
Type1_22.9	114,199	114,199	0	0	0
Type1_22.10	114,229	114,229	0	3.33	6.39
Type1_22.11	114,214	114,214	0	9.16	8.27
Type1_22.12	114,214	114,214	0	10.30	7.10
Type1_22.13	114,233	114,233	0	2.66	4.41
Type1_22.14	114,216	114,216	0	0.23	0.76
Type1_22.15	114,240	114,240	0	0.83	0.63
Type1_22.16	114,335	114,335	0	2.36	4.23
Type1_22.17	114,255	114,255	0	1.43	2.99
Type1_22.18	114,408	114,408	0	0.53	1.02
Type1_22.19	114,201	114,201	0	0	0
Type1_22.20	114,349	114,349	0	0.60	1.80

Table 4.8: Performance of MAMEP on the 20 random Type II instances ($|V| = 500; m = 50$)

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
Type2.1	778030.57	778030.57	0	0	0
Type2.2	779963.54	779963.54	0	0	0
Type2.3	776768.17	776768.17	0	0	0
Type2.4	775394.47	775394.47	0	0	0
Type2.5	775610.96	775610.96	0	0	0
Type2.6	775153.58	775153.58	0	0	0
Type2.7	777232.88	777232.88	0	0	0
Type2.8	779168.62	779168.62	0	0	0
Type2.9	774802.05	774802.05	0	0	0
Type2.10	774961.12	774961.12	0	0	0
Type2.11	777468.78	777468.78	0	0	0
Type2.12	775492.89	775492.89	0	0	0
Type2.13	780191.78	780191.78	0	0	0
Type2.14	782232.68	782232.68	0	0	0
Type2.15	780300.33	780300.33	0	0	0
Type2.16	775436.19	775436.19	0	0	0
Type2.17	776618.99	776618.99	0	0	0
Type2.18	775850.64	775850.64	0	0	0
Type2.19	778802.82	778802.82	0	0	0
Type2.20	778644.65	778644.65	0	0	0

Table 4.9: Performance of MAMEP on the 10 Beasley instances ($|V| = 2500; m = 1000$)

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
b2500-1	1153,068	1153,068	0	0	0
b2500-2	1129,310	1129,310	0	7.46	27.93
b2500-3	1115,538	1115,538	0	0	0
b2500-4	1147,840	1147,840	0	0	0
b2500-5	1144,756	1144,756	0	0	0
b2500-6	1133,572	1133,572	0	0	0
b2500-7	1149,064	1149,064	0	0	0
b2500-8	1142,762	1142,762	0	0	0
b2500-9	1138,866	1138,866	0	0	0
b2500-10	1153,936	1153,936	0	0	0

Table 4.10: Performance of MAMEP on the 10 large random instances ($m = 0.5 * |V|$)

Instance	f_{prev}	MAMEP Algorithm			
		f_{best}	g_{best}	g_{avg}	standard deviation
p3000_1	6502,308	6502,330	-22	25.20	38.74
p3000_2	18272,568	18272,568	0	0	0
p3000_3	29867,138	29867,138	0	0	0
p3000_4	46915,044	46915,044	0	0	0
p3000_5	58095,467	58095,467	0	0	0
p5000_1	17509,215	17509,369	-154	-121.00	84.82
p5000_2	50102,729	50103,092	-363	-338.60	8.68
p5000_3	82039,686	82040,316	-630	-462.27	135.46
p5000_4	129413,112	129413,710	-598	-500.40	119.53
p5000_5	160597,781	160598,156	-375	-254.74	73.22

Table 4.11: Comparison of MAMEP with four best performing MEWCP algorithms in the literature.

Instance	f_{pre}	MAMEP		ITS		VNS		TIG		LTS-EDA	
		g_{best}	g_{avg}	g_{best}	g_{avg}	g_{best}	g_{avg}	g_{best}	g_{avg}	g_{best}	g_{avg}
Type1_22.1	114,271	0	2.83	65	209.87	48	150.60	48	101.57	5	60.73
Type1_22.2	114,327	0	0	29	262.27	0	168.87	0	69.90	0	89.87
Type1_22.3	114,195	0	5.46	69	201.40	19	110.83	5	117.77	0	98.97
Type1_22.4	114,093	0	7.13	22	200.53	70	188.13	58	141.93	0	79.87
Type1_22.5	114,196	0	22.23	95	273.27	87	184.10	99	194.70	51	134.47
Type1_22.6	114,265	0	5.70	41	168.17	30	99.30	9	96.20	0	40.17
Type1_22.7	114,361	0	0.03	12	167.47	0	56.30	0	71.27	0	18.20
Type1_22.8	114,327	0	1.26	25	256.40	0	163.33	0	193.60	0	159.10
Type1_22.9	114,199	0	0	9	139.83	16	78.47	16	80.37	0	70.97
Type1_22.10	114,229	0	3.33	24	204.93	7	139.33	35	121.43	0	56.20
Type1_22.11	114,214	0	9.16	74	237.77	42	145.13	59	139.57	3	69.87
Type1_22.12	114,214	0	10.30	55	249.53	95	143.30	88	156.00	15	84.93
Type1_22.13	114,233	0	2.66	93	279.87	22	168.07	42	167.40	6	85.30
Type1_22.14	114,216	0	0.23	92	248.50	117	194.30	64	202.80	0	81.00
Type1_22.15	114,240	0	0.83	11	117.50	1	62.87	6	80.53	0	22.03
Type1_22.16	114,335	0	2.36	11	225.40	42	215.43	35	167.90	0	36.47
Type1_22.17	114,255	0	1.43	56	217.53	0	170.00	18	144.53	6	57.07
Type1_22.18	114,408	0	0.53	46	169.97	0	57.10	2	117.37	2	22.83
Type1_22.19	114,201	0	0	34	243.20	0	124.60	0	144.37	0	35.87
Type1_22.20	114,349	0	0.60	151	270.67	65	159.43	45	187.23	0	95.40
b2500-1	1153,068	0	0	624	3677.33	96	1911.93	42	1960.20	0	369.20
b2500-2	1129,310	0	7.46	128	1855.33	88	1034.33	1096	1958.47	154	454.53
b2500-3	1115,538	0	0	316	3281.93	332	1503.67	34	2647.87	0	290.40
b2500-4	1147,840	0	0	870	2547.93	436	1521.07	910	1937.13	0	461.73
b2500-5	1144,756	0	0	356	1800.27	0	749.40	674	1655.87	0	286.07
b2500-6	1133,572	0	0	250	2173.47	0	1283.53	964	1807.60	80	218.00
b2500-7	1149,064	0	0	306	1512.60	116	775.47	76	1338.73	44	264.60
b2500-8	1142,762	0	0	0	2467.73	96	862.47	588	1421.53	22	146.47
b2500-9	1138,866	0	0	642	2944.67	54	837.07	658	1020.60	6	206.33
b2500-10	1153,936	0	0	598	2024.60	278	1069.40	448	1808.73	94	305.27
p3000_1	6502,308	-22	25.20	444	1465.53	251	887.80	114	692.67	74	345.93
p3000_2	18272,568	0	0	0	1321.60	0	924.20	0	991.07	140	387.00
p3000_3	29867,138	0	0	1442	2214.73	328	963.53	820	1166.13	0	304.33
p3000_4	46915,044	0	0	1311	2243.93	254	1068.47	426	2488.20	130	317.07
p3000_5	58095,467	0	0	423	1521.60	0	663.00	278	1353.27	0	370.40
p5000_1	17509,215	-154	-121.00	2046	3410.93	848	1817.27	1000	2391.80	37	417.00
p5000_2	50102,729	-363	-338.60	2568	4444.80	1136	2277.00	183	2172.73	184	550.80
p5000_3	82039,686	-630	-462.27	4822	7612.33	1284	3064.40	1526	5377.13	74	828.53
p5000_4	129413,112	-598	-500.40	1032	4478.90	915	2367.90	1098	3276.80	260	677.20
p5000_5	160597,781	-375	-254.74	1682	4058.90	816	1903.30	914	1753.90	204	642.90

(p3000_1, p5000_1, p5000_2, p5000_3, p5000_4 and p5000_5), our MAMEP algorithm is able to improve on the previous best objective values.

4.3.4 Comparison with other algorithms

In order to further evaluate our MAMEP algorithm, in this section we compare our results with four MEWCP algorithms in the literature: ITS [Palubeckis, 2007], VNS [Brimberg *et al.*, 2009], TIG [Lozano *et al.*, 2011] and LTS-EDA [Wang *et al.*, 2012]. As stated previously in Section 4.3.2, these 4 reference algorithms are the best performing approaches for MEWCP currently available.

Table 4.11 shows the best and average results of our MAMEP algorithm compared with the reference algorithms. The results of these 4 reference algorithms are compiled from Tables 6 and 11 from [Wang *et al.*, 2012]. Note that the results of all these algorithms are obtained under the same time limit (see Section 4.3.2). Table 4.11 summarizes the solution difference between the best objective values and the average objective values obtained by these 5 algorithms with the best known objective values on the 40 large size benchmark instances with 2,000 to 5,000 vertices.

Table 4.12: Comparative results of MAMEP and MTS on 10 instances for the maximum diversity problem

Instance	f_{pre}	MAMEP		TS	
		g_{best}	g_{avg}	g_{best}	g_{avg}
Type1_22.1	114,271	0	0	50	110.45
Type1_22.2	114,327	0	0	35	121.35
b2500-1	1153,068	0	278.50	43476	49295.30
b2500-3	1115,538	0	176.00	41840	48236.00
b2500-5	1144,756	0	7.90	40390	47986.70
b2500-7	1149,064	0	129.00	41860	47943.60
p3000_1	6502,308	-22	163.15	65751	71014.15
p3000_3	29867,138	0	253.70	126700	134585.60
p5000_2	50102,729	-285	1220.15	727769	757700.85
p5000_3	82039,686	-414	2293.95	848126	863409.20

From Table 4.11, it may be observed that the MAMEP algorithm outperforms the 4 reference algorithms, named ITS, VNS, TIG and LTS-EDA. In terms of the best solution, MAMEP matches the best known values on 34 instances and finds new best solutions for 6 out of the 40 instances, while ITS, VNS , TIG and LTS-EDA matched the best known solutions on 2, 10, 5, 19 instances respectively. Concerning the average solution value, the results of our MAMEP algorithm remain competitive when compared with these 4 reference algorithms. Indeed, for each of these 40 instances, our MAMEP algorithm is able to reach an average solution value better than each of those 4 reference algorithms.

4.4 Analysis of MAMEP

4.4.1 Influence of crossover

As indicated in Section 4.2.5, our proposed MAMEP algorithm employs a dedicated crossover operator which tries to preserve common vertices that are shared by parent solutions. In order to be sure this crossover operator is meaningful to the maximum edge weight clique problem, we carried out additional experiments to examine the influence of the crossover operator. For this purpose, we compare the performance of the MAMEP algorithm with its underlining TS algorithm. Furthermore, in order to highlight the role of the crossover operators, we weaken the underlining TS of MAMEP by reducing the number of tabu search iterations to $MaxIter = 500$.

Experiments were carried out on a selection of 10 MEWCP instances (Type1_22.1, Type1_22.2, b2500-1, b2500-3, b2500-5, b2500-7, p3000_1, p3000_3, p5000_2 and p5000_3). To solve each instance, we run both methods 20 times under exactly the same timeout limit, which was set to be 300 seconds for each run. In order not to penalize the TS algorithm, we use a multi-start technique to restart it every 500 iterations whenever the timeout limit is not reached.

Table 4.12 presents the comparative results between MAMEP and TS on the 10 instances. For each instance, the following statistics are provided: the best solution gap to the previous best known objective values g_{best} (i.e., $f_{prev} - f_{best}$) and the average solution gap to the previous best objective value g_{avg} (i.e., $f_{prev} - f_{avg}$). From Table 4.12, we observe that for each of these 10 instances, MAMEP performs far better than its underlining TS in terms of both best and average solution values. Furthermore, we also note that although the performance of its underlining TS algorithm is poor, the results of the hybrid MAMEP algorithm remain competitive compared with the current best known results. Indeed, for 3 instances, the MAMEP algorithm is able to improve on the current best known results. This further confirms our conclusion that the crossover operator makes an interesting contribution to the overall performance of the hybrid algorithm.

4.4.2 Structural similarity analysis and motivation for the proposed crossover operator

In the last section, we demonstrated that our proposed crossover operator makes a meaningful contribution to the overall performance of the hybrid algorithm. In this section, we provide empirical motivations for this crossover operator. For this purpose, we show an analysis on the structural similarity between local optima of various quality in terms of the size of the commonly shared vertices. For two local optima S^t and S^s, we define their similarity as $sim(S^t, S^s) = \frac{|S^t \cap S^s|}{m}$. We can see that the larger the similarity between two solutions, the more common vertices are shared by them.

To analyze structural similarity between local optima, we employ the 10 problem instances used in the last section. For each instance, we collect 1000 local optima of different quality using our memetic algorithm as well as its underlying tabu search. Among these 1000 local optima, we select the top 10% (100) with the largest objective values and call them 'high-quality solutions'. Similarly, we take the bottom 10% (100) with the smallest objective values and call them 'low-quality solutions'.

Table 4.13 contains the data related to the similarity between our 1000 local optima. Columns S_{hq}, S_{all} and S_{lo} report respectively the average degree of similarity between the 100 high-quality solutions, the average degree of similarity between all the 1000 sampled local optima, and the average degree of similarity between the 100 low-quality solutions. From Table 4.13, we observe that in most cases, the degree of similarity between high-quality solutions is generally very large, from 0.52 to 0.89. High similarity indicates high quality solutions share a large number of common vertices. Assume that high-quality solutions are close to an optimal solution or could themselves be optimal solutions. As a consequence, it is wise for a recombination operator to preserve the common vertices shared by two (or more) high quality solutions. This is exactly what the proposed crossover operator undertakes to do.

4.4.3 Population Updating Strategy

As stated in Section 4.2.6, our MAMEP algorithm relies on a quality-and-distance replacement strategy (denoted by $DisQual$) for population updates to maintain the population diversity. In order to assess this strategy, we compare it with a traditional strategy (denoted by $PoolWorst$) which simply replaces the worst solution of the population by the new offspring solution. We show experimental evidences on a large random instance p3000.1 (which proves to be one of the most difficult instances for most algorithms). The

Table 4.13: Analysis of structural similarity between high-quality solutions for 10 MEWCP instances

Instance	S_{hq}	S_{all}	S_{lo}
Type1_22.1	0.63	0.20	0.14
Type1_22.2	0.52	0.19	0.14
b2500-1	0.81	0.62	0.44
b2500-3	0.77	0.63	0.44
b2500-5	0.79	0.63	0.44
b2500-7	0.82	0.64	0.44
p3000_1	0.86	0.68	0.53
p3000_3	0.89	0.69	0.53
p5000_2	0.86	0.59	0.51
p5000_3	0.84	0.59	0.52

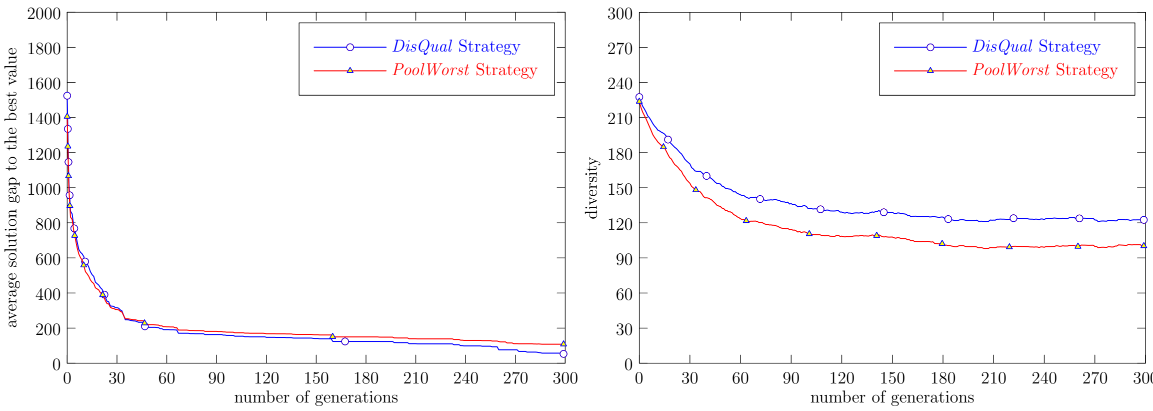


Figure 4.3: Comparison between two population updating strategies

stopping condition is the number of generations (i.e., the number of applications of the crossover operation) which is limited to 300.

Keeping other ingredients unchanged in the MAMEP algorithm, we observe two evolution profiles of each population updating strategy: a) the best solution gap to the previous best known objective value (over 20 runs) vs. the number of generations (Fig. 4.3, left); b) the population diversity vs. the number of generations (Fig. 4.3, right). The population diversity is defined as the average distance between each pair of solutions in the population (see Section 4.2.6).

From Fig. 4.3 (left), we observe that MAMEP with our distance-and-quality based population updating strategy converges more quickly towards high quality solutions than with the traditional pool updating $PoolWorst$ strategy. In addition, from Fig. 4.3 (right), we can see that the $DisQual$ strategy preserves better the population diversity during the evolution process than the $PoolWorst$ strategy, which is tightly correlated to the evolution of the solution quality. We can conclude that considering the diversity of the population as well as the quality for pool updating contributes favorably to the performance of our algorithm.

4.5 Conclusions

This chapter deals with the maximum edge weight clique problem, which is another important generalization of MCP. To approximate the MEWCP, we proposed a hybrid memetic algorithm (MAMEP) mixing a dedicated crossover operator and a constrained neighborhood tabu search procedure. The proposed crossover operator tries to preserve the vertices shared by the parent solutions which hopefully belong to the optimal solution. Offspring solutions are improved with the tabu search optimization procedure which relies on a constrained neighborhood. To maintain a healthy population diversity, MAMEP applies a pool updating strategy that considers both the quality of an offspring solution and its distance to the solutions of the population.

Experimental evaluations on a large collection of 7 sets of 120 instances from the literature showed that our MAMEP algorithm attains consistently the previous best known results within a time limit ranging from 20 seconds (for problems with 100 to 500 vertices) to 30 minutes (for instances with 5000 vertices). Specifically, for 101 out of 120 cases (84%), MAMEP reaches the previous best known objective value for each of its runs (a successful rate of 100%). More importantly, for 6 large and very challenging instances (p3000_1, p5000_1, p5000_2, p5000_3, p5000_4 and p5000_5), our MAMEP algorithm is able to yield improved solutions with respect to the current best known results. We also compared MAMEP with 4 best performing MEWCP algorithms published recently (2007, 2009, 2011, 2012) and showed that MAMEP dominates these reference algorithms in terms of solution quality under comparable experimental conditions.

We also investigated the impact of several essential components of MAMEP. We carried out experiments to demonstrate the beneficial role of the proposed crossover operator and showed an analysis of structural similarity between local optima which provides motivations for the crossover. Moreover, we demonstrated the important role of the distance-

and-quality pool updating strategy which allows MAMEP to maintain a useful population diversity.

In the previous three chapters, we dealt with the maximum clique problem and its two important generalizations and developed highly effective heuristic approaches to approximate these problems. In the next chapters, we will apply these approaches to solve some hard combinatorial optimization problems. As we will see in the these chapters, approaches based on our methods for MCP as well as its generalizations are able to attain highly competitive results for these hard combinatorial optimization problems.

Chapter 5

Applications in graph coloring: A heuristic approach for coloring large graphs based on independent set extraction

In this chapter, we apply the proposed adaptive multistart tabu search approach (AMTS) for the MCP (Chapter 2) to the graph coloring problem, and present an effective heuristic approach (denoted by EXTRACOL) to coloring large graphs. The proposed approach uses a preprocessing method to extract large independent sets from the graph and a memetic algorithm to color the residual graph. Each preprocessing application identifies with AMTS a number of pairwise disjoint independent sets of a given size in order to maximize the vertices removed from the graph. We evaluate EXTRACOL on the 11 largest graphs (with 1000 to 4000 vertices) of the DIMACS challenge benchmarks and show improved results for 4 very difficult graphs (DSJC1000.9, C2000.5, C2000.9, C4000.5). The behavior of the proposed algorithm is also analyzed. The content of this chapter is detailed in [Wu and Hao, 2012a].

Contents

5.1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A legal k-coloring of G corresponds to a partition of V into k independent sets. Graph coloring aims at finding the smallest k for a given graph G (its chromatic number $\chi(G)$) such that G has a legal k-coloring. The graph coloring problem is a well-known NP-hard combinatorial optimization problem [Garey and Johnson, 1979]. Prominent applications of graph coloring include crew scheduling [Gamache *et al.*, 2007], computer register allocation [Werra *et al.*, 1999], timetabling and scheduling [Burke *et al.*, 2007], radio frequency assignment [Smith *et al.*, 1998], printed circuit board testing [Garey and Johnson, 1976], manufacturing [Glass, 2002] and satellite range scheduling [Zufferey *et al.*, 2008]. Exact solution methods can solve problems of relatively small size, heuristics are preferred to handle larger graphs. A comprehensive survey of the most significant heuristic methods can be found in [Galinier and Hertz, 2006].

Local search has been repeatedly applied to graph coloring. Well-known examples include the seminal TabuCOL algorithm [Hertz and de Werra, 1987], Simulated Annealing [Johnson *et al.*, 1991], GRASP [Laguna and Martí, 2001], Iterated Local Search [Chiarandini and Stützle, 2002], Neighborhood Search [Avanthay *et al.*, 2003], Reactive Partial Tabu Search [Blöchliger and Zufferey, 2008], Variable Space Search [Hertz *et al.*, 2008] and Clustering-Guided Tabu Search [Porumbel *et al.*, 2010c]. Local search coloring algorithms are usually simple and have achieved satisfactory performance on the standard DIMACS graphs. They are also often used as a key component of more sophisticated hybrid algorithms.

Indeed, it was observed that some graphs, especially large random graphs, cannot be colored efficiently by using pure local search algorithms. Several hybrid approaches have been proposed as a very interesting alternative. Examples of classical hybrid algorithms are presented in [Costa *et al.*, 1995; Dorne and Hao, 1998; Fleurent and Ferland, 1996; Galinier and Hao, 1999; Hamiez and Hao, 2002; Morgenstern, 1996]. More recent and powerful algorithms can be found in [Galinier *et al.*, 2008; Lü and Hao, 2010; Malaguti *et al.*, 2008; Porumbel *et al.*, 2010b; Xie and Liu, 2009].

Another approach for dealing with large graphs is based on a general principle of "reduce-and-solve". This approach consists in applying, prior to the phase of graph coloring, a preprocessing procedure to extract large independent sets from the graph until the remaining graph (called residual graph) becomes sufficiently small and then coloring the residual graph with any coloring algorithm. Typically, large independent sets are extracted and removed from G in an iterative manner by identifying one largest possible independent set each time. This approach was used with success by some prominent classical algorithms [Chams *et al.*, 1987; Fleurent and Ferland, 1996; Hertz and de Werra, 1987; Johnson *et al.*, 1991; Morgenstern, 1996]. Yet surprisingly we observe that it is rarely employed nowadays.

In this chapter, we revisit the graph coloring approach using independent set extraction and apply our proposed adaptive multistart tabu search approach (AMTS) for the MCP to the graph coloring problem. Furthermore, we develop an improved preprocessing procedure which is able to pack more vertices than with the conventional one-by-one extraction strategy with the same number of color classes. This generates smaller residual graphs that tend to be easier to color. Basically, instead of extracting independent sets one by one, Our improved preprocessing procedure tries to identify, at each preprocessing step, a maximal set of pairwise disjoint independent sets. The rationale behind this approach is that by extracting many pairwise disjoint independent sets each time, more vertices are removed from the initial graph, making, hopefully, the residual graph easier to color. For both tasks of identifying an individual independent set and pairwise disjoint independent sets, we employ the AMTS algorithm proposed in Chapter 2 for the MCP.

Our large graph coloring algorithm (denoted by EXTRACOL) combines this improved independent set extraction preprocessing with a recent memetic coloring algorithm (MACOL [Lü and Hao, 2010]). We evaluate the performance of EXTRACOL on the 11 largest DIMACS benchmark graphs (with 1000, 2000 and 4000 vertices) and present new results for four very hard instances (DSJC1000.9, C2000.5, C2000.9, C4000.5), with respectively 1, 2, 4 and 11 colors below the currently best colorings.

5.2 Review of graph coloring based on maximum independent set

As observed in [Hertz and de Werra, 1987; Fleurent and Ferland, 1996; Johnson *et al.*, 1991], it is difficult, if not impossible, to find a k-coloring of a large graph G (e.g. $|V| \geq 1000$) with k close to $\chi(G)$ by applying directly a given algorithm α on G. To color large and very large graphs, a possible approach is to apply a preprocessing to extract i large independent sets from the graph to obtain a much smaller residual graph which is expected to be easier than the initial graph. A coloring algorithm α is then invoked to color the residual graph. Since each of the i independent sets forms a color class, i plus the number of colors needed for the residual graph gives an upper bound on $\chi(G)$.

The conventional methods for the preprocessing phase operate greedily by extracting one independent set each time from the graph G. For instance, in [Chams *et al.*, 1987; Hertz and de Werra, 1987], the authors uses a tabu search algorithm to identify a large independent set and then remove the vertices of the independent set as well as their incident edges from G. In [Fleurent and Ferland, 1996], the authors refine the choice of the independent set to be removed and prefer an independent set that is connected to as many vertices as possible in the remaining graph. The selected independent set can then remove as many edges as possible with the vertices of the independent set, hence, tends to make the residual graph easier to color.

In the context of graph coloring, several heuristics have been used to identify large independent set in the literature. For instance, in [Chams *et al.*, 1987], the authors apply a simple greedy algorithm. In [Fleurent and Ferland, 1996; Hertz and de Werra, 1987], the authors propose a dedicated tabu search algorithm. In [Johnson *et al.*, 1991], the authors introduce the XRLF method that combines a randomized greedy search with an exhaustive search.

For the subsequent residual graph coloring, various methods have been used including exhaustive search [Johnson *et al.*, 1991], tabu search [Hertz and de Werra, 1987], simulated annealing [Chams *et al.*, 1987] and hybrid genetic tabu search [Fleurent and Ferland, 1996].

The performance of this coloring approach depends on the method to find large independent sets and the method to color the residual graph. It also depends significantly on how independent sets are selected by the preprocessing phase.

In the next section, we present our preprocessing method which tries to extract a large number of pairwise disjoint independent sets from the graph at each preprocessing iteration.

5.3 EXTRACOL: an algorithm for large graph coloring

The proposed EXTRACOL algorithm follows the general schema presented in the previous section and is composed of two sequential phases: a *preprocessing phase* and a *coloring phase*. What distinguishes our work from the existing methods is the way in which the preprocessing phase is carried out. In this section, we explain the basic rationale and illustrate the main techniques implemented. The coloring algorithm applied to the residual graph (MACOL [Lü and Hao, 2010]) is also briefly reviewed at the end of the section.

5.3.1 General procedure

As stated previously, a legal k-coloring of a given graph $G = (V, E)$ corresponds to a partition of V into k independent sets. Suppose that we want to color G with k colors. Assume now that we extract $t < k$ independent sets $I_1, ..., I_t$ from G. It is clear that if we could maximize the number of vertices covered by the t independent sets (i.e. $|I_1 \cup ... \cup I_t|$ is as large as possible), we would obtain a residual graph with fewer vertices when $I_1 \cup ... \cup I_t$ are removed from G. This could in turn help to ease the coloring of the residual graph using $k - t$ colors because there are fewer vertices left in the residual graph.

For this reason, for a given graph G, each time a maximum (or large) independent set is identified we try to maximize the number of *pairwise disjoint* independent sets of that size and then remove all of them from the graph. Our preprocessing phase repeats this process until there are no more than q vertices left in the residual graph, q being a parameter fixed by the user. The value of q depends clearly on the coloring algorithm applied to the residual graph. In our case, q is set to be equal to 800 for all the tested graphs in this paper.

Our preprocessing phase is then described in Algorithm 5.1. Basically, our preprocessing phase iterates the following 4 steps and stops when the residual graph contains no more than q vertices.

1. Apply the Adaptive Multistart Tabu Search (AMTS) algorithm (see Chapter 2) to identify a maximum independent set I_M in G and set $z_{max} = |I_M|$. Then use I_M to initialize set M which will collect other independent sets of the same size (lines 3-5 in Algorithm 5.1).

2. Apply repeatedly ATS to generate as many independent sets of size z_{max} as possible and put them in set M (lines 7-15).

Algorithm 5.1: Preprocessing phase: Extraction of pairwise disjoint independent sets

```
Require: Graph $G = (V, E)$, the size of the residual graph $q$
Ensure: Residual graph of $G$
 1: Begin
 2: while ($|V| > q$) do
 3:    $I_M$ = AMTS($G$, _, $Iter$) {Use AMTS to find a maximum independent set in $G$}
 4:    $z_{max} = |I_M|$
 5:    $M = \{I_M\}$
 6:    $reapt = 0$
 7:    while ($reapt \leq p_{max}$ AND $|M| \leq M_{max}$) do
 8:       $I$ = AMTS($G, z_{max}, Iter$) {Use AMTS to find an independent set of size $z_{max}$}
 9:       if $I \in M$ then
10:          $reapt = reapt + 1$
11:       else
12:          $M = M \bigcup \{I\}$
13:          $reapt = 0$
14:       end if
15:    end while
16:    Find a maximal number of pairwise disjoint independent sets in $M$:
       $\{I_1, ..., I_t\} = arg\ max\{|S| : S \subseteq M, \forall I_x, I_y \in S, I_x \bigcap I_y = \emptyset\}$ {see Section 5.3.2}
17:    if $|V| - |I_1 \cup ... \cup I_t| > q$ then
18:       Remove $I_1, ..., I_t$ from $G$
19:    else
20:       $p = \lceil \frac{|V|-q}{z_{max}} \rceil$
21:       Remove $I_1^{'}, ..., I_p^{'}$ randomly selected from $\{I_1, ..., I_t\}$ from $G$
22:    end if
23: end while
24: End
```

3. Find as many *pairwise disjoint* independent sets $I_1, ..., I_t$ as possible from M (line 16, see Section 5.3.2).

4. Remove all the vertices of $I_1 \cup ... \cup I_t$ from G if the removal leads to a residual graph with at least q vertices. Otherwise, remove randomly independent sets from $\{I_1, ..., I_t\}$ such that the residual graph contains at most q vertices (lines 17-23).

Notice first that this preprocessing procedure is different from conventional methods since steps (2)-(3) are missing from these conventional methods. In our case, after a maximum independent set is identified, we continue to identify as many independent sets of that size as possible with the purpose of extracting a maximum number of pairwise disjoint independent sets. As we will see in Section 5.5, our preprocessing procedure is able to pack more vertices than with the conventional one-by-one extraction strategy with the same number of color classes. This generates smaller residual graphs that tend to be easier to color.

In Algorithm 5.1, step (1) identifies a first maximal independent set I_M whose size z_{max} is used at step (2) to build the pool M of independent sets of that size. The search for a new independent set I of size z_{max} stops when the number of independent sets contained in M reaches a desired threshold (M_{max}) or when no new independent set of that size is found after p_{max} consecutive tries. Obviously, both a larger value for M_{max} or p_{max} could include more independent sets in M, thus giving more chance of finding more pairwise disjoint independent sets of size z_{max} in M in step (3). On the other hand, a larger value for M_{max} (or p_{max}) also implies longer computing time for the preprocessing phase. According to our experiments we fix p_{max} equal to 100 and M_{max} equal to $|V| \cdot \rho$, where ρ is the density of the graph. For the tested DIMACS graphs, M_{max} varies from 100 to 2000.

Step (3) aims at finding a maximum number of *pairwise disjoint* independent sets from the set of collected independent sets of size z_{max}. As we explain in Section 5.3.2, this problem is in fact equivalent to the maximum set packing problem which itself can be approximated by our Adaptive Tabu Search for the maximum independent set problem.

5.3.2 Finding maximal pairwise disjoint independent sets

As mentioned before, given the set of independent sets $M = \{I_1, ..., I_n\}$ of G which are found by ATS, we want to determine from M as many pairwise disjoint sets as possible. In fact, this is a typical maximum set packing problem (MSPP), which itself is equivalent to the maximum clique (thus independent set) problem [Garey and Johnson, 1979].

Our approach exploits the strict relation between the MSPP and the maximum independent set problem. More precisely, we transform the MSPP into the maximum independent set problem and then determine a maximum independent set in the transformed graph. Given n independent sets $\{I_1, ..., I_n\}$, we define a new maximum independent set instance $G' = (V', E')$ as follows. Define V' by $\{1, ..., n\}$ and define the edge matrix by:

$$e_{ij} = \begin{cases} 0, & \text{if } I_i \cap I_j = \emptyset, i, j \in \{1, ..., n\} \\ 1, & \text{otherwise.} \end{cases}$$

Note that the order n of graph G' is bounded by $M_{max} = |V| \cdot \rho$ (see Section 5.3.1). For the graphs used in this paper, n varies from several to 2000.

Now it is straightforward to see that if $\{i_1, ..., i_r\}$ is an independent set in G', $\{I_{i_1}, ..., I_{i_r}\}$ is a pairwise disjoint set in $\{I_1, ..., I_n\}$. Consequently, to obtain a maximum pairwise disjoint set in $\{I_1, ..., I_n\}$, we can determine a maximum independent set in G'.

To illustrate the idea, consider Fig. (left) which is the original graph and contains 8 maximum independent sets of size 3: $I_1 = \{A, B, F\}$, $I_2 = \{A, D, G\}$, $I_3 = \{A, D, H\}$, $I_4 = \{C, E, G\}$, $I_5 = \{C, E, H\}$, $I_6 = \{C, F, I\}$, $I_7 = \{C, G, I\}$, $I_8 = \{D, G, I\}$. Let $M = \{I_1, I_2, \ldots, I_8\}$, the associated graph $G' = (V', E')$ is shown in Fig. 5.1 (right). It is clear that the maximum independent set $\{1, 5, 8\}$ in Fig. (right) leads to the maximum set of disjoint independent sets $\{I_1, I_5, I_8\}$ in M.

For this purpose, we can approximate the problem by applying directly our AMTS algorithm described in Chapter 2 or use an exact algorithm. To make the decision, we implement the exact algorithm described in [Östergård, 2002]. Experiments with this algorithm and comparisons with AMTS lead to two clear observations. First, if G' is solved optimally by the exact algorithm, then AMTS is able to find a solution (independent set) of the same size (thus an optimal solution). Second, many G' graphs cannot be solved by the exact algorithm. In fact, for most of the 11 tested graphs, there is always at least one G' that cannot be solved by the exact algorithm. For these graphs, AMTS still provides approximate solutions. Based on this experiment, we decide to use our AMTS to handle the underlying MSPP.

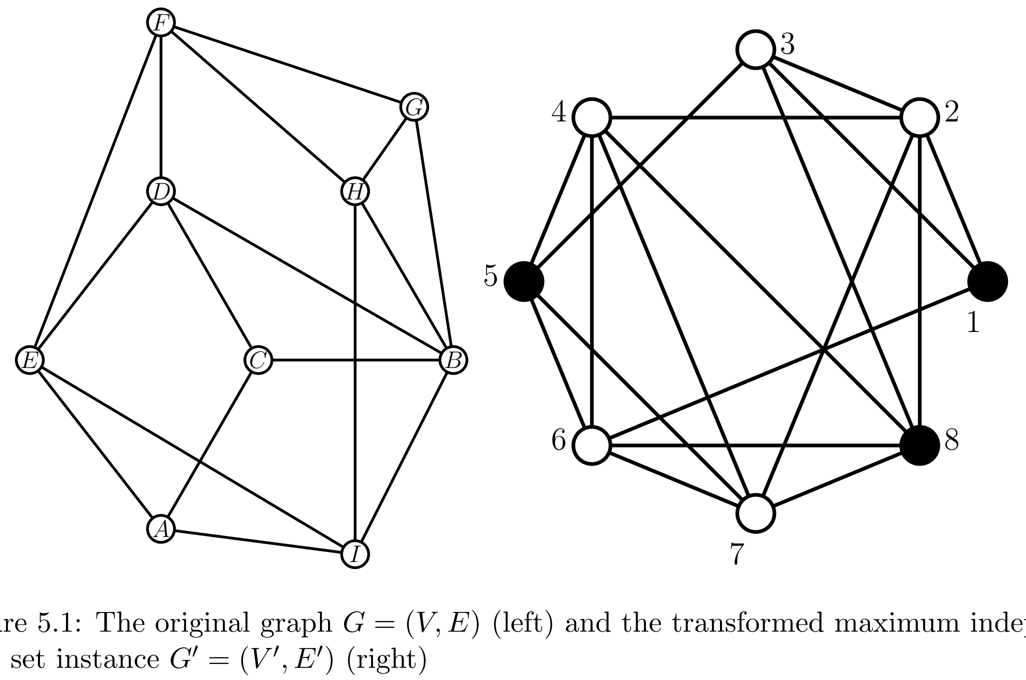


Figure 5.1: The original graph $G = (V, E)$ (left) and the transformed maximum independent set instance $G' = (V', E')$ (right)

5.3.3 Coloring the residual graph

To color the residual graph, one can apply any graph coloring algorithm. For instance, the authors of [Hertz and de Werra, 1987] use a tabu search algorithm (TabuCOL). In [Fleurent and Ferland, 1996], the residual graph is colored by a genetic local search procedure. In [Johnson *et al.*, 1991], independent sets are removed until the residual graph can be colored by a branch-and-bound algorithm. In this work, we employ MACOL which is a recent and effective Memetic Algorithm for graph coloring [Lü and Hao, 2010].

For a given graph G and a fixed number k of colors, MACOL explores a search space Φ composed of all the k-colorings of the graph $G = (V, E)$, i.e., $\Phi = \{C : V \rightarrow \{1, \ldots, k\}\}$. MACOL tries to find a *legal* k-coloring by optimizing (minimizing) a simple function $f(C)$ which counts the number of color conflicts in a k-coloring C. Formally, let $C = \{c_1, c_2, ..., c_k\}$ be a (legal or illegal) k-coloring, the evaluation function $f(C)$ is given by the following formula:

$$f(C) = \mid \{\{u, v\} \in E : \exists c_i \in C,\ u \in c_i, v \in c_i\} \mid \tag{5.1}$$

C is a legal k-coloring if and only if $f(C) = 0$, i.e., each color class c_i of C is an independent set (conflict free)..

MACOL is composed of four basic components: a population of candidate solutions (each solution being a k-coloring in Φ) to sample the search space, a dedicated recombination operator (crossover) to create new candidate solutions (offspring) by blending two or more existing solutions, a tabu search based local optimization operator, and a population management strategy.

MACOL starts with an initial population of illegal k-colorings whose individual k-colorings are first improved by the tabu coloring algorithm which is a variant of the seminal TabuCOL [Hertz and de Werra, 1987]. MACOL improves the solutions of its population throughout a number of *generations*. At each generation, MACOL takes randomly $m \geq 2$ parents and uses the adaptive multi-parent crossover operator (AMPaX) to generate an offspring k-coloring. AMPaX builds one by one the color classes of the offspring solution by taking at each step the largest color class among the parents. During the crossover process, AMPaX takes care of using color classes from different parents in order to generate diversified offspring solutions. Once the new offspring coloring is created, it is immediately improved by the tabu coloring algorithm. The tabu coloring algorithm improves an illegal k-coloring by minimizing the above evaluation function f (Formula 5.1). This is achieved by iteratively changing the color of a vertex that shares the same color with at least one adjacent vertex. To decide whether the improved offspring k-coloring can be added to the population, MACOL implements a distance-and-quality based replacement strategy for the pool updating.

As shown in [Lü and Hao, 2010], the performance of MACOL on the DIMACS graphs is quite competitive compared to other state-of-art coloring algorithms. More details about MACOL can be found in [Lü and Hao, 2010].

5.4 Experimental Results

In this section, we assess the performance of the proposed EXTRACOL algorithm. For this purpose, we present computational results on the 11 largest benchmark graphs from the well-known DIMACS graph coloring Challenge [Johnson and Trick, 1996]. Comparisons are also reported with respect to the underlying MACOL algorithm and 7 other top-performing coloring algorithms from the literature.

5.4.1 Experimental settings

Test instances. As EXTRACOL is designed to color large graphs, we only consider graph instances with at least 1000 vertices. These graphs are known to be difficult and represent a real challenge for coloring algorithms. These graphs belong to four families[1]: three large random graphs (DSJC1000.1, DSJC1000.5, DSJC1000.9), three large flat graphs (flat1000_50_0, flat1000_60_0, flat1000_76_0), two large random geometric graphs (R1000.1c, R1000.5) and three very large random graphs (C2000.5, C2000.9, C4000.5). The details of these 11 instances are described in Section 1.6.

Parameter. Our EXTRACOL algorithm is programmed in C and compiled using GNU GCC on a PC with 2.83 GHz CPU and 8G RAM. To obtain our computational results, each instance is solved 20 times independently with different random seeds (the very large instances are solved 5 times). For each run, the time limit is set to 5 CPU hours except for the three very large graphs C2000.5, C2000.9 and C4000.5, for which a time limit of 120 hours (5 days) is allowed. Notice that these time conditions are comparable with those used in the most recent works like [Galinier *et al.*, 2008; Malaguti *et al.*, 2008; Porumbel *et al.*, 2010b; Lü and Hao, 2010].

To run EXTRACOL, we need to fix q, the number of vertices left in the residual graph. We tested different values and choose the value 800 for all of our experiments. Let us notice that the choice of the value of q mainly depends on the performance of the coloring algorithm applied to the residual graph. In addition to q, MACOL requires also several parameters. In our case, we adopt those used in the original paper [Lü and Hao, 2010].

5.4.2 Computational results

5.4.2.1 Comparison with the current best results

Our first experiment aims to evaluate EXTRACOL with the current best results on our benchmark graphs. Table 5.1 summarizes the computational statistics. Columns 2-3 give the features of the graphs: the number of vertices (n) and the density of each graph ($dense$). Column 4 shows the current best known result k^* reported in the literature. In columns 5-9, the computational statistics of our EXTRACOL algorithm are given, including the number of colors obtained (k_{best}), the success rate ($Succ$), the average computation time in minutes required by the preprocessing procedure (T_{extr}), the average number of

[1]http://www.info.univ-angers.fr/pub/porumbel/graphs/index.html.

Table 5.1: Computational results of EXTRACOL (and MACOL) on the set of 11 largest DIMACS benchmark graphs. The four improved results are indicated in bold.

Instance	n	*dense*	k^*	EXTRACOL[a]					MACOL [Lü and Hao, 2010]		
				k_{best}	*Succ*	$T_{extr}(m)$	*iter*	$T(m)$	k_{macol}	*Succ*	*iter*
DSJC1000.1	1000	0.10	20	20	20/20	11	3.1×10^7	93	20	20/20	3.5×10^7
DSJC1000.5	1000	0.50	83	83	20/20	3	2.0×10^8	132	83	20/20	2.2×10^8
DSJC1000.9	1000	0.90	223	**222**	3/20	8	5.4×10^8	258	223	18/20	4.5×10^8
R1000.1c	1000	0.97	98	101	18/20	10	6.4×10^5	18	98	20/20	7.5×10^5
R1000.5	1000	0.48	234	250	11/20	12	8.8×10^8	183	245	13/20	1.2×10^9
flat1000_50_0	1000	0.49	50	50	20/20	8	2.9×10^5	12	50	20/20	3.2×10^5
flat1000_60_0	1000	0.49	60	60	20/20	7	5.1×10^5	15	60	20/20	6.3×10^5
flat1000_76_0	1000	0.49	82	82	20/20	3	6.7×10^7	79	82	20/20	7.2×10^7
C2000.5	2000	0.50	148	**146**	5/5	145	1.7×10^8	253	148	1/5	8.2×10^8
C2000.9	2000	0.90	413[b]	**409**	2/5	118	4.5×10^8	329	413[b]	2/5	7.5×10^8
C4000.5	4000	0.50	271	**260**	4/5	5186	1.8×10^8	5298	272	3/5	1.2×10^9

a. See http://www.info.univ-angers.fr/pub/hao/extracol.html for the coloring result of each graph.

b. This result, which is not reported in the original paper [Lü and Hao, 2010], is obtained by running MACOL ourself.

iterations required by MACOL for coloring the residual graph (*iter*) and the total time required in minutes (T). The results of MACOL are also reproduced in Table 5.1 (columns 10-12) and we discuss these results in the next section.

From Table 5.1, we observe that, except for the two R1000 graphs, the results obtained by EXTRACOL are very competitive when compared to the current best known results reported in the literature.

For two of the three large random graphs (DSJC1000.1 and DSJC1000.5) and the three *flat* graphs (flat1000_x_0,x=50,60,76), EXTRACOL can easily reach the previous best known result within no more than 2.5 hours and with a success rate of 100%. More importantly, for DSJC1000.9, our EXTRACOL algorithm obtains for the first time a new 222-coloring, improving thus the previous best-known 223-coloring solution that has been obtained very recently in [Malaguti *et al.*, 2008; Xie and Liu, 2009; Porumbel *et al.*, 2010b; Lü and Hao, 2010].

For the three very large random graphs with 2000 and 4000 vertices, EXTRACOL shows even better performance. Indeed, the previous best known coloring requires respectively $k^* = 148$, 413 and 271 for these graphs, which have been found very recently in [Porumbel *et al.*, 2010b; Lü and Hao, 2010]. It is interesting to observe that EXTRACOL is able to color these graphs with $k = 146$, 409 and 260, leading to a gain of 2, 4 and 11 colors respectively. In Section 5.5.1, we show some elements to explain why EXTRACOL performs so well on these graphs.

However, for the two random geometric graphs (R1000.1c and R1000.5), our algorithm performs poorly. In Section 5.5.2, we show an analysis of EXTRACOL on these two graphs to try to understand why this happens.

5.4.2.2 Comparison between EXTRACOL and MACOL

Since EXTRACOL uses MACOL in its coloring phase to color the residual graphs, it is interesting to compare the results of EXTRACOL against those of MACOL. The results

Table 5.2: Detailed information of EXTRACOL and MACOL on C4000.5 and C2000.5 regarding the number of independent sets (color classes) of different sizes.

Size of independent set	C2000.5		C4000.5	
	EXTRACOL	MACOL	EXTRACOL	MACOL
18	0	0	63	1
17	0	0	57	29
16	53	11	42	79
15	18	43	21	67
14	14	36	15	42
13	14	26	22	20
12	16	10	12	16
11	10	8	10	9
10	15	8	6	3
9	4	3	5	4
8	2	1	6	1
7	0	2	1	1
Total	146	148	260	272

of MACOL without the preprocessing phase are listed in Table 5.1 (columns 10–12). Note that for both algorithms, the time limit is set to be 5 CPU hours for graphs of 1000 vertices and 5 days for graphs of 2000 and 4000 vertices.

As one can observe in Table 5.1, for the three very large graphs (C2000.5, C2000.9 and C4000.5), EXTRACOL is able to find much better solutions than MACOL with respectively 2, 4 and 12 fewer colors. In order to get some insight about this difference, we show in Table 5.2 more detailed information about the computational results on C2000.5 and C4000.5. For each graph, columns 2-3 and columns 4-5 show the number of independent sets of size $|S|$ for the results obtained by EXTRACOL and MACOL respectively on these graphs. Now recall that to generate a new offspring solution, MACOL operates by transmitting large color classes (independent sets in legal k-colorings) from parents to offspring. However, from Table 5.2, we observe that for these two graphs, it is very difficult for MACOL to generate very large color classes when it is directly applied to the initial graphs. For instance, if we consider C4000.5, we see that EXTRACOL obtains 63 color classes of size 18 while MACOL obtains only one class of this size. We can make the same remark on C2000.5 for which EXTRACOL and MACOL obtain respectively 53 and 11 color classes of size 16. More generally, for large graphs, it seems wise to use a dedicated algorithm (as EXTRACOL's preprocessing phase) to find large independent sets and remove them from the graph.

For DSJC1000.9, EXTRACOL improves on the result of MACOL with one fewer color. For the two random graphs (DSJC1000.x, x=1,5) and the three flat graphs (flat1000_x_0, x=50, 60, 76), EXTRACOL and MACOL achieve the same results in terms of the number of colors used. However, EXTRACOL requires less search effort and finds these solutions more quickly in terms of the number of iterations. This observation remains in fact valid for most of the tested graphs. This tends to show that EXTRACOL's preprocessing makes the residual graph easier to color.

Finally, for the two random geometric graphs (R1000.1c and R1000.5), the results of EXTRACOL are inferior to those obtained with MACOL. We investigate this phenomenon in Section 5.5.2.

Table 5.3: Comparison of EXTRACOL with 7 best performing coloring algorithms. Most of them are population-based hybrid algorithms.

Instance	n	k^*	k_{best}	7 reference graph coloring algorithms						
				[Porumbel *et al.*, 2010b]	[Xie and Liu, 2009]	[Malaguti *et al.*, 2008]	[Galinier *et al.*, 2008]	[Funabiki and Higashino, 2000]	[Morgen-stern, 1996]	[Fleurent and Ferland, 1996]
DSJC1000.1	1000	20	20	20	-	20	20	21	-	-
DSJC1000.5	1000	83	83	83	84	83	84	88	89	84
DSJC1000.9	1000	223	**222**	223	223	225	224	228	-	-
R1000.1c	1000	98	101	98	-	98	-	98	98	99
R1000.5	1000	234	249	238	-	234	-	237	241	268
flat1000_50_0	1000	50	50	50	50	50	50	50	50	84
flat1000_60_0	1000	60	60	60	60	60	60	60	60	84
flat1000_76_0	1000	82	82	82	83	82	84	87	89	84
C2000.5	2000	148	**146**	148	150	-	-	162	165	153
C2000.9	2000	413	**409**	-	-	-	-	-	-	-
C4000.5	4000	271	**260**	271	-	-	-	301	-	280

5.4.2.3 Comparison with other algorithms

Now we compare EXTRACOL with 7 other state-of-art heuristics published in the literature [Porumbel *et al.*, 2010b; Xie and Liu, 2009; Malaguti *et al.*, 2008; Galinier *et al.*, 2008; Funabiki and Higashino, 2000; Morgenstern, 1996; Fleurent and Ferland, 1996]. One notices that all of them but one [Funabiki and Higashino, 2000] are population-based hybrid algorithms.

For this comparison, we are mainly interested in solution quality in terms of the number of colors needed to color a graph. Table 5.3 presents the comparative results on the set of the 11 graphs. Columns 3 and 4 recall the previous best known k^* and the best results obtained by EXTRACOL (k_{best}). Columns 5-11 present the best results obtained by these reference algorithms.

From Table 5.3, we observe that recent hybrid algorithms like [Porumbel *et al.*, 2010b; Xie and Liu, 2009; Malaguti *et al.*, 2008] show globally very good or excellent performance, in particular on graphs with no more than 1000 vertices. However, none of them can find a 222-coloring for DSJC1000.9 which is achieved by EXTRACOL. This difference seems even more pronounced when the three very large graphs are considered. Indeed, to color C2000.5 and C4000.5, the reference algorithms require at least 148 and 271 colors respectively with the best results obtained recently by Evo_Div [Porumbel *et al.*, 2010b] using a time limit of 5 and 30 days while EXTRACOL requires only 146 and 260 colors. For C2000.9 which is very dense (and hard), few results are available in the literature. On the other hand, we observe that EXTRACOL performs poorly on R1000.1c and R1000.5 with respect to most of the reference algorithms.

5.5 Analysis and insights

5.5.1 Influence of preprocessing

The EXTRACOL's preprocessing phase uses a heuristic method to extract at each iteration as many pairwise disjoint independent sets as possible. Compared to the conventional preprocessing which extracts greedily independent sets one by one, our preprocessing is

Table 5.4: Effect of our preprocessing (EXTRACOL) and conventional preprocessing (OBOCOL) on C2000.5.

	EXTRACOL			OBOCOL		
	$\|S\| \times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used	$\|S\| \times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used
Independent sets obtained by preprocessing	16×53	1202	77	16×36	1205	79
	15×18			15×27		
	14×6			14×16		
Independent sets (color classes) from MACOL	14×8	798	69	14×5	795	69
	13×14			13×20		
	12×16			12×14		
	11×10			11×13		
	10×15			10×6		
	9×4			9×7		
	8×2			8×3		
	7×0			7×1		
Total		2000	146		2000	148

Table 5.5: Effect of the preprocessing (EXTRACOL) and conventional greedy preprocessing (OBOCOL) on C4000.5.

	EXTRACOL			OBOCOL		
	$\|S\| \times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used	$\|S\| \times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used
Independent sets obtained by preprocessing	18×63	3202	191	18×40	3200	195
	17×57			17×66		
	16×42			16×40		
	15×21			15×32		
	14×8			14×17		
Independent sets (color classes) from MACOL	14×7	798	69	14×7	800	69
	13×22			13×20		
	12×12			12×11		
	11×10			11×16		
	10×6			10×5		
	9×5			9×5		
	8×6			8×4		
	7×1			7×1		
Total		4000	260		4000	264

able to cover more vertices than the greedy extraction approach. In other words, in order to cover the same number of vertices (nearly $|V| - q$), our preprocessing needs fewer independent sets and thus fewer colors for these vertices.

To highlight the difference of these two preprocessing methods, we show a detailed comparison by considering two large graphs (C2000.5 and C4000.5). We recall that these two graphs are very difficult if they are directly colored without a preprocessing phase. Indeed, even the most recent hybrid algorithms can only find 148-coloring for C2000.5 and 271-coloring for C4000.5.

To solve these two instances, we apply both the conventional greedy preprocessing method and our preprocessing method to extract independent sets until there are at most 800 vertices left in the residual graph, which is then colored by MACOL. For both algorithms, each instance is solved 5 times with a time limit of 5 days per run. The results are summarized in Table 5.4 (C2000.5) and Table 5.5 (C4000.5). In the table, we show the results obtained respectively with our preprocessing (EXTRACOL) and the conventional greedy preprocessing method (OBOCOL). Columns 2 and 5 show, in the form of $x \times y$, the number y of independent sets of size x extracted by the preprocessing phase (upper part of each table) or established by MACOL (lower part of each table). Columns 3 and 6 give the number of vertices covered by the independent sets. Columns 4 and 7 indicate the number of these independent sets (color classes).

From Table 5.4, we observe that for C2000.5, our preprecessing identifies 77 independent sets (of sizes 16, 15, 14) covering 1202 vertices while the conventional preprecessing removes 1205 vertices, but using 79 colors. We notice also that our preprocessing extracts more independent sets of the largest size than the conventional preprecessing (53 against 36). For both residual graphs, they are colored by MACOL with 69 colors. This leads to a difference of 2 colors in favor of our preprocessing.

Similarly, from Table 5.5, we observe that for C4000.5, the maximal independent set that is identified is of size 18. The conventional preprocessing can only extract 40 independent sets of this size while our preprocessing extracts 63. Moreover, our preprocessing removes 191 large independent sets which cover 3202 vertices, while the conventional preprocessing extracts 4 more independent sets (i.e. 195) which cover only 3200 vertices. To color both residual graphs, MACOL requires additional 69 colors. Once again, this leads to a better solution with our preprocessing method with 4 colors in less with respect to the conventional preprocessing.

Finally, if we compare the coloring results of Tables 5.4 and 5.5 with those reported in Table 5.3, we observe that for these two very large random graphs, even the conventional preprocessing method leads to better results than the direct coloring approach.

5.5.2 Limitation of preprocessing

As previously observed, EXTRACOL performs poorly on the two geometric graphs R1000.1c and R1000.5 although it shows excellent performance on all the other large graphs. It is then interesting to investigate what happens on these graphs. Clearly, it would be very difficult to provide a formal justification. Still, empirical observations would contribute to some extent to the understanding of EXTRACOL's counter-performance on these graphs.

Notice first that the geometric graphs are constructed in a special way [Sewell, 1996]. For a graph R$N.d$, the set of N points (nodes) are randomly scattered in an 1 by 1 square. Two nodes are adjacent if their geometric distance is smaller than d. So contrary to standard random graphs, geometric graphs have special structures in terms of node degrees, independent sets and cliques. These structures may imply a particular relation between the number of color classes of a given size in an optimal coloring (call this number A) and the number of independent sets (potential color classes) of the same size existing in a graph (call this number B). To illustrate this point, we sampled a set of optimal 65-colorings for geometric graph r250.5. Inspecting these optimal solutions shows the number of color classes of size 6 (the largest possible size for this graph) within a 65-coloring varies between 2 and 6 while it is easy to extract 12 pairwise disjoint independent sets of this size, i.e. $A << B$. In other words, more than half of the extracted independent sets of size 6 are not part of an optimal coloring and using these independent sets as color classes

Table 5.6: Detailed results on R1000.5

size of independent set	EXTRACOL			OBOCOL		
	N_1	N_2	N_3	N_1	N_2	N_3
7	12	225	≥ 237	8	229	≥ 237
6	20	209	≥ 241	24	211	≥ 243

Table 5.6 shows the number of independent sets extracted (N_1), the maximum clique size of the residual graph (N_2), and the lower bound on the needed colors after the extraction ($N_3 = N_1 + N_2$).

would increase the number of needed colors with respect to the chromatic number.

To complement this explanation, we show in Table 5.6 some statistics obtained on R1000.5 with the preprocessing phase.

R1000.5 has a known chromatic number of 234. For this graph, it is quite easy for our ATS algorithm to find maximum cliques of size 234. Consequently, for the preprocessing to be helpful, each time an independent set is removed from the graph, the maximum clique size for the residual graph should decrease by 1. However, we observe from Table 5.6 that this is not always the case when the preprocessing is applied to this graph.

For instance, the first iteration of our preprocessing extracts 12 pairwise disjoint independent sets of size 7. However, one observes that at least 3 out of these 12 independent sets cannot be part of an optimal solution and are wrongly extracted. Indeed, the residual graph after extracting these 12 independent sets contains cliques of size 225, implying that we need at least 237 (12 plus 225) colors for the initial graph which is 3 colors above the chromatic number. The situation becomes even worse when 20 pairwise disjoint independent sets of size 6 are additionally extracted because the lower bound of the needed colors becomes now 241. The same observation can be made with the conventional greedy preprocessing which leads to even worse results (see OBOCOL results). We conclude that for this graph, some large independent sets are not part of any optimal 234-coloring and extracting such independent sets cannot help decrease the number of colors needed for the whole graph. The analysis realized on R1000.1c leads to the same conclusion.

This analysis shows the limit of the preprocessing approach which is basically due to its greedy nature. Indeed, if a mistake is made during the preprocessing phase, the mistake cannot be repaired. To remedy this difficulty, one possibility would be to allow the coloring procedure to integrate the extracted independent sets during its search.

5.6 Conclusion

In this chapter, we revisit the graph coloring approach using independent set extraction and apply our proposed adaptive multistart tabu search approach (AMTS) for the MCP to the graph coloring problem. Furthermore, we develop an improved preprocessing procedure which is able to pack more vertices than with the conventional one-by-one extraction strategy with the same number of color classes. This generates smaller residual graphs that tend to be easier to color. Instead of extracting one independent set each time, the proposed preprocessing method tries to extract many pairwise disjoint sets. Such a preprocessing maximizes the number of vertices covered by the extracted independent sets, hopefully making the residual graph easier to color.

The computational results obtained on the 11 largest DIMACS benchmark graphs with 1000, 2000 and 4000 vertices show, except for the two geometric random graphs, remarkable performance. In particular, for four very hard instances (DSJC1000.9, C2000.5, C2000.9, C4000.5), EXTRACOL is able to improves on the previous best known results reported in the literature by finding solutions with 222, 146, 409 and 260 colors respectively, implying a gain of 1, 2, 4 and 11 colors with respect to the current best colorings for these graphs.

However, EXTRACOL performs poorly on some large geometric graphs although its performance is remarkable on all the other tested graphs. A further analysis on these large geometric graphs disclose that the extracting independent sets as a preprocessing technique suffers some inevitable limitations. Actually, if an independent set is wrongly extracted such that it is not part of the optimal coloring, the mistake can never be repaired. To remedy this difficulty, we will improve EXTRACOL by allowing the coloring algorithm to reconsider the independent sets extracted during the preprocessing phase, and present an extraction and expansion approach in the next chapter.

Chapter 6

Advanced applications in graph coloring: An extraction and expansion approach for coloring large graphs

This chapter further extends EXTRACOL proposed in the last chapter by proposing additional strategies to remedy the limitation of EXTRACOL, leading to the improved extraction and expansion algorithm (IE^{2}COL). Basically, IE^{2}COL employs a forward independent set extraction strategy described in the last chapter to reduce the initial graph. From the reduced graph, IE^{2}COL triggers a backward coloring process which uses extracted independent sets as new color classes for intermediate subgraph coloring. The proposed method is assessed on 11 large benchmark graphs with 1000 to 4000 vertices. Computational results show that it provides new upper bounds for 4 graphs and matches consistently the current best known results for 6 other graphs. The chapter is based on two articles, one is published in Discrete Applied Mathematics [Hao and Wu, 2012] and the other is accepted in Asia-Pacific Journal of Operational Research [Wu and Hao, 2011b].

Contents

6.1 Introduction

In the last chapter, we presented an extraction and coloring approach based on independent set extraction (denoted by EXTRACOL) for coloring large graph. The extraction phase of EXTRACOL tries to remove at each step a maximum collection of disjoint independent sets of maximum size instead of a single independent set. As shown in the last chapter, this extraction strategy is able to pack more vertices than with the conventional one-by-one extraction strategy with the same number of color classes. This generates smaller residual graphs that tend to be easier to color. Evaluation of EXTRACOL on the set of the largest graphs (with 1 000 to 4 000 vertices) of the DIMACS challenge benchmarks showed remarkable results.

Preprocessing a graph by extracting independent sets reduces the initial graph and tends to ease the coloring task. Such a preprocessing strategy relies on the hypothesis that each extracted independent set defines a color class of the final coloring. Unfortunately, this may not be the case all the time. Indeed, as demonstrated in the last chapter, EXTRACOL performs poorly on the two geometric graphs R1000.1c and R1000.5 although it shows excellent performance on all the other large graphs. A further analysis on these two geometric graphs shows that due to the particular structure of these graphs, many largest (extracted) independent sets are not part of a final coloring. In this case, it is harmful to remove definitively these independent sets from the graph since this will prevent inevitably the subsequent coloring algorithm from reaching an optimal coloring. To overcome the deficiency of the proposed EXTRACOL algorithm, one solution is to allow the subsequent coloring algorithm to "reconsider" the extracted independent sets and allow some vertices of these extracted sets to change their colors.

Based on the above idea, in this chapter, we propose an extraction and expansion approach to further improve EXTRACOL. Basically, IE^{2}COL employs a forward independent set extraction strategy described in the last chapter to reduce the initial graph. From the reduced graph, IE^{2}COL triggers a backward coloring process which uses extracted independent sets as new color classes for intermediate subgraph coloring.

We report experimental studies of IE^{2}COL on the set of 11 largest and most challenging benchmark graphs (with 1000 to 4000 vertices) from the DIMACS benchmarks. These results show that the proposed algorithm obtains new upper bounds for 4 graphs (flat1000_76_0, C2000.5, C4000.5, C2000.9) and matches consistently the current best-known results for 6 other graphs.

The rest of this chapter is organized as follows. Section 6.2 presents the proposed algorithm. Section 6.3 is dedicated to extensive computational evaluations and comparisons. Section 6.4 investigates some key components of the proposed approach, followed by the concluding section.

6.2 Improved extraction and expansion coloring (IE^{2}COL)

6.2.1 General IE^{2}COL procedure

The proposed IE^{2}COL algorithm is based on and extends the basic extraction and expansion method of [Wu and Hao, 2012a; Wu and Hao, 2011b] and can be summarized by the following general procedure composed of three phases.

1. The *extraction* phase simplifies the initial graph G by removing iteratively large independent sets (as well as the corresponding edges) from the original graph. To be effective, each iteration removes a collection of disjoint independent sets of the same size (the largest possible) according to the method developed in the last chapter. This phase stops when the residual graph contains no more than a fixed number of q vertices.

2. The *initial coloring* phase applies a graph coloring algorithm (the memetic algorithm presented in [Lü and Hao, 2010]) to the residual graph G_z to determine a $(k-t)$-coloring where t is the number of extracted independent sets. If a legal $(k-t)$-coloring $C = \{c_1, ..., c_{k-t}\}$ for G_z is found, then C plus the t independent sets extracted during the phase 1 constitutes a legal k-coloring of the initial graph G, return this k-coloring and stop. Otherwise, continue to phase 3 to trigger the expansion and backward coloring phase. The memetic coloring algorithm applied to G_z and intermediate subgraphs (phase 3) is discussed in Section 6.2.2.

3. The *expansion and backward coloring* phase extends the current subgraph G' by adding back some extracted independent sets S to obtain an extended subgraph G''. Then the coloring algorithm is run on G'' by starting from the current coloring of G' extended with the independent sets of S as new color classes. Once again, if a legal coloring is found for the subgraph G'', this coloring plus the remaining independent sets forms a legal k-coloring of the initial graph G and the whole procedure stops. Otherwise, one repeats this expansion and backward coloring phase until no more independent set is left or a legal coloring is found for the current subgraph under consideration. Possible strategies to select independent sets for expansion are discussed in Sections 6.2.3.

Notice that the EXTRACOL approach described in the last chapter corresponds to phases 1 and 2 and consequently can be considered as a special case of the extraction and expansion approach. The expansion and backward coloring phase (phase 3) is critical since the extracted independent sets are re-examined by the coloring process. If some vertices of an extracted independent set should not receive the same color, they have a chance to be assigned the right color by the applied coloring algorithm.

Figures 6.1 and 6.2 illustrate how our proposed approach works. In this example, we want to color the initial graph G_0 with $k = 4$ colors. In Fig. 6.1, we obtain two smaller graphs G_1, G_2 by successively extracting two independent sets $I_1 = \{2, 5, 7, 9\}$ and $I_2 = \{1, 8, 11\}$ from G_0. The smallest graph G_2 is then colored with 2 colors and we get an initial (and illegal) 2-coloring solution s'_2 of G_2 (Fig. 6.2, G_2). Using s'_2, we construct an initial (and illegal) 3-coloring solution s_1 of G_1. s_1 is then improved by the coloring algorithm and a legal 3-coloring s'_1 of G_1 is obtained (Fig. 6.2, from G_1 to G'_1). At last, we use the legal 3-coloring s'_1 to construct a legal 4-coloring of G_0 (Fig. 6.2, from G'_1 to G_0).

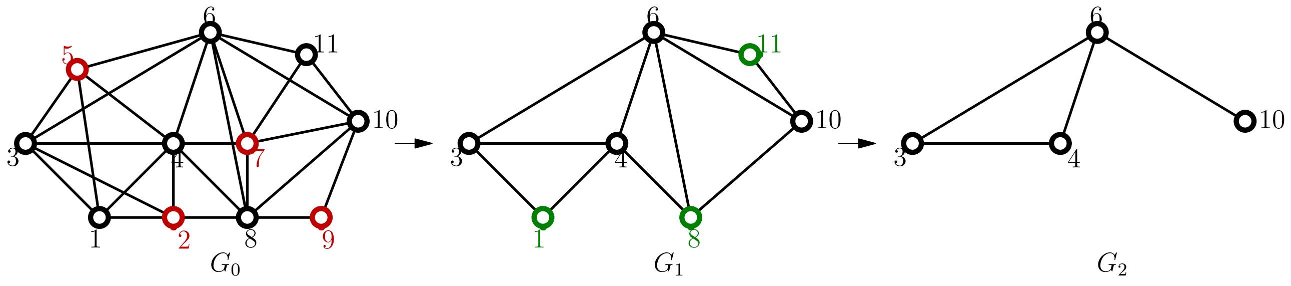


Figure 6.1: The extraction phase

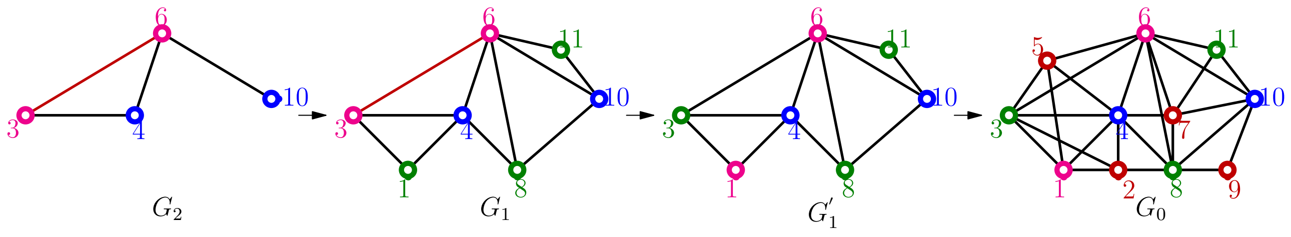


Figure 6.2: The expansion phase

The proposed IE^{2}COL algorithm, designed for the graph k-coloring problem, implements this general approach and is described in Alg. 6.1. In what follows, we show how the main components of IE^{2}COL are implemented.

6.2.2 Initial and intermediate graph coloring

The IE^{2}COL algorithm needs an algorithm to color the residual graph G_z and some intermediate subgraphs (Alg. 1, lines 12 and 23). For this purpose, we adopt MACOL [Lü and Hao, 2010] once again. As shown in the last chapter, the performance of MACOL on the DIMACS graphs is quite competitive compared to other state-of-art coloring algorithms. This is why we employ MACOL once again as our underlying coloring algorithm in our IE^{2}COL algorithm.

6.2.3 Expansion strategies for backward coloring

The expansion and backward coloring phase takes as its input the current subgraph G' of G and the colorings of G' in the population $\mathcal{P}$, extends G' to another subgraph G'' by adding some extracted independent sets S and color G'' with the colorings in $\mathcal{P}$ expanded

Algorithm 6.1: The IE^{2}COL algorithm for large graph k-coloring

```
1: Input: An undirected graph $G = (V, E)$; an integer $k$
2: Output: A legal $k$-coloring of $G$ or report failure
3: {EXTRACTION}
4: {Each extraction iteration removes a maximal collection of disjoint independent sets of maximal size
   in $G$.}
5: while ($G$ has more than $q$ vertices) do
6:     Find in $G$ a maximal collection $\mathcal{I}$ of pairwise disjoint independent sets of the largest size possible
7:     Simplify $G$ by removing from $G$ all the independent sets of $\mathcal{I}$ and the associated edges
8: end while
9: Let $\Omega$ contains all the extracted disjoint independent sets; let $t$ the total number of the extracted
   independent sets ($t = |\Omega|$); let $G_z$ be the residual graph from the extraction phase
10: {INITIAL COLORING}
11: {A population of $(k - t)$-colorings is obtained by the MACOL coloring algorithm applied to the
    residual graph $G_z$, see Sect. 6.2.2}
12: Generate a population $\mathcal{P}$ of $(k - t)$-colorings for graph $G_z$ and run MACOL with the colorings of $\mathcal{P}$
    to color $G_z$
13: if (A legal $(k - t)$-coloring $C \in \mathcal{P}$ for $G_z$ is found by MACOL) then
14:     The coloring $C$, plus the $t$ extracted independent sets, forms a legal $k$-coloring for the initial graph
        $G$. Return this $k$-coloring and stop
15: end if
16: {EXPANSION AND BACKWARD COLORING}
17: {Backward coloring of intermediate subgraphs by reconsidering extracted independent sets of $\Omega$}
18: Let $G' = (V', E')$ be the current subgraph of $G$ under consideration, $\mathcal{P}$ be the set of (illegal)
    colorings of $G'$ produced by MACOL
19: while ($\Omega \neq \emptyset$) do
20:     Select some independent sets $S$ from $\Omega$ ($S \subset \Omega$) and recover the corresponding subgraph $G''$
        induced by the vertices of $V' \cup S$ (see Sect. 6.2.3)
21:     $\Omega \leftarrow \Omega \setminus S$
22:     Extend each coloring $C \in \mathcal{P}$ by including the independent sets of $S$ as new color classes
23:     Run MACOL with the extended colorings of $\mathcal{P}$ to color $G''$ (see Sect. 6.2.2)
24:     if (A legal coloring $C \in \mathcal{P}$ for $G''$ is found by MACOL) then
25:         The coloring $C$, plus the remaining extracted independent sets of $\Omega$, forms a legal $k$-coloring for
            the initial graph. Return this $k$-coloring and stop
26:     end if
27: end while
28: Return (No legal $k$-coloring found)
```

by S (see Alg. 6.1, lines 16-27). The key issues concern the way to select the independents sets S and to rebuild the corresponding subgraph G''. We consider in this section possible strategies to determine the independent sets for expansion.

To determine the set S of independent sets, we can first consider how many independent sets that we pick for expansion. Basically, this decision can be made according to one of two rules: one independent set or several independent sets. This choice may have influences on the subsequent coloring process. Indeed, adding back one independent set at a time implies limited changes between subgraphs G' and G'' and limited extensions to the current colorings (only one new color class is added). This leads thus to a more gradual coloring optimization. On the other hand, using several independents sets to extend the current subgraph and colorings offers more freedom for coloring optimization.

We can also consider which independent set(s) are to be selected. This decision can be achieved following one of three (at least) rules: reverse of extraction order, extraction order and random order. Given the way independent sets are extracted during the extraction phase, applying the reverse of extraction order handles the independent sets from the smallest to the largest while applying extraction order does the opposite.

It is clear that any combination of the above two decisions defines a strategy that can be used to determine the independent set(s) for subgraph and coloring extensions. Based on experimental observations, we have decided for this work to use the following simplified strategy which proved to be effective for the set of graphs tested in the paper. After the initial coloring phase of the residual graph G_z, we backtrack directly to the initial graph G and add back all the extracted independent sets as new color classes of colorings of G. Experiments showed that this strategy performs quite well for the graphs used in the paper for a computational analysis. In the general case, (e.g., if still larger and harder graphs are considered), it would be necessary to recover and color additional intermediate subgraphs during the expansion and backward coloring phase.

6.3 Experimental Results

In this section, we assess the performance of the proposed IE^{2}COL algorithm. For this purpose, we present computational results on the same set of 11 largest benchmark graphs as in Chapter 5. We also report comparisons with respect to 13 top-performing coloring algorithms from the literature.

6.3.1 Experimental settings

Test instances. Since IE^{2}COL is designed to color large graphs, we only consider graph instances with at least 1000 vertices and use the same 11 benchmark instances as in the last chapter.

Parameter. To run IE^{2}COL, we need to fix the threshold q, the number of vertices left in the smallest residual graph G_z. Based on preliminary experiments and as shown in Section 6.4.1, we have fixed q equal to 500 for all our experiments. In addition to q, MACOL (as well as its tabu coloring algorithm) requires also several parameters. In our case, we adopt those used in the original paper [Lü and Hao, 2010].

Stop condition. All experiments for this study were performed on a computer equipped with an Intel Xeon E5440 processor (2.83 GHz, 8GB RAM) running GNU/Linux. Following the DIMACS machine benchmark[1], our machine requires respectively 0.23, 1.42 and 5.42 CPU seconds for the graphs r300.5, r400.5 and r500.5. For all the tested graphs, the same parameter values are used. To report our computational results, 20 independent runs (5 runs for the three largest random graphs C2000.5, C2000.9 and C4000.5) of IE^{2}COL were performed on each graph with different random seeds. The IE^{2}COL algorithm stops if one of the following conditions is verified:

1. A legal $(k-t)$-coloring is found in the initial coloring phase by MACOL which is limited to 300 generations.

2. A legal coloring is found during the expansion and backward coloring phase.

3. The processing time reaches its timeout limit. The timeout limit is set to be 5 CPU hours except for 3 large graphs C2000.5, C2000.9, C4000.5. For C2000.5, C2000.9, C4000.5, a limit of 5 day is allowed. Notice that these timeout limits are comparable with those reported in the latest papers on large graph coloring like [Lü and Hao, 2010; Malaguti *et al.*, 2008; Porumbel *et al.*, 2010b; Xie and Liu, 2009; Wu and Hao, 2012a; Wu and Hao, 2011b] to obtain state-of-the-art results.

6.3.2 Computational Results

Table 6.1[2] summarizes the computational statistics of our IE^{2}COL algorithm on the set of 11 large benchmark instances. Columns 2–4 indicate the features of the tested instances: the number of vertices ($Node$), the number of edges ($Edge$) and the density of the graph ($Density$). Column 5 displays the current best known results k^* reported in the literature, i.e., the smallest k for which a legal k^*-coloring has ever been found by a coloring algorithm. In columns 6–9, the computational statistics of our IE^{2}COL algorithm are presented, including the smallest number of colors (k) for which IE^{2}COL obtains a legal k-coloring, the success rate (hit) and the average computation time in minutes over the runs where a solution with k colors is found. The last column shows the average number of iterations for the successful runs. If IE^{2}COL has a success rate inferior to 100%, we show additional results with larger k until a 100% success rate is reached.

From Table 6.1, we observe that the results obtained by IE^{2}COL (column 6, k) are highly competitive when compared to the current best known results reported in the literature (column 5, k^*). For the three huge random graphs C2000.5, C2000.9 and C4000.5, colorings with respectively $k = 146$, 409 and 260 were obtained by EXTRACOL (Chapter 5) and reported recently in [Wu and Hao, 2012a]. It is noteworthy that IE^{2}COL is able to further improve these bounds and obtain colorings with $k = 145$, 408 and 259 respectively.

For the three flat graphs, IE^{2}COL can reach the current best known results consistently with a success rate of 20/20. More importantly, for flat1000_76_0, IE^{2}COL obtains for the

[1]dmclique, ftp://dimacs.rutgers.edu in directory /pub/dsj/clique

[2]The results of IE^{2}COL are available at http://www.info.univ-angers.fr/pub/hao/ie2col.html

Table 6.1: Computational results of IE^{2}COL on the set of 11 large and difficult benchmark instances.

Instance	*Node*	*Edge*	*Density*	k^*	IE^{2}COL			
					k	*hit*	*time(m)*	*Iterations*
DSJC1000.1	1000	49629	0.1	20	20	20/20	65	3.2×10^7
DSJC1000.5	1000	249826	0.5	83	83	20/20	116	1.2×10^8
DSJC1000.9	1000	449449	0.9	222^a	222	3/20	256	5.1×10^8
					223	20/20	216	4.3×10^8
flat1000_50_0	1000	245000	0.49	50	50	20/20	25	1.2×10^6
flat1000_60_0	1000	245830	0.49	60	60	20/20	25	1.3×10^6
flat1000_76_0	1000	246708	0.49	82	**81**	3/20	281	5.8×10^8
					82	20/20	26	5.3×10^7
R1000.1c	1000	485090	0.97	98	98	20/20	67	3.9×10^7
R1000.5	1000	238267	0.48	234	245	2/20	282	8.5×10^8
					246	8/20	251	6.8×10^8
					247	20/20	186	4.3×10^8
C2000.5	2000	999836	0.5	146^c	**145**	1/5	1198	1.7×10^9
					146	5/5	223	1.4×10^8
C2000.9	2000	1799532	0.9	409^c	**408**	5/5	720	1.1×10^9
C4000.5	4000	4000268	0.5	260^c	**259**	2/5	6987	6.8×10^8
					260	5/5	5223	1.4×10^8

Note a: This bound was reported very recently in [Titiloye and Crispin, 2011; Titiloye and Crispin, 2013; Wu and Hao, 2012a].
Note c: These bounds were reported very recently in [Wu and Hao, 2012a].

first time a new 81-coloring, improving thus the current best-known result which requires 82 colors.

For the 3 random DSJC graphs which are known to be hard to color for many algorithms, IE^{2}COL can attain the current best known results for two of them (DSJC1000.1 DSJC1000.5) with a hit rate of 20/20. In particular, for DSJC1000.9, IE^{2}COL is able to find 222-colorings which were reported very recently for only three algorithms [Titiloye and Crispin, 2011; Titiloye and Crispin, 2013; Wu and Hao, 2012a].

Table 6.2: Comparison of IE^{2}COL with three related algorithms on the set of 11 large DIMACS benchmark instances.

Instance	k^*	IE^{2}COL			MACOL			EXTRACOL			E2COL		
		k	*hit*	*Iter*	k	*hit*	*Iter*	k	*hit*	*Iter*	k	*hit*	*Iter*
DSJC1000.1	20	20	20/20	3.2×10^7	20	20/20	3.5×10^7	20	20/20	3.1×10^7	20	10/10	5.2×10^7
DSJC1000.5	83	20	20/20	1.2×10^8	20	20/20	2.2×10^8	20	20/20	2.0×10^8	20	4/10	7.2×10^8
DSJC1000.9	222	222	3/20	5.1×10^8	223	18/20	4.5×10^8	222	3/20	5.4×10^8	224	6/10	6.7×10^8
flat1000_50_0	50	50	20/20	1.2×10^6	50	20/20	3.2×10^5	50	20/20	3.2×10^5	50	10/10	1.2×10^6
flat1000_60_0	60	60	20/20	1.3×10^6	60	20/20	6.3×10^5	60	20/20	5.1×10^5	60	10/10	1.7×10^6
flat1000_76_0	82	**81**	3/20	5.8×10^8	82	20/20	7.2×10^7	82	20/20	6.7×10^7	82	10/10	3.5×10^8
R1000.1c	98	98	20/20	3.9×10^7	98	20/20	7.5×10^5	101	18/20	6.4×10^5	98	10/10	5.2×10^8
R1000.5	234	245	3/20	8.5×10^8	245	13/20	1.2×10^9	250	11/20	8.8×10^8	256	1/10	4.7×10^8
C2000.5	146	**145**	1/5	1.7×10^9	148	1/5	8.8×10^8	146	5/5	1.7×10^8	147	5/5	1.1×10^9
C2000.9	409	**408**	5/5	1.1×10^9	413	2/5	7.5×10^8	409	2/5	4.5×10^8	413	2/5	1.3×10^9
C4000.5	260	**259**	2/5	6.8×10^8	272	3/5	1.2×10^9	260	4/5	1.8×10^8	262	5/5	1.8×10^9

6.3.3 Comparing IE^{2}COL with MACOL, EXTRACOL and E2COL

In this section, we compare IE^{2}COL with three related approaches using the set of 11 DIMACS graphs: its underlying memetic coloring algorithm (MACOL [Lü and Hao, 2010]), the approach using independent set extraction as a preprocessing method in the last chapter (EXTRACOL) and the initial basic extraction and expansion algorithm (E2COL [Wu and Hao, 2011b]). The purpose of this comparison is to know to which extend IE^{2}COL can improve on the results of these related approaches and show the added value of the enhancements implemented in IE^{2}COL. Table 6.2 summarizes the computational results of these 4 algorithms.

When comparing IE^{2}COL against MACOL, we notice that they reach the same minimal k value for 6 graphs (DSJC1000.1, DSJC1000.5, flat1000_50_0, flat1000_60_0, R1000.1c and R1000.5). For the other 5 graphs, IE^{2}COL finds better solutions than MACOL. This shows the added value of embedding the memetic coloring algorithm into the proposed extraction and backward coloring approach.

When comparing IE^{2}COL and EXTRACOL, one observes that even though EXTRACOL performs very well on these graphs (except on the two R1000.x graphs), IE^{2}COL delivers better results in 6 out of 11 cases. In particular, thanks to the backward coloring strategy, IE^{2}COL is able to further improve on the current best known results of 3 very difficult graphs (C2000.5, C2000.9, C4000.5) which have been established by EXTRACOL. This highlights the critical role of the expansion-coloring strategy employed by IE^{2}COL.

Finally, when it comes to comparing IE^{2}COL and E2COL, the results are once again in favor of IE^{2}COL because IE^{2}COL improves on the results of E2COL in 6 out of 11 cases. This is possible thanks to the enhancements presented in Section 6.2, concerning particularly the improved strategies for the backward coloring phase. This also underscores the importance of the underlying coloring algorithm (notice that E2COL employs a perturbation-based tabu search coloring algorithm).

6.3.4 Comparison with other state of the art algorithms

In this section, we compare the results of our IE^{2}COL algorithm with 13 state-of-art coloring algorithms, which are based on diverse approaches: reactive tabu search with partial solutions (PCol) [Blöchliger and Zufferey, 2008], iterated local search (ILS) [Chiarandini and Stützle, 2002], variable space search (VSS) [Hertz *et al.*, 2008], quantum annealing (QA) [Titiloye and Crispin, 2011], hybrid evolutionary algorithms (HEA [Galinier and Hao, 1999], MMT [Malaguti *et al.*, 2008], Evo [Porumbel *et al.*, 2010b]), multiagent fusion search (MFS) [Xie and Liu, 2009], mimimal-state processing search (MSP) [Funabiki and Higashino, 2000], distributed coloration neighborhood search (DCNS) [Morgenstern, 1996], adaptive memory search (AmaCol) [Galinier *et al.*, 2008] and ant local search (ALS) [Plumettaz *et al.*, 2010]. For this experiment, we focus on the quality criterion, i.e., the lowest value of k for which a k-coloring can be found.

Table 6.3 presents the comparative results on the set of the DIMACS graphs (except C2000.9 for which no results are reported for the reference algorithms). Columns 2 and 3 recall the best known results (k^*) and the best results found by IE^{2}COL. Columns 4–13 give

Table 6.3: Comparisons between IE^{2}COL and 13 state-of-the-art coloring algorithms in the literature.

Graph	k^*	IE^{2}COL	state-of-the-art coloring algorithms												
			PCol	ILS	VSS	QA	Evo	MMT	MFS	MSP	HGA	DCNS	AmaCol	HEA	ALS
DSJC1000.1	20	20	20	-	20	20	20	20	-	21	-	-	20	20	20
DSJC1000.5	83	83	89	89	86	83	83	83	84	88	84	89	84	83	84
DSJC1000.9	222	222	226	-	224	222	223	225	223	228	-	226	224	224	224
flat1000_50_0	50	50	50	-	50	-	50	50	50	50	84	50	50	-	50
flat1000_60_0	60	60	60	-	60	-	60	60	60	60	84	60	60	-	60
flat1000_76_0	82	**81**	87	-	85	82	82	82	83	87	84	89	84	83	83
R1000.1c	98	98	98	-	-	98	98	98	-	98	99	98	-	-	-
R1000.5	234	245	248	-	-	238	238	234	-	237	268	241	-	-	-
C2000.5	146	**145**	-	-	-	-	148	-	150	162	153	151	-	-	-
C4000.5	260	**259**	-	-	-	-	271	-	-	301	280	-	-	-	-

'-' means unavailability of a result. For 9 of the 10 large DIMACS benchmark graphs, IE^{2}COL obtains the same or improved results with respect to the reference algorithms.

the best results reported by these reference algorithms. From Table 6.3, one observes that IE^{2}COL competes very favorably with these top-performing coloring algorithms. Indeed, if one compares IE^{2}COL with each of the reference algorithm, one finds that over these 10 hard graphs, IE^{2}COL can obtain one or more better solutions (smaller k) and at most one worse result (larger k).

Notice that a completely fair comparison is impossible since the reference algorithms are implemented by different authors and run under different conditions. This comparison is thus presented only for indicative purposes and should be interpreted with caution. Nevertheless, this experiment does show very positive indications about the competitiveness of IE^{2}COL when compared to these state-of-the-art algorithms.

6.4 Analysis of IE^{2}COL

6.4.1 Effect of the size of residual graph

We now turn our attention to a study on the influence of the size of residual graph on the performance of the IE^{2}COL algorithm. Recall that the extraction phase of IE^{2}COL stops when no more than q vertices are left in the residual graph from which the initial coloring and possibly backward coloring phases are launched. Different values of q may impact the outcome of IE^{2}COL. We carry out additional experiments on 4 instances (DSJC1000.5, DSJC1000.9, R1000.1c, flat1000_76_0) and run IE^{2}COL 10 times on each of these instances with $q \in \{300, 500, 600\}$ and show in Table 6.4 the computational results. In addition to k and *hit*, we also indicate the average number of iterations needed to find a k-coloring. For DSJC1000.9, we aim at finding a 223-coloring, for flat1000_76_0, we aim at finding a 82-coloring. For each run of the IE^{2}COL, the timeout limit is set to be 5 CPU hours.

From Table 6.4, we observe that all these q values allow the algorithm to find a legal k-coloring. Nevertheless, IE^{2}COL with $q = 500$ and $q = 600$ reaches more stable results (higher hits), but may require more iterations than with $q = 300$. Therefore, it seems that a relatively larger q makes the algorithm more robust but also slower. This implies

Table 6.4: Influence of the size of residual graph (parameter q) on the performance of IE^{2}COL.

Graph	k^*	$q = 300$			$q = 500$			$q = 600$		
		k	hit	$Iterations$	k	hit	$Iterations$	k	hit	$Iterations$
DSJC1000.5	83	83	6/10	8.4×10^7	83	10/10	1.2×10^8	83	10/10	1.5×10^8
DSJC1000.9	222	223	10/10	4.1×10^8	223	10/10	4.3×10^8	223	9/10	4.6×10^8
flat1000_76_0	81	82	9/10	5.0×10^7	82	10/10	5.3×10^7	82	10/10	5.7×10^7
R1000.1c	98	98	10/10	4.1×10^7	98	10/10	3.9×10^7	98	10/10	4.2×10^7

that there may not be an absolute best value for this parameter and that a compromise between robustness and speed could be possible.

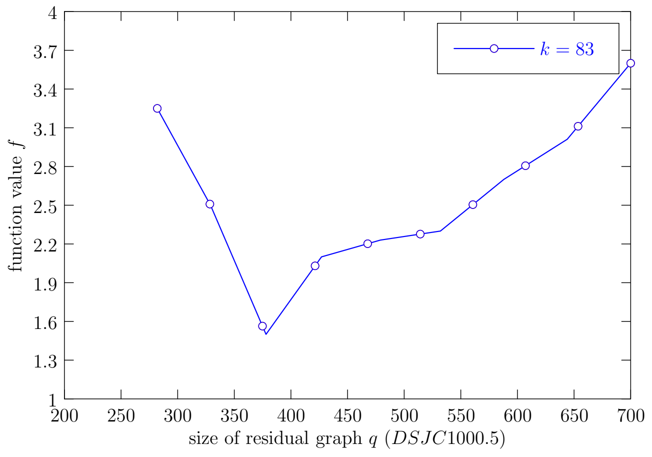


Figure 6.3: Influence of the size of residual graph on the evaluation function f

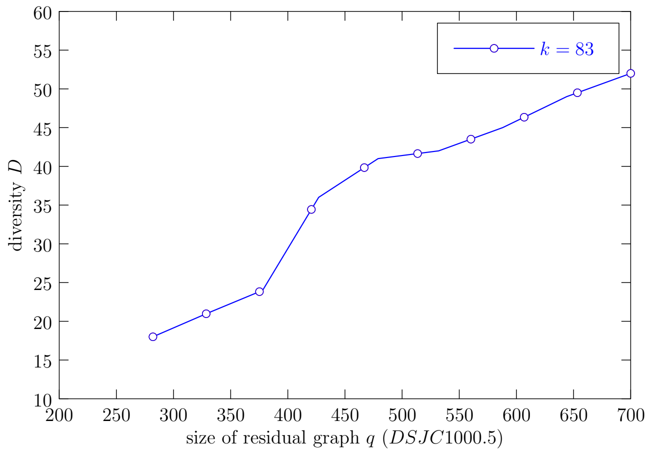


Figure 6.4: Influence of the size of residual graph on the diversity D of the population

To complement this experiment and get more insight, we analyze the influence of q on two other interesting points: (1) the evaluation function f (Eq. 1, Chapter 5) and (2) the diversity of the population. For this purpose, we present below in detail the results on a single graph, but the observations remain valid for several other tested graphs.

The considered instance is DSJC1000.5 with $k = 83$. We show in Fig. 6.3 the influence of q on the evaluation function f using a running profile. The profile is defined by the function $q \longmapsto f_*(q)$ where q is the size of residual graph and $f_*(q)$ the best (smallest) f value at the end of the initial coloring phase (averaged over 10 independent runs). From Fig. 6.3, one can observe that a too large or too small q value can lead to worse (large) results for f. q values ranging from 350 to 500 seem to give the best results.

For memetic algorithms, it is well known that population diversity has an important influence on the performance [Hao, 2011]. A fast lost of the diversity in the population leads to a premature convergence. We show in Fig. 6.4 influence of q on the diversity D of the population. The population diversity is calculated according to the method described in [Porumbel *et al.*, 2010b; Porumbel *et al.*, 2010a]. The plotted profile in Fig. 6.4 is defined by the function $q \longmapsto D_*(q)$ where q is the size of the residual graph and $D_*(q)$ the population diversity at the end of the initial coloring phase (averaged over 10 independent runs). From Fig. 6.4, one observes that a larger value for q can better preserve the population diversity while a smaller value for q can lead to a fast lost of the diversity in the population, thus leading to premature convergence of the memetic algorithm.

Considering jointly Fig. 6.3 and 6.4, we conclude $q = 500$ is an appropriate value, which explains why this value was used for all the experiments reported in this paper. More generally, it is reasonable to believe that q may depend on the effectiveness of the underlying coloring algorithm and on the structure of the graphs to be colored.

6.5 Conclusion

In this chapter, we extended the EXTRACOL algorithm proposed in the previous chapter and presented an extraction and expansion approach for the graph coloring problem able to handle large graphs. This method combines an independent set extraction phase with an expansion and backward coloring phase. The expansion-coloring phase provides a way of reconsidering extracted independent sets as additional color classes for the purpose of allowing some vertices of these extracted independent sets to change their colors.

The proposed IE^{2}COL algorithm implementing this method has achieved noteworthy performance on the set of 11 largest benchmark graphs with 1000 to 4000 vertices from DIMACS benchmarks. IE^{2}COL improves on the current best colorings (new upper bounds) for 4 graphs and matches the current best results for 6 other graphs, while its results are worse in only one case. The improved upper bounds, combined with the new development of lower bounds, constitute a step forward toward the goal of finding the chromatic number of these graphs.

Even though it is believed that it becomes more and more difficult to obtain better upper bounds for the tested benchmark graphs, this study shows that improvements are still possible with new solution strategies, in particular combined method.

Chapter 7

Applications in sum coloring: An Effective Heuristic Algorithm for Sum Coloring of Graphs

Given an undirected graph $G = (V, E)$, the minimum sum coloring problem (MSCP) is to find a legal vertex coloring of G, using colors represented by natural numbers $(1, 2, \ldots)$ such that the total sum of the colors assigned to the vertices is minimized. In this chapter, we apply the proposed adaptive multistart tabu search approach (AMTS) for the MCP to the minimum sum coloring problem, and present EXSCOL, a heuristic algorithm based on independent set extraction for this NP-hard problem. EXSCOL identifies iteratively collections of disjoint independent sets of same size and assign to each independent set the smallest available color. Experimental evaluations on a collection of 52 DIMACS and COLOR2 benchmark graphs show that the proposed approach achieves highly competitive results. For more than one fourth of the graphs used in the literature, our approach improves the current best known upper bounds. While for nearly one third of the tested instances, our approach is able to to improve the current best lower bounds. Parts of this chapter are detailed in one published paper [Wu and Hao, 2012b] and in one submitted paper [Wu and Hao, 2012d].

Contents

7.1 Introduction

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, the minimum sum coloring problem (MSCP) is to find a vertex coloring $c = \{I_1, \ldots, I_k\}$ of G such that the following total sum of the colors is minimized:

$$Sum(c) = \sum_{i=1}^{k} \sum_{v \in I_i} i \tag{7.1}$$

The optimal (smallest) value of this sum is called the *chromatic sum* of G and denoted by $\sum(G)$. The number k of the k-coloring leading to the *chromatic sum* is called the strength of the graph and denoted by $s(G)$. It is clear that $s(G)$ is lower bounded by $\chi(G)$, i.e. $s(G) \geq \chi(G)$.

The minimum sum coloring problem is known to be NP-hard in the general case [Kubicka and Schwenk, 1989]. In addition to its theoretical significance as a difficult combinatorial problem, the MSCP is notable for its ability to formulate a number of important problems, including those from VLSI design, scheduling and resource allocation [Bar-Noy *et al.*, 1998; Malafiejski, 2004].

During the past two decades, the MSCP has been studied essentially from a theoretical point of view and special cases (e.g. tree, interval graphs, line graphs etc) have been identified which admit efficient approximation algorithms or polynomial algorithms [Bar-Noy *et al.*, 1998; Malafiejski, 2004]. For the purpose of practical solving of the general MSCP, several heuristic algorithms have recently been proposed to find suboptimal solutions.

For instance, Kokosiński and Kawarciany proposed a parallel genetic algorithm [Kokosiński and Kawarciany, 2007]. Li et al. presented MRLF [Li *et al.*, 2009], an effective greedy algorithm based on the well-known RLF graph coloring heuristic [Leighton, 1979]. Moukrim et al. showed a technique for computing the lower bound for the MSCP based on extraction of specific partial graphs [Moukrim *et al.*, 2010]. Bouziri and Jouini adapted a tabu coloring algorithm to sum coloring [Bouziri and Jouini, 2010]. Douiri and Elbernoussi illustrated a hybrid algorithm which combines a genetic algorithm with a local search heuristic [Douiri and Elbernoussi, 2011]. Finally, in [Bar-Noy *et al.*, 1998] Bar-Noy et al. presented a theoretical study of a heuristic algorithm based on finding iteratively maximum independent sets (MaxIS) and showed that the MaxIS is a 4-approximation to the MSCP, which is a tight bound to within a factor of 2. Nevertheless, the practical performance of this heuristic was not verified with computational experiments.

In this chapter, we apply the same idea of EXTRACOL proposed in Chapter 5 for the graph coloring to the MSCP, and present an advanced heuristic algorithm called EXSCOL for the MSCP based on independent set extraction. Basically, EXSCOL iteratively extracts from the graph as many large disjoint independent sets of equal size as possible. For each extracted independent set, we assign to it the smallest available color (colors are represented by natural numbers 1, 2...). This process is repeated until the graph becomes empty. The rationale behind this approach is that by extracting many large disjoint independent sets, we naturally favor the construction of large color classes and reduce the number of needed color classes, leading to a reduced total sum of colors.

Experimental results are presented on a large set of 56 benchmark graphs in the literature, showing that the proposed algorithm achieves very competitive results. Indeed, for more than one fourth of the instances used in the literature, the proposed approach improves the current best known results. Furthermore, we assess the performance of two other solution methods using respectively the conventional independent set extraction strategy and graph vertex coloring algorithms.

7.2 EXSCOL: an algorithm for the MSCP

7.2.1 Rationale and general procedure

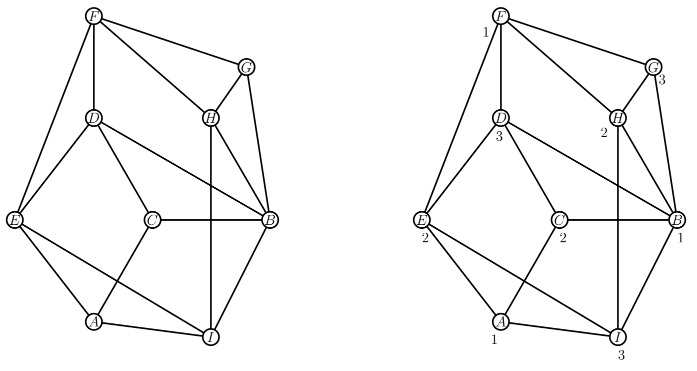


Figure 7.1: An illustration of the proposed EXSCOL algorithm

Let $c = \{I_1, \ldots, I_k\}$ be a legal coloring of graph $G = (V, E)$, each independent set I_i is a color class of c such that all the vertices $v \in I_i$ receive color i. Given the coloring c, its sum of colors $Sum(c)$ according to Eq. (7.1) counts the total sum of the colors induced by c. Suppose $|I_1| \geq |I_2| \geq \ldots \geq |I_1|$, Eq. (7.1) can be rewritten as follows.

$$Sum(c) = 1 \cdot |I_1| + 2 \cdot |I_2| + \ldots + k \cdot |I_k| = \sum_{i=1}^{k} i \cdot |I_i| \qquad (7.2)$$

It is clear that the sum depends on both the number k of the used colors and the size of the color classes. To minimize this sum, one can try to construct large color classes and assign to them small colors. For this purpose, one can remove iteratively the maximum number of disjoint independent sets of the maximum size from the graph until the graph becomes empty. Note that the same idea has been applied to the graph coloring problem by EXTRACOL in Chapter 5. For both tasks of computing a maximum independent set of a graph and finding a maximum set of disjoint sets (which is the maximum set packing problem, see Section 5.3.2), we could use the AMTS algorithm for the MCP.

The proposed EXSCOL algorithms follows this basic idea and can be summarized by the following procedure.

1. Identify an independent set of the largest size possible from the graph;

2. Identify as many pairwise disjoint independent sets of that size as possible and extract them from the graph;

3. Assign to each extracted independent set the smallest available color (the first color used is 1);

4. Stop if the graph becomes empty, go to Step 1 otherwise.

Fig. 7.1 illustrates how this approach works on a graph with 9 vertices. At the first step, we find a maximum independent set of size 3 (e.g. $\{A, D, H\}$). Then we try to identify as many disjoint independent sets of size 3 as possible from the graph, leading to 3 disjoint independent sets $\{A, B, F\}, \{C, E, H\}, \{D, G, I\}$. We assign to these independent sets the colors $1, 2, 3$. Since the graph becomes empty after removing these independent sets, the procedure stops. We obtain a coloring $c = \{\{A, B, F\}, \{C, E, H\}, \{D, G, I\}\}$ with $Sum(c) = 18$ for the graph.

7.2.2 The EXSCOL algorithm

The proposed EXSCOL algorithm (Alg. 7.1) implements the general procedure given in section 7.2.1. EXSCOL starts by identifying a first largest possible independent set I_M (line 4) whose size $|I_M|$ is used later to build a pool M of independent sets of that size (lines 5-15). The search for a new independent set of size $|I_M|$ stops when the number of independent sets contained in M reaches a desired threshold (M_{max}) or when no new independent set of that size is found after p_{max} consecutive tries. From M, EXSCOL tries to determine a maximum number of disjoint independent sets (line 16). As described in Chapter 5, this task corresponds in fact to the maximum set packing problem, which is equivalent to the maximum clique problem and thus can be solved by our AMTS algorithm. Each identified independent set is then removed from the graph (line 17). This process is repeated until the graph becomes empty.

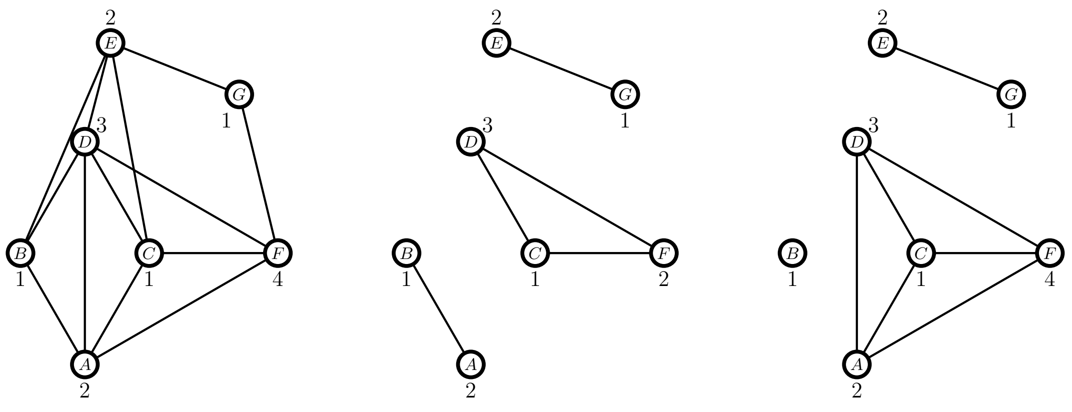
7.3 Lower bounds for the MSCP based on clique decomposition

Given $G = (V, E)$, a partial graph of $G = (V, E)$ is a graph $G' = (V, E')$ such that E' is a subset of E. It is easy to observe that the chromatic sum of G' is a lower bound for the chromatic sum of G. Indeed, any legal coloring of G is a legal coloring of G' while the reverse does not hold. Thus, to calculate a lower bound for the chromatic sum of the original graph G, one could try to find a partial graph of the original graph whose chromatic sum can be efficiently computed and maximized.

This can be achieved by decomposing the vertex set of G into k pairwise disjoint cliques C_1, C_2 ,..., C_p such that $\forall i \neq j,\ C_i \bigcap C_j = \emptyset$ and $\bigcup_i C_i = V$ (see Fig.7.2 for an illustrative example). For a given clique decomposition $\digamma = \{C_1, C_2,..., C_p\}$, its chromatic sum value can be easily computed by assigning to the vertices of each clique C_i of $\digamma$ with the first

Algorithm 7.1: Pseudo-code of the EXSCOL algorithm

```
Require: Graph $G = (V, E)$
Ensure: A coloring of $G$
 1: Begin
 2: $k \leftarrow 1$
 3: while ($|V| > 0$) do
 4:    $I_M \leftarrow AMTS(G)$ {Apply AMTS to find an independent set as large as possible}
 5:    $M \leftarrow \{I_M\}$
 6:    $r \leftarrow 0$
 7:    while ($r \le p_{max}$   $|M| \le M_{max}$) do
 8:       $I \leftarrow AMTS(G, |I_M|)$ {Apply AMTS to find an independent set of size $|I|$}
 9:       if $I \in M$ then
10:          $r \leftarrow r + 1$
11:       else
12:          $M \leftarrow M \bigcup \{I\}$
13:          $r \leftarrow 0$
14:       end if
15:    end while
16:    Find in $M$ as many pairwise disjoint independent sets as possible (see Section 5.3.2):
       $(I_1, ..., I_l) \leftarrow arg\ max\{|A| : A \subseteq M, \forall I^a, I^b \in A, I^a \bigcap I^b = \phi\}$
17:    Remove $(I_1, ..., I_l)$ from $G$
18:    for $i = 1$ to $l$ do
19:       Assign color $k$ to $I_i$
20:       $k = k + 1$
21:    end for
22: end while
23: End
```



$(a) \sum(G) = 14$ $(b) \sum(G'_1) = 12$ $(c) \sum(G'_2) = 14$

Figure 7.2: Partial graphs of G via clique decomposition

$|C_i|$ colors of $[1..|C_i|]$. Thus, for a given clique decomposition $\mathcal{F} = \{C_1, C_2,..., C_p\}$, its chromatic sum is $\sum_{i=1}^{p} \frac{|C_i|(|C_i|+1)}{2}$.

A clique decomposition $\mathcal{F} = \{C_1, C_2,..., C_p\}$ of G corresponds to a partial graph of G and the chromatic sum of $\mathcal{F}$ is therefore a lower bound to the chromatic sum $\sum(G)$ of G. The quality of this lower bound depends on the way to decompose the graph into cliques. For instance, consider the graph $G = (V, E)$ of Fig. 7.2(a), we decompose G in two different ways (Fig. 7.2(b) and Fig. 7.2(c)), we obtain the following chromatic sum $\sum(G'_1) = 12$ and $\sum(G'_2) = 14$. Thus, in order to obtain a lower bound as large as possible, one could try to find a clique decomposition of G whose chromatic sum is as large as possible. This can be considered as an optimization problem where we search for a decomposition of G into cliques, such that the associated chromatic sum is maximized over all the possible clique decompositions:

$$LB^* = max\{ \sum(\mathcal{F}) \mid \mathcal{F} \ is \ a \ clique \ decomposition \ of \ G \} \tag{7.3}$$

It should be noted that LB^* might be strictly smaller than $\sum(G)$ because, a clique decomposition, by ignoring some edges of the original graph G, is a less constrained problem for coloring.

This clique decomposition method for computing lower bounds for the MSCP was originally proposed in [Moukrim *et al.*, 2010] and were further exploited in [Douiri and Elbernoussi, 2012; Helmar and Chiarandini, 2011]. In these previous studies, clique decompositions are obtained by transforming the original graph G into its complement $\overline{G}$ and applying then a vertex coloring algorithm to $\overline{G}$. The color classes of $\overline{G}$ define a clique decomposition of G.

Given a clique decomposition $\mathcal{F} = \{C_1, C_2,..., C_k\}$ of $G = (V, E)$, there is one unique coloring for $\mathcal{F}$, i.e., for each clique C_i $(1 \leq i \leq k)$ of $\mathcal{F}$, we need exactly $|C_i|$ colors to color its vertices. Moreover, the larger the clique C_i is, the more the number of vertices in C_i that need to be colored with large colors. Thus, a clique decomposition with more large cliques tends to have a larger chromatic sum and therefore gives a better lower bound to G. For instance, consider again the example of Fig. 7.2. Moving vertex A from the clique of size 2 (Fig.7.2(b)) to a clique of size 4 (Fig.7.2(c)) increases the sum of colors by 2 because the color of vertex A goes from 2 to 4. For the purpose of obtaining a clique decomposition with more large cliques, one could try to identify as many large cliques as possible from the graph. To achieve this, we can first transform the original graph G into its complement $\overline{G}$ and applying then the EXSCOL algorithms proposed in the last section to $\overline{G}$ to determine a coloring of $\overline{G}$, which corresponds to a clique decomposition of the original G. In this way, we could obtain a lower bound of the original graph G.

7.4 Experimental results

To assess the practical efficiency of our proposed EXSCOL algorithm for computing the upper and lower bounds for the MSCP, we carry out experiments on a total of 56 graphs in the literature and compare EXSCOL with 4 state-of-the-art existing algorithms. We also assess the interest of the conventional strategy of extracting independent set extractions one by one and the pertinence of using graph coloring algorithms to solve the sum coloring problem.

7.4.1 Problem instances and experimental protocol

Two sets of benchmark graphs from the literature are considered in the experiments. The first set (Table 7.1) is composed of 29 well-known DIMACS graphs[1]. These graphs are very popular for testing graph coloring algorithms [Galinier and Hertz, 2006; Johnson and Trick, 1996; Lü and Hao, 2010; Malaguti *et al.*, 2008; Porumbel *et al.*, 2010b]. However only the 12 DSJC random graphs have been recently used for sum coloring [Bouziri and Jouini, 2010; Li *et al.*, 2009].

The second set of benchmarks (see Table 7.2) is composed of 27 graphs from the COLOR02 website[2]. Like the first set, these graphs are initially collected for the purpose of the COLOR02 competition. Among these graphs, sum coloring results have been reported in the literature for 16 graphs [Douiri and Elbernoussi, 2011; Kokosiński and Kawarciany, 2007; Moukrim *et al.*, 2010].

Our EXSCOL algorithm is programmed in C and compiled using GNU GCC on a PC with 2.83 GHz CPU and 8G RAM. To report computational statistics, we run our EXSCOL algorithm on each graph 20 times (5 times for the two huge Cxxxx.5 instances) and reports the following information: the minimum sum of colors, the number of used colors, the average sum of colors over the multiple runs, and the average CPU time.

A desirable feature of EXSCOL is that it does not need any external stop condition to terminate. In fact, it stops when the graph under consideration becomes empty. Nevertheless, EXSCOL has two parameters to be fixed (see Alg. 7.1): p_{max} and M_{max}. As explained in Section 7.2.2, large values for p_{max} and M_{max} could lead to more independent sets collected in M and thus increase the chance of finding more disjoint independent sets. On the other hand, large values for p_{max} or M_{max} also imply long computing times. Based on preliminary experiments we have fixed $p_{max} = 100$ and $M_{max} = 2000$ for all our experiments, although fine-tuning these parameters would lead to better results.

7.4.2 Experimental results for computing upper bounds

Table 7.1 and Table 7.2 summarize respectively the computational statistics of our EXSCOL algorithm obtained on the two sets of benchmark instances[3]. Columns 2-4 give the features of the tested instances: the number of vertices ($|V|$), the number of edges ($|E|$) and the density of the graph (Den). Column 5 indicates the current best known sum values (Sum^*) reported in the literature. Column 6 presents the smallest number of colors (k^*) for which a solution (k-coloring) has ever been found by a graph coloring algorithm. In columns 7-9, the computational statistics of our EXSCOL algorithm are given, including the smallest sum of colors (Sum) with the number of required colors in brackets (k), the average sum of colors ($Avg.$) with the standard deviation in brackets, the average CPU

[1] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/
[2] http://mat.gsia.cmu.edu/COLOR02/
[3] The results of EXSCOL are available at http://www.info.univ-angers.fr/pub/hao/exscol.html

Table 7.1: Computational results of the upper bounds on 27 DIMACS challenge benchmarks.

Instance	$\|V\|$	$\|E\|$	*Den*	Sum^*	k^*	EXSCOL			
						$Sum(k)$	$Avg.(Std.)$	$T[min.]$	Diff
DSJC125.1	125	736	0.09	326	5	326(7)	326.7(0.8)	1	0
DSJC125.5	125	3891	0.50	1015	17	1017(20)	1019.7(8.7)	1	2
DSJC125.9	125	6961	0.89	2511	44	2512(44)	2512.0(0.0)	1	1
DSJC250.1	250	3218	0.10	977	8	985(10)	985.0(0.0)	4	8
DSJC250.5	250	15668	0.50	3281	28	3246(31)	3253.9(6.7)	6	−35
DSJC250.9	250	27897	0.90	8412	72	8286(75)	8288.8(6.8)	7	−126
DSJC500.1	500	12458	0.10	2951	12	2850(14)	2857.4(141.0)	9	−101
DSJC500.5	500	62624	0.50	11717	48	10910(51)	10918.2(54.1)	11	−807
DSJC500.9	500	112437	0.90	30872	126	29912(132)	29936.2(630.4)	15	−960
DSJC1000.1	1000	49629	0.10	10123	20	9003(22)	9017.9(91.1)	28	−1120
DSJC1000.5	1000	249826	0.50	43614	83	37598(87)	37673.8(2288.0)	24	−6016
DSJC1000.9	1000	449449	0.90	112749	223	103464(231)	103531(5070)	27	−9285
flat300_20_0	300	21375	0.48	-	20	3150(20)	3150.0(0.0)	3	-
flat300_26_0	300	21633	0.48	-	26	3966(26)	3966.0(0.0)	3	-
flat300_28_0	300	21695	0.48	-	28	4282(34)	4286.1(30.0)	3	-
flat1000_50_0	1000	245000	0.49	-	50	25500(50)	25500.0(0.0)	9	-
flat1000_60_0	1000	245830	0.49	-	60	30100(60)	30100.0(0.0)	11	-
flat1000_76_0	1000	246708	0.49	-	82	37167(86)	37213.2(575.0)	19	-
le450_15a	450	8168	0.08	-	15	2632(18)	2641.9(29.0)	5	-
le450_15b	450	8169	0.08	-	15	2642(19)	2643.4(7.2)	7	-
le450_15c	450	16680	0.17	-	15	3866(24)	3868.9(9.8)	6	-
le450_15d	450	16750	0.17	-	15	3921(26)	3928.5(36.5)	5	-
le450_25a	450	8260	0.08	-	25	3153(26)	3159.4(12.4)	7	-
le450_25b	450	8263	0.08	-	25	3366(26)	3371.9(18.0)	6	-
le450_25c	450	17343	0.17	-	25	4515(31)	4525.4(161.8)	8	-
le450_25d	450	17425	0.17	-	25	4544(31)	4550.0(24.0)	7	-
latin_sqr_10	900	307350	0.76	-	98	42223(109)	42392.7(4445)	4	-
C2000.5	2000	999836	0.50	-	148	132515(150)	132682(7342)	656	-
C4000.5	4000	4000268	0.50	-	271	473234(266)	473211(1027)	2588	-

time in minutes ($T[min.]$). The last column indicates the difference (gain) in sum of colors between our best results (column *Sum*) and the current best known results (column Sum^*).

Concerning the 29 DIMCAS graphs, one observes from Table 7.1 that our EXSCOL algorithm obtains excellent results (columns 7) with respect to the current best known results reported in the literature (column 5). Indeed, for 8 of the 12 random DSJC graphs used in the literature, our EXSCOL algorithm significantly improves the best known results by reducing largely the sum values (column 10). For the remaining 17 graphs, we report for the first time computational results for the sum coloring problem. EXSCOL needs from 1 to 27 minutes to achieve these results except for the two huge Cxxxx.5 graphs for which much larger computing times are required. Finally Table 7.1 discloses that for this set of instances, the results may vary with large standard deviation for several graphs.

Concerning the 27 COLOR02 graphs, one notices from Table 7.2 that EXSCOL improves the current best known results for 2 graphs while equaling the best ones for 22 graphs. Only for 3 graphs, EXSCOL obtains a worse result. For the 20 graphs used in the literature, EXSCOL achieves its results within less than 5 minutes and with small

Table 7.2: Computational results of the upper bounds on 23 COLOR02 benchmarks.

Instance	$\|V\|$	$\|e\|$	*Den*	*Sum**	k^*	EXSCOL			
						Sum(k)	*Avg.*(*Std.*)	T[*min.*]	Diff
myciel3	11	20	0.40	21	4	21(4)	21.0(0.0)	1	0
myciel4	23	71	0.28	45	5	45(5)	45.0(0.0)	1	0
myciel5	47	236	0.22	93	6	93(6)	93.0(0.0)	1	0
myciel6	95	755	0.17	189	7	189(7)	189.0(0.0)	2	0
myciel7	191	2360	0.13	381	8	381(8)	381.0(0.0)	2	0
anna	138	493	0.05	276	11	283(11)	283.2(0.2)	2	7
david	87	406	0.11	237	11	237(11)	238.1(1.0)	1	0
huck	74	301	0.11	243	11	243(11)	243.8(1.0)	1	0
jean	80	254	0.08	217	10	217(10)	217.3(0.2)	1	0
queen5.5	25	160	0.53	75	5	75(5)	75.0(0.0)	1	0
queen6.6	36	290	0.46	138	7	150(10)	150.0(0.0)	1	12
queen7.7	49	476	0.40	196	7	196(7)	196.0(0.0)	1	0
queen8.8	64	728	0.36	291	9	291(9)	291.0(0.0)	1	0
games120	120	638	0.09	446	9	443(9)	447.9(3.2)	2	-3
miles250	128	387	0.05	325	8	328(9)	333.0(7.6)	2	3
miles500	128	1170	0.14	712	20	709(20)	714.5(26.8)	2	-3
wap05	905	43081	0.10	-	50	13680(51)	13718.4(1047.1)	21	-
wap06	947	43571	0.10	-	46	13778(48)	13830.9(905.4)	27	-
wap07	1809	103368	0.06	-	46	28629(51)	28663.8(721.8)	112	-
wap08	1870	104176	0.06	-	45	28896(51)	28946.0(1361.2)	127	-
qg.order30	900	26100	0.06	-	30	13950(30)	13950.0(0.0)	28	-
qg.order40	1600	62400	0.05	-	40	32800(40)	32800.0(0.0)	35	-
qg.order60	3600	212400	0.03	-	60	110925(74)	110993.0(7054.9)	87	-
zeroin.i.2	211	3541	0.16	1004	30	1004(30)	1004(0.0)	186	0
zeroin.i.3	206	3540	0.17	998	30	998(30)	998(0.0)	228	0
fpsol2.i.1	496	11654	0.09	3403	65	3403(65)	3403(0.0)	181	0
inithx.i.1	864	18707	0.05	3676	54	3676(54)	3676(0.0)	292	0

standard deviations. For the 7 additional (large) graphs, EXSCOL requires a computing time ranging from 21 to 292 minutes.

7.4.3 Comparison with other algorithms for computing upper bounds

In this section, we compare our EXSCOL algorithm with 4 recent reference algorithms in the literature for computing upper bounds of the MSCP: Hybrid Local Search (HLS) [Douiri and Elbernoussi, 2011], MRLF [Li *et al.*, 2009], Parallel Genetic Algorithm (PGA) [Kokosiński and Kawarciany, 2007], Tabu Search (TS) [Bouziri and Jouini, 2010]. The comparisons are based on the criterion of quality, i.e. the smallest sum of colors reached by a given algorithm. Notice that information like computing time are not available for the reference algorithms.

Table 7.3 and Table 7.4 show the best results of our EXSCOL algorithm compared with these reference algorithms. Column 2 recall the current best known *Sum**. Columns 3-6 present the best results obtained by these reference algorithms. For indicative purposes, the number of colors k required by the best sum coloring is given in brackets. Notice that the same authors reported different results for some graphs for the MRLF heuristic in [Li *et al.*, 2009] and [Moukrim *et al.*, 2010]. For these graphs, Table 7.3 and Table 7.4 show the best of them.

Table 7.3: Comparison results of upper bounds on 12 graphs of the DIMACS challenge benchmarks.

Instance	*Sum**	sum coloring algorithms		
		EXSCOL	MRLF (2009,2010)	TS (2010)
DSJC125.1	326	**326**(7)	352(6)	344(6)
DSJC125.5	1015	**1017**(20)	1141(21)	1103(18)
DSJC125.9	2511	**2512**(44)	2653(50)	2631(49)
DSJC250.1	977	**985**(10)	1068(10)	1046(10)
DSJC250.5	3281	**3246**(31)	3658 (34)	3779(33)
DSJC250.9	8412	**8286**(75)	8942(83)	9198(82)
DSJC500.1	2951	**2850**(14)	3229(15)	3205(15)
DSJC500.5	11717	**10910**(51)	12717(60)	-
DSJC500.9	30872	**29912**(132)	32703(148)	-
DSJC1000.1	10123	**9003**(22)	10276(25)	-
DSJC1000.5	43614	**37598**(87)	45408(104)	-
DSJC1000.9	112749	**103464**(231)	119111(265)	-

The symbol '-' means that the related statistics are not available. The best *sum* values are highlighted in bold.

Table 7.4: Comparison results of upper bounds on 16 graphs of the COLOR02 benchmarks.

Instance	*Sum**	sum coloring algorithms			
		EXSCOL	HLS (2011)	MRLF (2009,2010)	PGA (2007)
myciel3	21	21(4)	21(4)	21(-)	21(4)
myciel4	45	45(5)	45(5)	45(-)	45(5)
myciel5	93	93(6)	93(6)	93(-)	93(6)
myciel6	189	189(7)	189(7)	189(-)	189(7)
myciel7	381	381(8)	381(8)	381(-)	382(8)
anna	276	283(11)	-(-)	**277**(-)	281(11)
david	237	**237**(11)	-(-)	241(-)	243(11)
huck	243	243(11)	243(11)	244(-)	243(11)
jean	217	217(10)	-(-)	217(-)	218(10)
queen5.5	75	75(5)	-(-)	75(-)	75(5)
queen6.6	138	150(10)	**138**(8)	**138**(-)	**138**(8)
queen7.7	196	196(7)	-(-)	196(-)	196(7)
queen8.8	291	**291**(9)	-(-)	303(-)	302(10)
games120	446	**443**(9)	446(9)	446(-)	460(9)
miles250	325	**328**(9)	343(10)	334(-)	347(8)
miles500	712	**709**(20)	755(22)	715(-)	762(20)

The symbol '-' means that the related statistics are not available. The best *sum* values are highlighted in bold.

From Table 7.3, one observes that, for the 12 DIMACS random graphs, EXSCOL dominates the TS and MRLF algorithms. For each of these graphs, our EXSCOL algorithm obtains a better sum coloring compared with the reference algorithms. One notices that in most cases, the colorings of EXSCOL require a smaller number of colors.

Table 7.4 discloses that EXSCOL competes favorably with the three reference algorithms (HLS, MRLF, PGA) on the set of 16 COLOR02 instances. Indeed, for each of these three algorithms, our EXSCOL algorithm obtains worse results for at most one instance while better results for at least 4 instances.

7.4.4 Improved lower bounds for the MSCP

Tables 7.5 and 7.6 respectively show the computational statistics of the lower bounds obtained by EXSCOL algorithm on the DIMACS benchmarks and the COLOR02 instances. In both tables, columns 2–4 indicate the features of the tested graphs, including the number of vertices ($|V|$), the number of edges ($|E|$) and the density of the graph (Den). Columns 5 and 6 give respectively the current best upper and lower bounds reported in the literature [Bouziri and Jouini, 2010; Douiri and Elbernoussi, 2011; Douiri and Elbernoussi, 2012; Helmar and Chiarandini, 2011; Kokosiński and Kawarciany, 2007; Li *et al.*, 2009; Moukrim *et al.*, 2010; Wu and Hao, 2012b]. The results of our EXSCOL approach are given in columns 7-9, including the best lower bounds found by our EXSCOL approach over the 20 runs, the averaged lower bound value with the standard deviation between parentheses and the average CPU time in seconds.

Concerning the 29 DIMACS instances, from Table 7.5, we observe that the results obtained by EXSCOL are very competitive when compared to the current best lower bounds reported in the literature. Indeed, for the 12 random DSJC graphs with a known lower bound, we managed to improve on the current best bounds in all the cases. For the remaining 17 DIMACS graphs, we report the computational statistics for the lower bounds for the first time. Finally, Table 7.5, also discloses that for most of these DIMACS instances, the gaps between the best known upper bounds and our best lower bounds are still large.

Concerning the 27 COLOR02 instances, from Table 7.6, we observe that our EXSCOL algorithm is able to improve the current best known lower bounds in the literature for 2 instances (fpsol2.i.1 and inithx.i.1) while equaling the best known lower bounds for 16 instances. Only for 2 instances (david and miles500), EXSCOL obtains a worse result. For the other 7 COLOR02 instances, we report for the first time lower bounds. Table 7.6 also indicates that we are able to prove optimality for 10 instances. Among these 10 instances, the optimality of 4 instances (qg.order30, qg.order40, fpsol2.i.1 and inithx.i.1) is proven for the first time.

Table 7.5: Computational results of the lower bounds obtained by EXCLIQUE on 29 DIMACS challenge benchmarks.

Instance	$\lvert V\rvert$	$\lvert E\rvert$	*Den*	UB^*	LB^*	EXCLIQUE		
						LB	*Avg.*(*Std.*)	*T*[*second*]
DSJC125.1	125	736	0.09	326	238	**246**	244.10(0.94)	80
DSJC125.5	125	3891	0.50	1015	504	**536**	522.40(5.46)	35
DSJC125.9	125	6961	0.89	2511	1621	**1664**	1592.50(26.61)	47
DSJC250.1	250	3218	0.10	977	537	**567**	561.95(1.96)	46
DSJC250.5	250	15668	0.50	3246	1150	**1270**	1258.80(5.44)	37
DSJC250.9	250	27897	0.90	8286	3972	**4179**	4082.40(51.66)	158
DSJC500.1	500	12458	0.10	2850	1163	**1250**	1246.55(1.37)	1269
DSJC500.5	500	62624	0.50	10910	2616	**2921**	2902.60(11.94)	60
DSJC500.9	500	112437	0.90	29912	10074	**10881**	10734.50(74.30)	276
DSJC1000.1	1000	49629	0.10	9003	2499	**2762**	2758.55(2.13)	5193
DSJC1000.5	1000	249826	0.50	37598	5787	**6708**	6665.90(14.49)	155
DSJC1000.9	1000	449449	0.90	103464	23863	**26557**	26300.25(84.04)	2741
flat300_20_0	300	21375	0.48	3150	-	**1524**	1505.65(6.78)	35
flat300_26_0	300	21633	0.48	3966	-	**1525**	1511.40(8.40)	34
flat300_28_0	300	21695	0.48	4282	-	**1532**	1515.25(7.81)	43
flat1000_50_0	1000	245000	0.49	25500	-	**6601**	6571.80(15.54)	118
flat1000_60_0	1000	245830	0.49	30100	-	**6640**	6600.50(18.01)	414
flat1000_76_0	1000	246708	0.49	37167	-	**6632**	6583.15(17.53)	98
le450_15a	450	8168	0.08	2632	-	**2329**	2313.65(15.32)	252
le450_15b	450	8169	0.08	2642	-	**2343**	2315.65(15.05)	600
le450_15c	450	16680	0.17	3866	-	**2591**	2545.30(24.67)	187
le450_15d	450	16750	0.17	3921	-	**2610**	2572.40(24.13)	175
le450_25a	450	8260	0.08	3153	-	**2997**	2964.40(28.08)	967
le450_25b	450	8263	0.08	3366	-	**3305**	3304.10(0.70)	1550
le450_25c	450	17343	0.17	4515	-	**3619**	3597.10(11.82)	689
le450_25d	450	17425	0.17	4544	-	**3684**	3627.35(45.33)	850
latin_sqr_10	900	307350	0.76	42223	-	**40950**	40950(0.00)	15
C2000.5	2000	999836	0.50	132515	-	**15091**	15077.60(11.74)	3994
C4000.5	4000	4000268	0.50	473234	-	**33033**	33018.80(11.42)	14413

The symbol '-' means that the related statistics are not available. The best *sum* values are highlighted in bold.

Table 7.6: Computational results of the lower bounds obtained by EXCLIQUE on 27 COLOR02 challenge benchmarks.

Instance	$\|V\|$	$\|E\|$	*Den*	UB^*	LB^*	EXCLIQUE		
						LB	*Avg.*(*Std.*)	*T*[*second*]
myciel3	11	20	0.40	21	16	16	16(0.00)	25
myciel4	23	71	0.28	45	34	34	34(0.00)	42
myciel5	47	236	0.22	93	70	70	70(0.00)	73
myciel6	95	755	0.17	189	142	142	142(0.00)	93
myciel7	191	2360	0.13	381	286	286	286(0.00)	148
anna	138	493	0.05	277	273	273	273(0.00)	168
david	87	406	0.11	241	234	*229*	229(0.00)	73
huck	74	301	0.11	243	243	243	243(0.00)	58
jean	80	254	0.08	217	216	216	216(0.00)	67
queen5.5	25	160	0.53	75	75	75	75(0.00)	19
queen6.6	36	290	0.46	138	126	126	126(0.00)	28
queen7.7	49	476	0.40	196	196	196	196(0.00)	41
queen8.8	64	728	0.36	302	288	288	288(0.00)	66
games120	120	638	0.09	446	442	442	441.40(0.91)	105
miles250	128	387	0.05	334	318	318	316.15(0.35)	131
miles500	128	1170	0.14	715	686	*677*	671.35(3.27)	117
wap05	905	43081	0.10	13680	-	**12428**	12339.25(44.03)	6283
wap06	947	43571	0.10	13778	-	**12393**	12348.75(43.59)	5417
wap07	1809	103368	0.06	28629	-	**24339**	24263.82(52.18)	8359
wap08	1870	104176	0.06	28896	-	**24791**	24681.09(56.12)	9127
qg.order30	900	26100	0.06	13950	-	**13950**	13950(0.00)	474
qg.order40	1600	62400	0.05	32800	-	**32800**	32800(0.00)	1379
qg.order60	3600	212400	0.03	110925	-	**109800**	109800(0.00)	7507
fpsol2.i.1	496	11654	0.09	3403	3402	**3403**	3403(0.00)	2676
inithx.i.1	864	18707	0.05	3676	3581	**3676**	3676(0.00)	3689
zeroin.i.2	211	3541	0.16	1004	1004	1004	1004(0.00)	453
zeroin.i.3	206	3540	0.17	998	998	998	998(0.00)	442

The symbol '-' means that the related statistics are not available. The best *sum* values are highlighted in bold.

Table 7.7: Comparisons between EXSCOL and MaxIS. EXSCOL dominates the conventional independent set approach (extraction of one independent set each time).

Instance	*Sum**	sum coloring algorithms		
		EXSCOL	MaxIS	EXSCOL−MaxIS
DSJC125.1	326	326(7)	335(7)	-9(0)
DSJC125.5	1015	1017(20)	1047(21)	-30(-1)
DSJC125.9	2511	2512(44)	2599(50)	-87(-6)
DSJC250.1	977	985(10)	1001(11)	-16(-1)
DSJC250.5	3281	3246(31)	3377 (34)	-131(-3)
DSJC250.9	8412	8286(75)	8548(83)	-262(-8)
DSJC500.1	2951	2850(14)	2932(16)	-82(-2)
DSJC500.5	11717	10910(51)	11163(56)	-253(-5)
DSJC500.9	30872	29912(132)	30957(145)	-1045(-13)
DSJC1000.1	10123	9003(22)	9161(24)	-158(-2)
DSJC1000.5	43614	37598(87)	38452(93)	-854(-6)
DSJC1000.9	112749	103464(231)	108487(254)	-5023(-23)

7.5 Discussion and analysis

7.5.1 Influence of the method to extract independent sets

Our EXSCOL algorithm uses a heuristic method to extract at each iteration as many disjoint independent sets as possible. A conventional method like MaxIS [Bar-Noy *et al.*, 1998] extracts exactly one independent set each time. The MaxIS method can be considered as a simplified version of our EXSCOL algorithm where lines 6–15 of Algorithm 1 (see Section 7.2.2) are disabled. Thus for MaxIS, each time a (large) independent set is found in the graph, it is assigned the smallest available color and its vertices are removed from the graph. This process is repeated until the graph becomes empty.

In order to highlight the difference between these two methods, we carry out additional experiments on the 12 DIMACS random DSJC graphs and show a comparison between EXSCOL and MaxIS. We run both methods 20 times and report in Table 7.7 the best sum values together with the number of used colors in brackets. The results show a clear dominance of EXSCOL over MaxIS. Indeed, EXSCOL finds a better sum coloring than MaxIS for each graph. Now it is interesting to observe that even if MaxIS cannot compete with EXSCOL, MaxIS does compete favorably with any of the existing sum coloring algorithms in the literature (compare column 2 of Table 7.3 and column 4 of Table 7.7). This observation highlights the interest of the general independent set extraction approach for the sum coloring problem.

Finally, if one checks the number k of colors used in the solutions, one notices that the solutions of EXSCOL need in general fewer colors compared to the solutions obtained by MaxIS. This can be explained by the fact that by extracting at each iteration as many disjoint independent sets as possible, EXSCOL is able to pack more vertices in the extracted independent sets, reducing thus the needed color classes to pack the vertices of the graph.

7.5.2 Sum coloring v.s. graph coloring

Given the relation between sum coloring and vertex coloring, one would wonder whether an effective vertex coloring algorithm could remain effective to approximate the sum coloring problem. Indeed, one can use a graph vertex coloring algorithm to find k-colorings with k as small as possible. Since the sum value depends partially on k, this coloring approach could help to solve the sum coloring problem. Yet, this approach does not necessarily lead to a good sum coloring as shown in the example of Fig. 7.3. Indeed, while the *chromatic number* $\chi(G)$ for this graph is 3, a 4-coloring is needed to obtain the *chromatic sum* $\sum(G) = 15$, i.e. the strength $s(G) = 4$.

In this section we provide computational evidence to show the limit of this graph coloring based method with respect to the proposed EXSCOL method. For this purpose, we once again adopt the Memetic Coloring Algorithm (MACOL) [Lü and Hao, 2010]. As indicated in Section 5.3.3, MACOL is a recent and competitive graph coloring algorithm. For this experiment, we consider again the set of 12 DSJC random instances. For each graph, we apply MACOL to obtain a legal coloring with k^* colors where k^* is the smallest number of colors for which a k^*-coloring has ever been found by a graph coloring algorithm. For such a k^*-coloring, we assign color 1 to the largest color class, color 2 to the next largest color class and so on. Table 7.8 shows the comparative results between EXSCOL and MACOL for the chosen graphs. The differences of sum values obtained by the two methods are given in the last column.

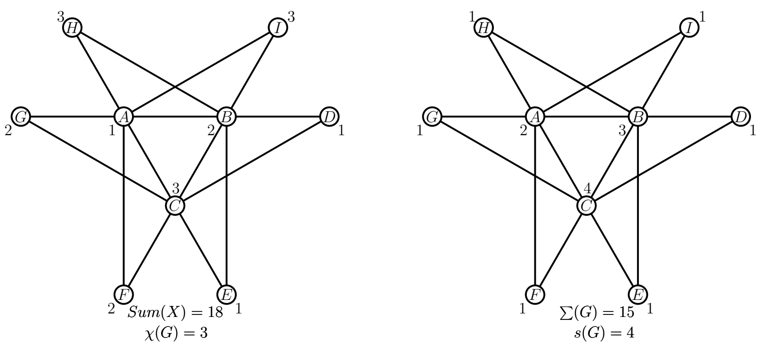


Figure 7.3: A coloring with a sum of colors equal to 18 using 3 colors (Left), and a coloring with a sum of colors equal to 15 using 4 colors (Right).

Table 7.8 shows clearly that EXSCOL reaches smaller sum values than MACOL for the tested graphs though EXSCOL may require more colors. In order to get some insights about this difference, we show in Table 7.9 additional information about the computational results for DSJC1000.5 reached by EXSCOL and MACOL.

Table 7.9 indicates the size of each independent set (column 'size $|I|$') and the number of extracted independent sets of size $|I|$ (column 'No. of IS of $|I|$') contained in the best solutions found by EXSCOL and MACOL. From Table 7.9, one notices that compared to the solution obtained by MACOL (a 83-coloring), the solution obtained by EXSCOL (a 87-coloring) contains much more independent sets of sizes 15 and 14 (which are the largest

Table 7.8: Comparisons between EXSCOL and MACOL. EXSCOL performs always better than the graph coloring approach.

Instance	*Sum**	sum coloring algorithms		
		EXSCOL	MACOL	EXSCOL-MACOL
DSJC125.1	326	326(7)	355(5)	-29(2)
DSJC125.5	1015	1017(20)	1058(17)	-41(3)
DSJC125.9	2511	2512(44)	2581(44)	-69(0)
DSJC250.1	977	985(10)	1071(8)	-86(2)
DSJC250.5	3281	3246(31)	3344(28)	-97(3)
DSJC250.9	8412	8286(75)	8372(72)	-86(3)
DSJC500.1	2951	2850(14)	3127(12)	-277(2)
DSJC500.5	11717	10910(51)	11176(48)	-266(3)
DSJC500.9	30872	29912(132)	30073(126)	-161(6)
DSJC1000.1	10123	9003(22)	9767(20)	-764(2)
DSJC1000.5	43614	37598(87)	38636(83)	-2038(4)
DSJC1000.9	112749	103464(231)	104203(223)	-739(8)

Table 7.9: Detailed computational results on DSJC1000.5.

EXSCOL		MACOL	
size $\|I\|$	No. of IS of $\|I\|$	size $\|I\|$	No. of IS of $\|I\|$
15	6	15	1
14	28	14	16
13	9	13	27
12	13	12	14
11	7	11	9
10	5	10	6
9	4	9	6
8	3	8	1
7	4	7	3
6	0	6	0
5	4	5	0
4	1	4	0
3	1	3	0
2	1	2	0
1	1	1	0

and second largest sizes). These independent sets cover a larger number of vertices and receive smaller colors, reducing thus the sum of colors.

Finally, even if MACOL performs worse than EXSCOL, its results remain competitive with other existing algorithms. This is the case because MACOL is a highly effective coloring algorithm able to find k-colorings with k equaling or being close to $\chi(G)$.

7.6 Conclusion

In this chapter, we applied the same idea of EXTRACOL proposed in Chapter 5 for the graph coloring to the MSCP, and presented EXSCOL, a heuristic algorithm based on independent set extraction for the minimum sum coloring problem. Instead of extracting independent sets one by one, the proposed algorithm tries to extract as many disjoint independent sets as possible each time. This strategy helps create more and large independent sets in the solution such that a larger number of vertices can be colored with small colors, thus leading to a smaller sum of colors.

We shown that with respect to the existing sum coloring heuristics, the proposed EXSCOL algorithm obtains highly competitive results in terms of both the upper bounds and lower bounds on a set of 56 DIMACS and COLOR02 benchmark graphs. In terms of upper bounds, among the 28 graphs used in the literature, EXSCOL improves the best known upper bounds for 10 graphs and equals the best known results for 11 graphs. In seven cases, EXSCOL fails to attain the best known sum values. In terms of lower bounds, our EXSCOL algorithm has improved on the current best lower bounds in 14 cases, proved optimality for the first time for 4 instances and attained the previously best bounds for 22 other instances. Only in two cases, our lower bounds are slightly worse than the best known lower bounds. The lower bounds obtained by EXSCOL on the set of 56 graphs are useful to assess the performance of both exact and heuristic algorithms for the sum coloring problem.

Finally, this study verified the (good) performance of the basic extraction method which extracts maximum independent sets one by one and the interest of the method based on applying directly powerful graph vertex coloring algorithms, even though these two methods cannot compete with the proposed EXSCOL algorithm.

General Conclusion

Conclusions

This thesis is dedicated to developing effective heuristic approaches to solving computationally difficult combinatorial optimization problems on graphs, namely the maximum clique problem and its two generalizations, graph coloring, minimum sum coloring, set packing and related problems. The application areas of the considered problems include information retrieval, classification theory, economics, scheduling, experimental design, and computer vision among many others. For each of these problems, new methods for finding good quality approximate solutions in reasonable computational times are introduced. All the algorithms are implemented and thoroughly tested on a number of benchmark instances from diverse application areas. The proposed methods are favorably competitive with other stat-of-the-art approaches.

After an overview of the most representative exact and heuristic approaches proposed in the literature for the maximum clique problem in Chapter 1, we presented an adaptive multistart tabu search algorithm (AMTS) for the maximum clique problem in Chapter 2. The proposed AMTS algorithm integrates some distinguishing features such as a constrained neighborhood, a dynamic tabu tenure mechanism and a long term memory based restart strategy. We extensively evaluated and compared the performance of AMTS with the current best-performing approaches from the literature, using the graph instances from DIMACS benchmarks. AMTS shows an excellent performance on the complete set of 80 standard DIMACS benchmark instances. Only for one instances (i.e., MANN_a81), AMTS failed to attain the best known results.

In Chapter 3, we presented a multi-neighbor tabu search algorithm (MN/TS) for the maximum vertex weight clique problem (MVWCP) based on a combined neighborhood induced by three types of moves. The MN/TS algorithm explores all these moves at each iteration and selects the best admissible (non-tabu or globally improving) solution that yields the largest weight gain. The tabu mechanism creates an effective local diversification and a multistart strategy is employed to create a global diversification. The proposed MN/TS algorithm is evaluated on a large number of MVWCP benchmarks from the BHOSLIB-W and DIMACS-W test sets (containing 40 instances and 80 instances, respectively) and is also applied to 16 instances derived from the set partitioning problem. Compared with leading reference algorithms from the literature, our MN/TS algorithm finds new best solutions in 26 cases (24 DIMACS-W instances and 2 set packing instances). Moreover, we also provided an analysis to show the relevance of the union combination of the underlying neighborhoods by comparing it to the sequential exploration of these neighborhoods. The outcomes suggest that the union combination of neighborhoods plays a key role in contributing to the effectiveness of the proposed algorithm.

In Chapter 4, we considered the maximum edge weight clique problem (MEWCP), which is another important generalization of the MCP, and proposed a memetic algorithm

(MAMEP) to tackle this problem. The proposed MAMEP algorithm integrates a dedicated crossover operator and a constrained neighborhood tabu search procedure. The proposed crossover operator tries to preserve the vertices shared by the parent solutions which hopefully belong to the optimal solution. Experimental evaluations on a large collection of 7 sets of 120 instances from the literature showed that the proposed MAMEP algorithm attains consistently the previous best known results within a time limit ranging from 20 seconds to 30 minutes. Specifically, for 101 out of 120 cases (84%), MAMEP reaches the previous best known objective value for each of its runs (a successful rate of 100%). More importantly, for 6 large and very challenging instances, the proposed MAMEP algorithm is able to yield improved solutions with respect to the current best known results.

In Chapter 5, we applied the AMTS algorithm to the graph coloring problem and proposed an effective heuristic approach based on independent set extraction (EXTRACOL) to coloring large graphs. The proposed EXTRACOL approach uses a preprocessing method to extract large independent sets from the graph and a memetic algorithm to color the residual graph. Each preprocessing application identifies with AMTS a number of pairwise disjoint independent sets of a given size in order to maximize the vertices removed from the graph. We evaluate EXTRACOL on the 11 largest graphs of the DIMACS challenge benchmarks and show improved results for 4 very difficult graphs (DSJC1000.9, C2000.5, C2000.9, C4000.5). The limitation of the proposed EXTRACOL algorithm is also analyzed for the purpose of further improvement.

Chapter 6 presented an extraction and expansion method (IE^{2}COL) to further improve EXTRACOL. To remedy the limitation of the EXTRACOL algorithm, from the reduced graph, IE2COL triggers a backward coloring process which uses extracted independent sets as new color classes for intermediate subgraph coloring. We evaluated the performance of IE^{2}COL on the same set of 11 benchmark instances used by EXTRACOL. Computational results showed that IE^{2}COL provides new upper bounds for 4 graphs and matches consistently the current best known results for 6 other graphs.

In the last chapter, we applied the same idea of the EXTRACOL algorithm for graph coloring to the minimum sum coloring problem, and presented an effective heuristic algorithm based on independent set extraction (EXSCOL) for sum coloring of graphs. Instead of extracting independent sets one by one, the proposed algorithm tries to extract as many disjoint independent sets as possible at each iteration. This strategy helps create more and large independent sets in the solution such that a larger number of vertices can be colored with small colors, thus leading to a smaller sum of colors. The proposed EXSCOL algorithm obtains highly competitive results on a set of 56 DIMACS and COLOR02 benchmark graphs. For a large number of instances, EXTRACOL is able to improve the previous best known results in both terms of lower and upper bounds.

Perspectives

The maximum clique problem is an important combinatorial optimization problem with a wide range of important applications. There are a lot of real-world problems that can be directly modeled by the maximum clique problem. In future work, we would like to extend our work to practical applications of these real-world problems. For instance, the protein structure alignment problem in bioinformatics can be directly translated to a maximum clique problem and a lot of exact algorithms based on the maximum clique problem have been developed to solve the protein structure alignment problem. However, for large and hard instances, heuristic approaches are required to find near-optimal solutions. We have applied our AMTS algorithm to the protein structure alignment problem and remarkable results have been achieved by our AMTS algorithm. For nearly all the tested instances with known optimal solutions, our AMTS algorithm is able to reach the optimal solution but requiring much less computing time than these exact approaches. Still, our AMTS algorithm can be further improved to be adapted to the protein structure alignment problem by taking into account the special structure information of the protein structure alignment problem. In addition to the protein structure alignment problem, there are many other real-world problems can be formulated as maximum clique problems. For these problems, we can exploit properties of the problem and integrate the specialized information within our AMTS algorithm to adapt AMTS to these applications.

In addition to practical applications in the real world, a variety of important combinatorial optimization problems are tightly related to the maximum clique problem. A large part of this dissertation is devoted to applications of our clique based approaches to these combinatorial optimization problems such as set packing, graph coloring and sum coloring. Besides these 3 problems, our clique based approaches can find applications in other combinatorial optimization problems. In particular, some of these problems can even be directly transformed to a maximum clique problem and thus can be directly modeled by our clique based approaches. For instance, the combinatorial auction problem can be modeled as a set packing problem, for which our MN/TS algorithm shows excellent performance. Thus we can apply our MN/TS algorithm to the combinatorial auction problem. To further improve the performance of the basic clique based approaches on these problems, we still need to exploit properties of these problems and combine our clique based approaches with some existing effective search strategies dedicated to these problems.

Finally, for the sum coloring problem, our proposed EXSCOL algorithm shows excellent performance on large instances with more that 300 vertices. However, for some small instances with less than 250 vertices, it is less effective than some local search sum coloring algorithms (For instance, the MDS algorithm [Helmar and Chiarandini, 2011] and the BLS algorithm [Benlic and Hao, 2012]). This may be explained by the fact that EXSCOL can be viewed as a greedy algorithm. The EXSCOL algorithm could be improved by following two directions. First, we can combine the EXSCOL algorithm with these effective local search algorithms dedicated to sum coloring so as to take advantages from each other. This can be achieved by first applying the main procedure of EXSCOL to reduce the graph until the residual graph becomes sufficiently small and then coloring the residual graph with these effective sum coloring algorithms. Second, the above approach could be further improved by allowing the sum coloring algorithm to reconsider the independent sets extracted by EXSCOL. Such an improvement can be for instance achieved within the multi-level optimization paradigm.

List of figures

List of tables

List of algorithms

List of publications

International journals

- Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. Computers & Operations Research 39(2): 283-290, 2012.
- Qinghua Wu and Jin-Kao Hao. An effective heuristic algorithm for sum coloring of graphs. Computers & Operations Research 39(7): 1593-1600, 2012.
- Qinghua Wu and Jin-Kao Hao. Memetic search for the max-bisection problem. Computers & Operations Research 40(1): 166179, 2013.
- Qinghua Wu, Jin-Kao Hao, Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. Annals of Operations Research 196(1): 611–634, 2012.
- Qinghua Wu and Jin-Kao Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. Accepted to Journal of Combinatorial Optimization Nov 2011 and to appear in 2013.
- Qinghua Wu and Jin-Kao Hao. An extraction and expansion approach for graph coloring. Accepted to Asia-Pacific Journal of Operational Research May 2011 and to appear in 2013.
- 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.

International conferences

- Qinghua Wu and Jin-Kao Hao. A Memetic Approach for the Max-Cut Problem. In C. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.): PPSN 2012, Lecture Notes in Computer Science 7492: 297-306, 2012.

Submitted and revised papers

- Qinghua Wu and Jin-Kao Hao. Improved lower bounds for sum coloring via clique decomposition, submitted to Discrete Applied Mathematics. June 2012.
- Qinghua Wu and Jin-Kao Hao. A Hybrid Metaheuristic Method for the Maximum Diversity Problem, submitted to European Journal of Operational Research. September 2012

References

[Alidaee *et al.*, 2007] cited page 9, 71, 71, 71
B. Alidaee, G. Kochenberger F. Glover, and H. Wang. Solving the maximum edge weight clique problem via unconstrained quadratic programming. *European Journal of Operational Research*, 181(2):592–597, 2007.

[Alidaee *et al.*, 2008] cited page 26, 59, 59, 59, 59
B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang. A new approach for modeling and solving set packing problems. *European Journal of Operational Research*, 186:204–582, 2008.

[Aringhieri and Cordone, 2011] cited page 71, 71
R. Aringhieri and R. Cordone. Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society*, 62(2):266–280, 2011.

[Aringhieri *et al.*, 2008] cited page 71
R. Aringhieri, R. Cordone, and Y. Melzani. Tabu search versus grasp for the maximum diversity problem. *4OR-A Quarterly Journal of Operations Research*, 6(1):45–60, 2008.

[Arora and Safra, 1992] cited page 9
S. Arora and S. Safra. Approximating clique is np-complete. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13, Piscataway, NJ, 1992.

[Arora *et al.*, 1998] cited page 9, 9
S. Arora, C. Lund, R. Motwani, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 18:501–555, 1998.

[Ausiello *et al.*, 1999] cited page 71
G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A.M. Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, 1999.

[Avanthay *et al.*, 2003] cited page 95
C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[Avondo-Bodeno, 1962] cited page 20
G. Avondo-Bodeno. *Economic Applications of the Theory of Graphs,Gordon and Breach.* Science Publishers, New York, 1962.

[Babel, 2008] cited page 24, 47
L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52:31–38, 2008.

[Balas and Yu, 1986] cited page 10
E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15:1054–1068, 1986.

[Ballard and Brown, 1982] cited page 20, 47
D.H. Ballard and M. Brown. *Computer Vision.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[Bar-Noy *et al.*, 1998] cited page 127, 127, 127, 136
A. Bar-Noy, M. Bellareb, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

[Barahona *et al.*, 1992] cited page 20
F. Barahona, A. Weintraub, and R. Epstein. Habitat dispersion in forest planning and the stable set problem. *Operations Research*, 1:S14–S21, 1992.

[Battiti and Brunato, 2008] cited page 18
R. Battiti and M. Brunato. Reactive search and intelligent optimization. *ser. Operations Research/Computer Science Interfaces 45*, 2008.

[Battiti and Mascia, 2010] cited page 63
R. Battiti and F. Mascia. Reactive and dynamic local search for the max-clique problem: engineering effective building blocks. *Computers and Operations Research*, 37:534–542, 2010.

[Battiti and Protasi, 2001] cited page 13, 13, 14, 23, 37, 38, 54
R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.

[Battiti and Tecchiolli, 1994] cited page 14
R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[Beasley, 1996] cited page 25
J.E. Beasley. Obtaining test problems via internet. *Journal of Global Optimization*, 8:429–433, 1996.

[Benlic and Hao, 2012] cited page 145
U. Benlic and J.K. Hao. A study of breakout local search for the minimum sum coloring

problem. In *L. T. Bui et al. (Eds.) SEAL 2012, Lecture Notes in Computer Science*, pages 7673: 128–137, 2012.

[Berman and Pelc, 1990] cited page 20
P. Berman and A. Pelc. Distributed fault diagnosis for multiprocessor systems. In *Proceedings of the 15th Annual International Symposium on Fault-Tolerant Computing*, pages 340–346, Newcastle, UK, 1990.

[Blöchliger and Zufferey, 2008] cited page 95, 120
I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35(3):960–975, 2008.

[Bomze *et al.*, 2000] cited page 18, 47
I. M. Bomze, M. Pelillo, and V. Stix. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 11(6):1228–1241, 2000.

[Bomze *et al.*, 2012] cited page 18
I.M. Bomze, L. Grippo, and L. Palagi. Unconstrained formulation of standard quadratic optimization problems. *Top*, 20:35–51, 2012.

[Bomze, 1997] cited page 18
I.M. Bomze. Evolution towards the maximum clique. *Journal of Global Optimization*, 10:143–164, 1997.

[Bouziri and Jouini, 2010] cited page 127, 132, 134, 136
H. Bouziri and M. Jouini. A tabu search approach for the sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36(1):915–922, 2010.

[Brimberg *et al.*, 2009] cited page 25, 25, 71, 71, 81, 86
J. Brimberg, N. Mladenović, D. Urošević, and E. Ngai. Variable neighborhood search for the heaviest k-subgraph. *Computers and Operations Research*, 36(11):2885–2891, 2009.

[Bringmann and Friedrich, 2009] K. Bringmann and T. Friedrich. Don't be greedy when calculating hypervolume contributions. In *FOGA'09: Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, pages 103–112, New York, NY, USA, 2009. ACM.

[Brotcorne *et al.*, 2002] cited page 23
L. Brotcorne, G. Laporte, and F. Semet. Fast heuristic for large scale coveringlocation problems. *Computers & Operations Research*, 29:651–665, 2002.

[Brouwer *et al.*, 1990] cited page 20
A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W. D. Smith. A new table of constant weight codes. *IEEE Transactions on information theory*, 36:1334–1380, 1990.

[Brunato and Battiti, 2011] cited page 18, 23
M. Brunato and R. Battiti. R-evo: A reactive evolutionary algorithm for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 15(6):770–782, 2011.

[Bui and Rizzo, 2004] cited page 19
T.N. Bui and J.R. Rizzo. Finding maximum cliques with distributed ants. In *Genetic and Evolutionary Computation - GECCO 2004, volume 3102 of LNCS*, pages 24–35. Springer-Verlag, 2004.

[Burer *et al.*, 2002] cited page 18
S. Burer, R.D.C Monteiro, and Y. Zhang. Maximum stable set formulations and heuristics based on continuous optimization. *Mathematical Programming*, 94(1):137–166, 2002.

[Burke *et al.*, 2003] cited page 17
E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Glover, F. (ed.) Handbook of Meta-heuristics*, pages 457–474, Norwell, 2003. Kluwer Academic.

[Burke *et al.*, 2007] cited page 95
E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper heuristic for timetabling problems. *European Journal of Operational Research*, 176:177–192, 2007.

[Busygin, 2006] cited page 18, 38, 47
S. Busygin. A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics*, 154(15):2080–2096, 2006.

[Carraghan and Pardalos, 1990] cited page 10, 10, 10, 19
R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[Carter and Park, 1993] cited page 17
B. Carter and K. Park. How good are genetic algorithms at finding large cliques: An experimental study. Technical Report BU-CS-93-015, Department of Computer Science, Boston University, 1993.

[Chams *et al.*, 1987] cited page 21, 21, 21, 22, 95, 96, 96, 96
M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

[Chandra and Halldórsson, 2001] cited page 71
B. Chandra and M.M. Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, 38:438–465, 2001.

[Chiarandini and Stützle, 2002] cited page 95, 120
M. Chiarandini and T. Stützle. An application of iterated local search to graph coloring. *In: D.S. Johnson, A. Mehrotra, M. Trick, editors, Proc. of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA*, pages 112–125, 2002.

[Christofides, 1975] cited page 20
N. Christofides. *Graph theory, An algorithmic approach.* Academic Press, 1975.

[Coello Coello *et al.*, 2006] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Costa *et al.*, 1995] cited page 95
D Costa, A Hertz, and O Dubuis. Embedding of a sequential procedure within an evolutionary algorithm for coloring problem in graphs. *Journal of Heuristics*, 1:105–28, 1995.

[Croce *et al.*, 2009] cited page 22, 22
F.D. Croce, A. Grosso, and M. Locatelli. A heuristic approach for the maxmin diversity problem based on max-clique. *Computers & Operations Research*, 36:2429–2433, 2009.

[de Andrade *et al.*, 2003] cited page 71
P.M.F. de Andrade, A. Plastino, L.S. Ochi, and S.L. Martins. Grasp for the maximum diversity problem. In *Proceedings of the Fifth Metaheuristics International Conference (MIC 2003)*, pages CD–ROM Paper: MIC0315, Kyoto, Japan, 2003.

[de Andrade *et al.*, 2005] cited page 71, 71
M.R.Q. de Andrade, P.M.F. de Andrade, S.L. Martins, and A. Plastino. Grasp with path-relinking for the maximum diversity problem. In *S.E. Nikoletseas (Ed.), Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005*, pages 558–569, Santorini Island, Greece, 2005. Springer, Berlin.

[Delorme *et al.*, 2004] cited page 59, 59, 59
X. Delorme, X. Gandibleux, and J. Rodriguez. Grasp for set packing problems. *European Journal of Operational Research*, 153:564–580, 2004.

[Deo, 1974] cited page 20
N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice-Hall, Englewwod Cliffs, 1974.

[Dijkhuizen and Faigle, 1993] cited page 71
G. Dijkhuizen and U. Faigle. A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 69:121–130, 1993.

[Dorigo and Caro, 1999] cited page 19
M. Dorigo and G. Di Caro. *The ant colony optimization meta-heuristic.* McGraw-Hill Ltd., Maidenhead, England, 1999.

[Dorigo *et al.*, 1996] cited page 19
M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26:29–41, 1996.

[Dorne and Hao, 1998] cited page 95
R. Dorne and J.K. Hao. A new genetic local search algorithm for graph coloring. *Lecture Notes in Computer Science*, 1498:745–754, 1998.

[Douiri and Elbernoussi, 2011] cited page 127, 132, 134, 136
S.M. Douiri and S. Elbernoussi. New algorithm for the sum coloring problem. *International Journal of Contemporary Mathematical Sciences*, 6(10):453–463, 2011.

[Douiri and Elbernoussi, 2012] cited page 131, 136
S.M. Douiri and S. Elbernoussi. A new ant colony optimization algorithm for the lower bound of sum coloring problem. *Journal of Mathematical Modelling and Algorithms*, 11(2):181–192, 2012.

[Duarte and Martí, 2007] cited page 24, 24, 24, 71, 74, 74
A. Duarte and R. Martí. Tabu search and grasp for the maximum diversity problem. *European Journal of Operational Research*, 178(1):71–84, 2007.

[Fahle, 2002] cited page 10
T. Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *Proceedings of ESA-2002*, pages 485–498, 2002.

[Feige and Kilian, 1998] cited page 9
U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57:187–199, 1998.

[Feige *et al.*, 1996] cited page 9
U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43:268–292, 1996.

[Feige *et al.*, 2001] cited page 71, 71
U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[Fenet and Solnon., 2003] cited page 19
S. Fenet and C. Solnon. Searching for maximum cliques with ant colony optimization. In *S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G.R. Raidl (eds.), Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003: EvoCOP, EvoIASP, EvoSTim, volume 2611 of LNCS*, pages 236–245, Málaga, Spain, 2003. Springer-Verlag.

[Feng *et al.*, 2010] cited page 71
B. Feng, Z.Z. Jiang, Z.P. Fan, and N. Fu. A method for member selection of crossfunctional teams using the individual and collaborative performances. *European Journal of Operational Research*, 203(3):652–661, 2010.

[Fleurent and Ferland, 1996] cited page 13, 14, 17, 21, 21, 22, 95, 95, 96, 96, 96, 96, 100, 104, 105
C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461, 1996.

[Fogel *et al.*, 1966] cited page 17
L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artiffcial Intelligence through Simulated Evolution.* John Wiley, New York, 1966.

[Foster and Soule, 1995] cited page 17
J.A. Foster and T. Soule. Using genetic algorithms to find maximum cliques. Technical Report TIK-Report CSM-450, Department of Computer Science, University of Idaho, 1995.

[Friden *et al.*, 1989] cited page 13, 13, 14, 27, 31
C. Friden, A. Hertz, and D. de Werra. Stabulus: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.

[Funabiki and Higashino, 2000] cited page 104, 104, 105, 120
N. Funabiki and T. Higashino. A minimal-state processing search algorithm for graph coloring problems. *IEICE Transaction Fundamentals*, E83–A:1420–1430, 2000.

[Galinier and Hao, 1999] cited page 33, 41, 95, 120
P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[Galinier and Hertz, 2006] cited page 95, 132
P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers and Operations Research*, 33(9):2547–2562, 2006.

[Galinier *et al.*, 2008] cited page 95, 102, 104, 105, 120
P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-colouring problem. *Discrete Applied Mathematics*, 156(2):267–279, 2008.

[Galinier *et al.*, 2011] cited page 77
P. Galinier, Z. Boujbel, and M.C. Fernandes. An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, 191(1):1–22, 2011.

[Gallego *et al.*, 2009] cited page 24, 71
M. Gallego, A. Duarte, M. Laguna, and R. Martí. Hybrid heuristics for the maximum diversity problem. *Computational Optimization and Applications*, 44(3):411–426, 2009.

[Gamache *et al.*, 2007] cited page 95
M. Gamache, A. Hertz, and J.O. Ouellet. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers and Operations Research*, 34(8):2384–2395, 2007.

[Garey and Johnson, 1976] cited page 9, 95
M.R. Garey and D.S. Johnson. The complexity of near-optimal coloring. *Journal of the ACM*, 23:43–49, 1976.

[Garey and Johnson, 1979] cited page 9, 95, 99
M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, New York, 1979.

[Gaspero and Schaerf, 2006] cited page 63
L. Di Gaspero and A. Schaerf. Neighborhood portfolio approach for local search applied

to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1):65–89, 2006.

[Gendreau *et al.*, 1993] cited page 14, 54
A. Gendreau, L. Salvail, and P. Soriano. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41:385–403, 1993.

[Geng *et al.*, 2007] cited page 13, 15
X. Geng, J. Xu, J. Xiao, and L. Pan. A simple simulated annealing algorithm for the maximum clique problem. *Information Sciences*, 177:5064–5071, 2007.

[Ghosh, 1996] cited page 71, 74
J.B. Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175–181, 1996.

[Glass, 2002] cited page 95
C. Glass. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society*, 53:544–551, 2002.

[Glover *et al.*, 1998] cited page 71
F. Glover, C.C. Kuo, and K.S. Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132, 1998.

[Glover, 1989] cited page 13, 27, 29, 33, 33, 51, 77
F. Glover. Tabu search - part i. *ORSA Journal onComputing*, 1(3):190–260, 1989.

[Gortázar *et al.*, 2010] cited page 71
F. Gortázar, A. Duarte, M. Laguna, and R. Martí. Black box scatter search for general classes of binary optimization problems. *Computers & Operations Research*, 37(11):1977–1986, 2010.

[Grossman, 1993] cited page 19
T. Grossman. Applying the inn model to the maxclique problem. *in [Johnson and Trick, 1996]*, 9(6):375–382, 1993.

[Grosso *et al.*, 2004] cited page 54
A. Grosso, M. Locatelli, and F.D. Croce. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics*, 14:135–152, 2004.

[Grosso *et al.*, 2008] cited page 37
A. Grosso, M. Locatelli, and W. Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14(6):587–612, 2008.

[Gu and Yu, 2004] cited page 20
S. Gu and S. Yu. A chaotic neural network for the maximum clique problem. *Advances in Artificial Intelligence, Lecture Notes in Computer Science*, 3060:391–405, 2004.

[Hamiez and Hao, 2002] cited page 95
J.P. Hamiez and J.K. Hao. Scatter search for graph coloring. *Lecture Notes in Computer Science*, 2310:168–179, 2002.

[Hansen and Jaumard, 1990] cited page 13
P. Hansen and B. Jaumard. Algorithms for the maximum satisfability problem. *Computing*, 44(4):279–303, 1990.

[Hansen and Mladenović, 2003] cited page 15
P. Hansen and N. Mladenović. *Handbook of Metaheuristics, chapter Variable neighborhood search.* Springer, 2003.

[Hansen *et al.*, 2004] cited page 15, 23
P. Hansen, N. Mladenović, and D. Urovsević. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145:117–125, 2004.

[Hao and Wu, 2012] cited page 3, 111
J.K. Hao and Q. Wu. Improving the extraction and expansion method for large graph coloring. *Discrete Applied Mathematics*, 160(16–17):2397–2407, 2012.

[Hao, 2011] cited page 17, 72, 78, 123
J.K. Hao. Memetic algorithms in discrete optimization. In *F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379, Chapter 6*, pages 73–94. Springer, 2011.

[Harley *et al.*, 2001] cited page 23
E. Harley, A. Bonner, and N. Goodman. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics*, 17:487–494, 2001.

[Hassin *et al.*, 1997] cited page 71
R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21:133–137, 1997.

[Hästad, 1999] cited page 9, 9
J. Hästad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[Helmar and Chiarandini, 2011] cited page 131, 136, 145
A. Helmar and M. Chiarandini. A local search heuristic for chromatic sum. *In L. D. Gaspero, A. Schaerf, T. Stützle (Eds.), Proceedings of the 9th Metaheuristics International Conference*, MIC:161–170, 2011.

[Hertz and de Werra, 1987] cited page 21, 21, 22, 95, 95, 96, 96, 96, 96, 100, 101
A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):344–351, 1987.

[Hertz *et al.*, 2008] cited page 95, 120
A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

[Holland, 1975] cited page 17
J.H. Holland. *Adaptation in natural and artiffcial systems.* The University of Michigan Press, Ann Arbor, 1975.

[Hopfield and Tank, 1985] cited page 19
J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, 1985.

[Hunting *et al.*, 2001] cited page 71, 71
M. Hunting, U. Faigle, and W. Kern. A lagrangian relaxation approach to the edge-weighted clique problem. *European Journal of Operational Research*, 131:119–131, 2001.

[J.C.Regin, 2003] cited page 11
J.C.Regin. Solving the maximum clique problem with constraint programming. In *Proceedings of CPAIOR'03,Springer, LNCS2883*, pages 634–648, 2003.

[Johnson and Trick, 1996] cited page 9, 101, 132, 162
D. S. Johnson and M. A. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, volume 26 of DIMACS Series.* American Mathematical Society, Providence, RI, 1996.

[Johnson *et al.*, 1991] cited page 21, 21, 22, 95, 95, 96, 96, 96, 100
D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–401, 1991.

[Johnson, 1974] cited page 12
D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[Katayama and Narihisa, 2004] cited page 71
K. Katayama and H. Narihisa. An evolutionary approach for the maximum diversity problem. In *W. Hart, N. Krasnogor, J.E. Smith (Eds.), Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, pages 31–47. Springer, Berlin, 2004.

[Katayama *et al.*, 2005] cited page 13, 13, 23, 37, 38, 54
K. Katayama, A. Hamamoto, and H.Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, 95(5):503–511, 2005.

[Kincaid, 1992] cited page 71
R.K. Kincaid. Good solutions to discrete noxious location problems via metaheuristics. *Annals of Operations Research*, 40:265–281, 1992.

[Kirkpatrick *et al.*, 1983] cited page 15
S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Kokosiński and Kawarciany, 2007] cited page 127, 132, 134, 136
Z. Kokosiński and K. Kawarciany. On sum coloring of graphs with parallel genetic algorithms. *Lecture Notes In Computer Science*, 4431:211–219, 2007.

[Konc and Janezic, 2007] cited page 11
J. Konc and D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry*, 58:569–590, 2007.

[Kopf and Ruhe, 1987] cited page 12, 12
R. Kopf and G. Ruhe. A computational study of the weighted independent set problem for general graphs. *Foundations of Control Engineering*, 12(4):167–180, 1987.

[Kubicka and Schwenk, 1989] cited page 127
E. Kubicka and A. J. Schwenk. An introduction to chromatic sums. *Proceedings of the 17th Annual ACM Computer Science Conference*, pages 39–45, 1989.

[Kuby, 1987] cited page 71
M.J. Kuby. Programming models for facility dispersion: the p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

[Kuo *et al.*, 1993] cited page 71
C.C. Kuo, F. Glover, and K.S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1993.

[Kuznetsova and Strekalovsky, 2001] cited page 18
A.A. Kuznetsova and A.S. Strekalovsky. On solving the maximum clique problem. *Journal of Global Optimization*, 21(3):265–288, 2001.

[Kwon, 2005] cited page 59
R.H. Kwon. Data dependent worst case bounds for weighted set packing. *European Journal of Operational Research*, 167(1):68–76, 2005.

[Laguna and Martí, 2001] cited page 95
M. Laguna and R. Martí. A grasp for coloring sparse graphs. *Computational optimization and applications*, 19(2):165–178, 2001.

[Laguna and Martrí, 2003] cited page 17
M. Laguna and R. Martrí. *Scatter Search. Methodology and Implementations in C.* Kluwer Academic Publishers, Boston MA, 2003.

[Larrañaga and Lozano, 2001] cited page 17
P. Larrañaga and J.A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Boston MA, 2001.

[Lecky *et al.*, 1989] cited page 20
J.E. Lecky, O.J. Murphy, and R.G. Absher. Graph theoretic algorithms for the pla folding problem. *IEEE transactions on computer-aided design of integrated circuits and systems*, 8(9):1014–1021, 1989.

[Leighton, 1979] cited page 22, 127
F. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[Li and Quan, 2010] cited page 11
C.M. Li and Z. Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 128–133, 2010.

[Li *et al.*, 2009] cited page 127, 132, 134, 134, 136
Y. Li, C. Lucet, A. Moukrim, and K. Sghiouer. Greedy algorithms for minimum sum coloring algorithm. In *Proceedings LT2009 Conference*, pages 619–652, Tunisia, 2009.

[Lin and Lee, 1993] cited page 19
F. Lin and K. Lee. A parallel computation network for the maximum clique problem. *IEEE International Symposium on Circuits and Systems*, pages 2549–2552, 1993.

[Lovász, 1979] cited page 20
L. Lovász. On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.

[Lozano *et al.*, 2011] cited page 25, 25, 71, 71, 81, 86
M. Lozano, D. Molina, and C. García-Martínez. Iterated greedy for the maximum diversity problem. *European Journal of Operational Research*, 214(1):31–38, 2011.

[Lü and Hao, 2010] cited page 95, 96, 97, 100, 101, 101, 102, 102, 103, 103, 103, 103, 114, 115, 117, 118, 119, 132, 140
Z. Lü and J.K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 200(1):235–244, 2010.

[Lü *et al.*, 2010] cited page 79, 79, 79, 80, 81
Z. Lü, F. Glover, and J.K. Hao. A hybrid metaheuristic approach to solving the ubqp problem. *European Journal of Operational Research*, 207(3):1254–1262, 2010.

[Lü *et al.*, 2011] cited page 63
Z. Lü, J.K. Hao, and F. Glover. Neighborhood analysis: a case study on curriculumbased course timetabling. *Journal of Heuristics*, 17(2):97–118, 2011.

[Lund and Yannakakis, 1994] cited page 9
C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41:960–981, 1994.

[Macambira and de Souza, 2000] cited page 24, 71
E.M. Macambira and C.C. de Souza. The edge-weighted clique problem: valid inequalities, facets and polyhedral computations. *European Journal of Operational Research*, 123:346–371, 2000.

[Macambira, 2003] cited page 71, 71, 71
E.M. Macambira. An application of tabu search heuristic for the maximum edge-weighted subgraph problem. *Annals of Operational Research*, 117:175–190, 2003.

[MacWilliams and Sloane, 1979] cited page 20
J. MacWilliams and N.J.A. Sloane. *The Theory of Error Correcting Codes.* Elsevier, Amsterdam, North-Holland, 1979.

[Malafiejski, 2004] cited page 127, 127
M. Malafiejski. Sum coloring of graphs. *Contemporary Mathematics*, 352:55–65, 2004.

[Malaguti *et al.*, 2008] cited page 95, 102, 103, 104, 105, 105, 118, 120, 132
E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

[Malod-Dognin *et al.*, 2010] cited page 21, 21
N. Malod-Dognin, R. Andonov, and N. Yanev. Maximum cliques in protein structure comparison. *Experimental Algorithms Lecture Notes in Computer Science*, 6049:106–117, 2010.

[Mannino and Stefanutti, 1999] cited page 47, 57, 59, 59, 59, 59
C. Mannino and E. Stefanutti. An augmentation algorithm for the maximum weighted stable set problem. *Computational Optimization and Applications*, 14:367–381, 1999.

[Marchiori, 1998] cited page 17, 23
E. Marchiori. A simple heuristic based genetic algorithm for the maximum clique problem. In *ACM Symposium on Applied Computing*, pages 366–373, 1998.

[Marchiori, 2002] cited page 18, 18, 19
E. Marchiori. Genetic, iterated and multistart local search for the maximum clique problem. In *Applications of Evolutionary Computing (EvoCOP 2002), LNCS 2279*, pages 112–121, Verlag, 2002. Springer.

[Martí *et al.*, 2011] cited page 24, 71, 71, 71, 71, 82, 82
R. Martí, M. Gallego, A. Duarte, and E.G. Pardo. Hybrid heuristics for the maximum diversity problem. *Computational Optimization and Applications*, 2011.

[Massaro *et al.*, 2001] cited page 18
A. Massaro, M. Pelillo, and I.M. Bomze. A complementary pivoting approach to the maximum weight clique problem. *SIAM: SIAM Journal on Optimization*, 12(4):928–948, 2001.

[Miller, 1993] cited page 20
W. Miller. Building multiple alignments from pairwise alignments. *Computer Applications in the Biosciences*, 9(2):169–176, 1993.

[Morgenstern, 1996] cited page 95, 95, 104, 105, 120
C. Morgenstern. Distributed coloration neighborhood search. *In [Johnson and Trick, 1996]*, page Distributed coloration neighborhood search, 1996.

[Moscato and Cotta, 2003] cited page 72
P. Moscato and C. Cotta. *A Gentle Introduction to Memetic Algorithms. In F. Glover and G. A. Kochenberger (Eds.), Handbook of Metaheuristic.* Kluwer, Norwell, Massachusetts, USA,, 2003.

[Moscato, 1989] cited page 17
P. Moscato. On evolution, search, optimization algorithms and martial arts: Towards memetic algorithms. Technical Report Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, 1989.

[Motzkin and Straus, 1965] cited page 18
T.S. Motzkin and E.G. Straus. Maxima for graphs and a new proof of a theorem of turán canad. *Canadian Journal of Mathematics*, 17:533–540, 1965.

[Moukrim *et al.*, 2010] cited page 127, 131, 132, 134, 136
A. Moukrim, K. Sghiouer, C. Lucet, and Y. Li. Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:663–670, 2010.

[Murthy *et al.*, 1994] cited page 17
A.S. Murthy, G. Parthasarathy, and V.U.K. Sastry. Clique finding - a genetic approach. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 18–21. IEEE Press, 1994.

[Neri *et al.*, 2012] cited page 78
F. Neri, C. Cotta, and P. Moscato. *Handbook of Memetic Algorithms. Studies in Computational Intelligence 379.* Springer, Berlin, 2012.

[Östergård, 2001] cited page 47
P.J.R. Östergård. A new algorithm for the maximum weight clique problem. *Nordic Journal of Computing*, 8:424–436, 2001.

[Östergård, 2002] cited page 10, 10, 100
P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120:197–207, 2002.

[Palubeckis, 2007] cited page 24, 25, 25, 71, 71, 71, 71, 80, 81, 82, 86
G. Palubeckis. Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, 189(1):371–383, 2007.

[Pardalos and Xue, 1994] cited page 20
P.M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

[Park and Carter, 1994] cited page 17
K. Park and B. Carter. On the effectiveness of genetic search in combinatorial optimization. Technical Report BU-CS-94-010, Department of Computer Science, Boston University, 1994.

[Park *et al.*, 1996] cited page 71
K. Park, K. Lee, and S. Park. An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 95:671–682, 1996.

[Pavan and Pelillo, 2007] cited page 18
M. Pavan and M. Pelillo. Dominant sets and pairwise clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):167–172, 2007.

[Plumettaz *et al.*, 2010] cited page 120
Plumettaz, D. Schindl, and N. Zufferey. Ant local search and its efficient adaptation to graph colouring. *Journal of Operational Research Society*, 61(5):819–826, 2010.

[Porumbel *et al.*, 2010a] cited page 79, 123
D.C. Porumbel, J.K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10):1822–1832, 2010.

[Porumbel *et al.*, 2010b] cited page 95, 102, 103, 103, 104, 105, 105, 105, 118, 120, 123, 132
D.C. Porumbel, J.K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10):1822–1832, 2010.

[Porumbel *et al.*, 2010c] cited page 95
D.C. Porumbel, J.K. Hao, and P. Kuntz. A search space cartography for guiding graph coloring heuristics. *Computers and Operations Research*, 37(4):769–778, 2010.

[Pullan and Hoos, 2006] cited page 13, 13, 16, 23, 35
W. Pullan and HH. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[Pullan *et al.*, 2011] cited page 17, 23, 62
W. Pullan, F. Mascia, and M. Brunato. Cooperating local search for the maximum clique problem. *Journal of Heuristics*, 17:181–199, 2011.

[Pullan, 2006] cited page 13, 13, 16, 23, 37, 38, 38, 47, 54, 62
W. Pullan. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12(3):303–323, 2006.

[Pullan, 2008] cited page 16, 23, 24, 47, 54, 54, 55, 57, 58, 59
W. Pullan. Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14:117–134, 2008.

[Ramanujam and Sadayappanv, 1988] cited page 19
J. Ramanujam and P. Sadayappanv. Optimization by neural networks. *IEEE International Conf. on Nerual Networks*, pages 325–332, 1988.

[Ravi *et al.*, 1994] cited page 71
S.S. Ravi, D.J. Rosenkrantz, and G.K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[Rechenberg, 1973] cited page 17
I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien-der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

[Reigold *et al.*, 1977] cited page 20, 20
E.M. Reigold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Englewwod Cliffs, 1977.

[Robson, 1986] cited page 10
J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[Sakamoto *et al.*, 1997] cited page 17
A. Sakamoto, X. Liu, and T. Shimamoto. A genetic approach for maximum independent set problems. *IEICE Trans. Fundamentals*, E80-A(3):551–556, 1997.

[Schaeffer, 2007] cited page 23
S.E. Schaeffer. Graph clustering. *Computer Science Review*, 1:27–64, 2007.

[Sewell, 1996] cited page 107
E.C. Sewell. An improved algorithm for exact graph coloring. *In [Johnson and Trick, 1996]*, pages 359–376, 1996.

[Silva *et al.*, 2004] cited page 24, 71
G.C. Silva, L.S. Ochi, and S.L. Martins. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In *Experimental and Efficient Algorithms, Lecture Notes in Computer Science*, pages 498–512, Berlin, 2004. Springer.

[Silva *et al.*, 2007] cited page 71
G.C. Silva, M.R.Q. De Andrade, L.S. Ochi, S.L. Martins, and A. Plastino. New heuristics for the maximum diversity problem. *Journal of Heuristics*, 13(4):315–336, 2007.

[Singh and Gupta, 2006] cited page 17, 23, 37, 38
A. Singh and A. K. Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12:5–22, 2006.

[Sloane, 1998] cited page 20
N. J. A. Sloane. Unsolved problems in graph theory arising from the study of codes. *Graph Theory Notes of New York*, 18:11–20, 1998.

[Smith *et al.*, 1998] cited page 95
D.H. Smith, S. Hurley, and S.U. Thiel. Improving heuristics for the frequency assignment problem. *European Journal of Operational Research*, 107(1):76–86, 1998.

[Solnon and Fenet, 2006] cited page 19
C. Solnon and S. Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12:158–180, 2006.

[Sorensen, 2004] cited page 71, 71
M. Sorensen. New facets and a branch-and-cut algorithm for the weighted clique problem. *European Journal of Operational Research*, 157:54–70, 2004.

[Strickland *et al.*, 2005] cited page 20
D.M. Strickland, E. Barnes, and J.S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53:389–402, 2005.

[Tarjan and Trojanowski, 1977] cited page 10
R. E. Tarjan and A. E. Trojanowski. Finding a maximum independent set. *SIAM Journal of Computing*, 6:537–546, 1977.

[Thierens, 1999] cited page 17
D. Thierens. Scalability problem of simple genetic algorithms. *Evolution Computing*, 7:331–352, 1999.

[Titiloye and Crispin, 2011] cited page 118, 119, 120
O. Titiloye and A. Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

[Titiloye and Crispin, 2013] cited page 118, 119
O. Titiloye and A. Crispin. Parameter tuning patterns for random graph coloring with quantum annealing. *Accepted to Plos One*, 2013.

[Tomita and Kameda, 2007] cited page 11
E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37:95–111, 2007.

[Tomita and Seki, 2003] cited page 11
E. Tomita and T. Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Proc. Discrete Mathematics and Theoretical Computer Science. LNCS 2731*, pages 278–289, 2003.

[Tomita *et al.*, 1988] cited page 12
E. Tomita, S. Mitsuma, and H. Takahashi. Two algorithms for finding a nera-maximum clique. Technical report, UEC-TR-C1, 1988.

[Vingron and Argos, 1991] cited page 20
M. Vingron and P. Argos. Motif recognition and alignment for many sequences by comparison of dot-matrices. *Journal of Molecular Biology*, 218:33–43, 1991.

[Vingron and Pevzner, 1992] cited page 20, 20
M. Vingron and P.A. Pevzner. Multiple sequence comparison and n-dimensional image reconstruction. *Working paper*, 1992.

[Wang *et al.*, 2012] cited page 25, 25, 71, 81, 82, 82, 82, 82, 86, 86
J. Wang, Y. Zhou, Y. cai, and J. Yin. Learnable tabu search guided by estimation of distribution for maximum diversity problems. *Soft Computing*, 16:711–728, 2012.

[Werra *et al.*, 1999] cited page 95
D. de Werra, C. Eisenbeis, S. Lelait, and B. Marmol. On a graph-theoretical model for cyclic register allocation. *Discrete Applied Mathematics*, 92(2–3):191–203, 1999.

[Wimer *et al.*, 1998] S. Wimer, R.Y. Pinter, and J. Feldman. Optimal chaining of cmos transistors in a functional cell. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):795–801, 1998.

[Wolpert and Macready, 1997] cited page 16
D. Wolpert and G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[Wu and Hao, 2011a] cited page 2, 13, 15, 23, 27, 62
Q. Wu and J.K. Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Accepted to Journal of Combinatorial Optimization*, 2011.

[Wu and Hao, 2011b] cited page 111, 113, 118, 119
Q. Wu and J.K. Hao. An extraction and expansion approach for graph coloring. *Accepted to Asia-Pacific Journal of Operational Research*, 2011.

[Wu and Hao, 2012a] cited page 3, 21, 25, 44, 93, 113, 118, 118, 118, 118, 119
Q. Wu and J.K. Hao. Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2):283–290, 2012.

[Wu and Hao, 2012b] cited page 23, 25, 44, 125, 136
Q. Wu and J.K. Hao. An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600, 2012.

[Wu and Hao, 2012c] cited page 3, 69
Q. Wu and J.K. Hao. A hybrid metaheuristic method for the maximum diversity problem. *submitted European Journal of Operational Research*, 2012.

[Wu and Hao, 2012d] cited page 125
Q. Wu and J.K. Hao. Improved lower bounds for sum coloring via clique decomposition. *submitted to Discrete Applied Mathematics*, 2012.

[Wu and Hao, 2012e] cited page 72
Q. Wu and J.K. Hao. A memetic approach for the max-cut problem. In *C. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.): 12th International Conference on Parallel Problem Solving From Nature, Lecture Notes in Computer Science 7492*, pages 297–306, Taormina, Italy, 2012. Springer.

[Wu and Hao, 2012f] cited page 2, 13, 13, 15, 23, 45
Q. Wu and J.K. Hao. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, 2012.

[Wu and Hao, 2013] cited page 72, 77
Q. Wu and J.K. Hao. Memetic search for the max-bisection problem. *Computers & Operations Research*, 40(1):166–179, 2013.

[Xie and Liu, 2009] cited page 95, 103, 104, 105, 105, 118, 120
X.F. Xie and J. Liu. Graph coloring by multiagent fusion search. *Journal of Combinatorial Optimization*, 18(2):99–123, 2009.

[Zhang *et al.*, 2005] cited page 18, 23
Q. Zhang, J. Sun, and E. Tsang. An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2):192–200, 2005.

[Zufferey *et al.*, 2008] cited page 95
N. Zufferey, P. Amstutz, and P. Giaccari. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, 11(4):263–277, 2008.

l'université nantes angers le mans

PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

université angers

Thèse de Doctorat

Qinghua WU

The maximum clique problems with applications to graph coloring

Problèmes de clique maximum avec applications à la coloration de graphe

Résumé
Le problème de la clique maximum (MCP) est un problème d'optimisation combinatoire important avec un large éventail d'applications pratiques dans de nombreux domaines, y compris la recherche d'information, l'analyse de la transmission du signal, la théorie de la classification, l'économie, la planification et l'ingénierie biomédicale. En outre, un certain nombre de problèmes d'optimisation combinatoire sont étroitement liés au MCP, tels que la coloration de graphe, la somme coloration, réglez détermination du gagnant emballage et optimale. Cette thèse est consacrée à l'élaboration d'approches heuristiques efficaces pour s'attaquer au problème de la clique maximum et ses généralisations. Pour atteindre cet objectif, nous avons développé une approche de recherche tabou adaptative multistart pour le problème de clique maximum classique, un algorithme recherche tabou multi-voisinage pour la clique maximum de sommets pondérés, et une méthode métaheuristique hybride pour le problème de la clique maximum d'arêtes pondérés. En outre, nous appliquons ces méthodes heuristiques développées pour résoudre ces problèmes difficiles qui sont étroitement liés au problème de la clique maximum. Tous les algorithmes sont mis en œuvre et testés avec succès sur un certain nombre de cas de référence provenant de divers domaines d'application. Les méthodes proposées concurrencent favorablement les autres approches de l'état de l'art.

Mots clés
Problème de la clique maximum, Problème de coloration de graphe, Problème de Somme coloration, Recherche tabou, Contraint de voisinage.

Abstract
The maximum clique problem (MCP) is an important combinatorial optimization problem with a wide range of practical applications in numerous fields, including information retrieval, signal transmission analysis, classification theory, economics, scheduling, and biomedical engineering. Moreover, a number of combinatorial optimization problems are tightly related to MCP, such as graph coloring, sum coloring, set packing and optimal winner determination. This thesis is devoted to developing effective heuristic approaches to tackle the maximum clique problem and its generalizations. To achieve this, we developed an adaptive multistart tabu search approach for the classic maximum clique problem, a multi-neighborhood tabu search algorithm for the maximum vertex weight clique and a hybrid metaheuristic method for the maximum edge weight clique problem. Moreover, we apply these developed heuristic approaches to solve these hard problems which are tightly related to the maximum clique problem. All algorithms are implemented and successfully tested on a number of benchmark instances from diverse application areas. The proposed methods favorably compete with other state-of-art approaches.

Key Words
Maximum clique problem, Graph coloring problem, Sum coloring problem, Tabu search, Constrained neighborhood.